SerialWatcher

Table des matières

Contenu

Introduction

Dans le cadre du projet, nous avions pour objectif de mettre en pratique un framework PHP. Pour ce faire, nous avons décidé de faire une application web qui a pour but de traiter des séries.

Celle-ci a été grandement inspirée d'une application existante : TVshowTime. Le but était de reprendre des fonctions utiles et de les développer par nous-mêmes. Au final, l'application permettra de s'identifier en tant qu'utilisateur et de s'abonner à différentes séries que l'on pourra rechercher depuis la page d'accueil. Le client pourra également consulter les épisodes et saison de toute série et d'indiquer quels épisodes il a déjà vu. Cette dernière option permet, notamment, de gérer une vue qui indique à l'utilisateur les épisodes qu'il n'a pas encore regardé et qui sont sortis pour chaque série à laquelle il est abonné. De plus, une vue "calendrier" sera également disponible afin d'afficher la date de sortie des épisodes à venir. Pour finir, une interface de gestion de profil sera à disposition pour modifier ses données personnelles.

Pour la gestion du back-end, nous avons utilisé le framework PHP laravel, le système mis à disposition par laravel eloquen pour la gestion des models et angularjs pour le front-end. Les choix concernant ces technologies seront expliqués plus loin.

Nous prenons en considération que le développeur connaît le Framework Laravel ainsi que le Framework AngularJS.

Prérequis

- PHP : **v7.0.6**
- Composer : Permet de gérer toutes les dépendances php.
- Une base de données mysql pour utiliser le projet. Exemple « serialwatcher ».

Installation

1. Cloner le projet.
2. Déplacez-vous à la racine du projet
   a. **Composer install** : cela va installer toutes les dépendances
3. Copier le fichier .env.example et nommer le « .env »
   a. **cp .env.example .env**
4. Générer la clef api Laravel
   a. **php artisan key:generate**
5. Ensuite éditer le fichier .env afin que les données de connexion à la base de données correspondent à votre configuration.
   a. DB_HOST : nom du host (localhost)
   b. DB_DATABASE : nom de la base de données (serialwatcher)
   c. DB_USERNAME : Nom d'utilisateur de la base de données.
   d. DB_PASSWORD : Mot de passe de la base de données. (laisser vide si pas de mot de passe).
6. Importer la base de données
   a. **php artisan migrate**
7. Importer les données de tests
   a. **php artisan db:seed**

Exécuter le projet

Maintenant que vous avez télécharger toutes les dépendances, créer la base de données et configurer la web application vous pouvez maintenant lancer le serveur.

php artisan serve –port 8082

Vous pouvez accéder à l'application via l'url : http://localhost:8082/

Pour accéder à l'api : http://localhost:8082/api/

Utilisateurs de test :

Pseudo	Password
Modjo	secret
Hacker	secret
Manu	blabla

External API

Dans le cadre de ce projet, nous utilisons l'api "omdbapi" (http://www.omdbapi.com/) qui pointe sur la grande base de données IMDB (http://www.imdb.com/). Celle-ci permet de faire des recherches selon un imdbID ou encore selon un mot compris dans le nom de la série.

Les requêtes principales utilisées

/s={string}	retourne 10 séries qui contiennent le mot {string}
/s={string}*	retourne 10 séries qui contiennent un mot commençant par {string}
/i={id}	retourne la série ou l'épisode correspondant à l'identifiant IMDB {id}
/i={id}&season={nb}	retourne la saison numéro {nb} de la série correspondant $ {id}

Choix techniques

Back-end

Pour ce projet nous utilisons le framework Laravel en tant que *RestFull Service*. Pour tous ce qui est des appels à notre base de données nous utilisons *Eloquent*. Les données retournée par le *RestFull Service* sont sous format *JSON*. Toutes les routes liées aux services commencent par */api/….* Ceci permet de faire la distinction entre les routes Laravel (RESTfull) et les routes du Framework AngularJS.

Front-end

Afin d'afficher les pages de l'application web, nous utilisons le Framework AngularJS. Laravel charge uniquement le fichier index.php qui contient toute les dépendances au Framework AngularJS. C'est-à-dire que la vue de ce projet est indépendante du Framework Laravel. La communication entre la vue est le Framework Laravel répond uniquement via l'appel de routes qui renvoient les éventuelles données sous format *JSON*.

Librairies front-end utilisées

Ce projet utilise plusieurs librairies.

- **Moment.js** : Permets de formater, valider, afficher des dates / heures.
- **CLNDR.js** : C'est cette librairie qui génère le calendrier de la page calendar.
- **Modernizr.js** : Permet d'assurer la rétrocompatibilité du HTML, css, JavaScript, entre les différents explorateurs web.
- **Satellizer.js** : Module permettant de gérer l'authentification par token.
- **Spin.js** : Permet de générer dynamiquement un spinner pour les requêtes AJAX.
- **Toucheffects.js** : Permet de gérer l'action de :hover dans la home page avec les différentes vignettes des séries

Authentification

Pour l'authentification nous avons choisis d'utiliser *jwt-auth*. Il permet d'effectuer une authentification à l'aide d'un jeton stocké sous format JSON (JSON Web Token Authentication).

Toute la logique pour l'authentification est située dans le controller *AuthenticateController*.

Pour protéger une route il suffit d'ajouter le middleware : **jwt.auth**. Seul l'utilisateur possédant un jeton valide pourra y accéder.

Avantage de l'authentification par token JSON:

Permet la réutilisation du même token dans plusieurs domaines. L'utilisateur peux accéder à plusieurs applications web avec le même token.

Protection contre les attaques cross-site request forgery (CSRF) (Valable uniquement si la web application a été conçue pour être protégée contre ce type d'attaque). Mais nécessite quand même l'utilisation de https afin de protéger l'envoie du token.

Meilleure performance : Le serveur doit uniquement calculer la valeur du jeton. Il n'a pas besoins contrairement à un cookie de désérialiser la session à chaque demande.

Structure du projet

Cette partie donne un descriptif très bref de l'arborescence du projet.

Back-end

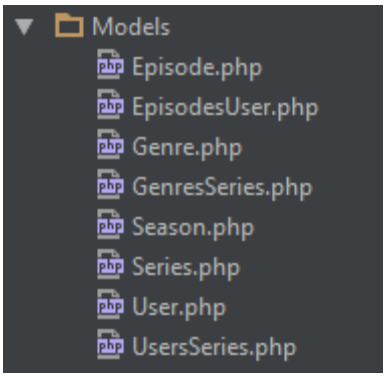
L'application étant basée sur le modèle MVCS, notre back-end est essentiellement composé de 3 parties : les controllers et ses routes, les services et les models.

Description du contenu du dossier App :

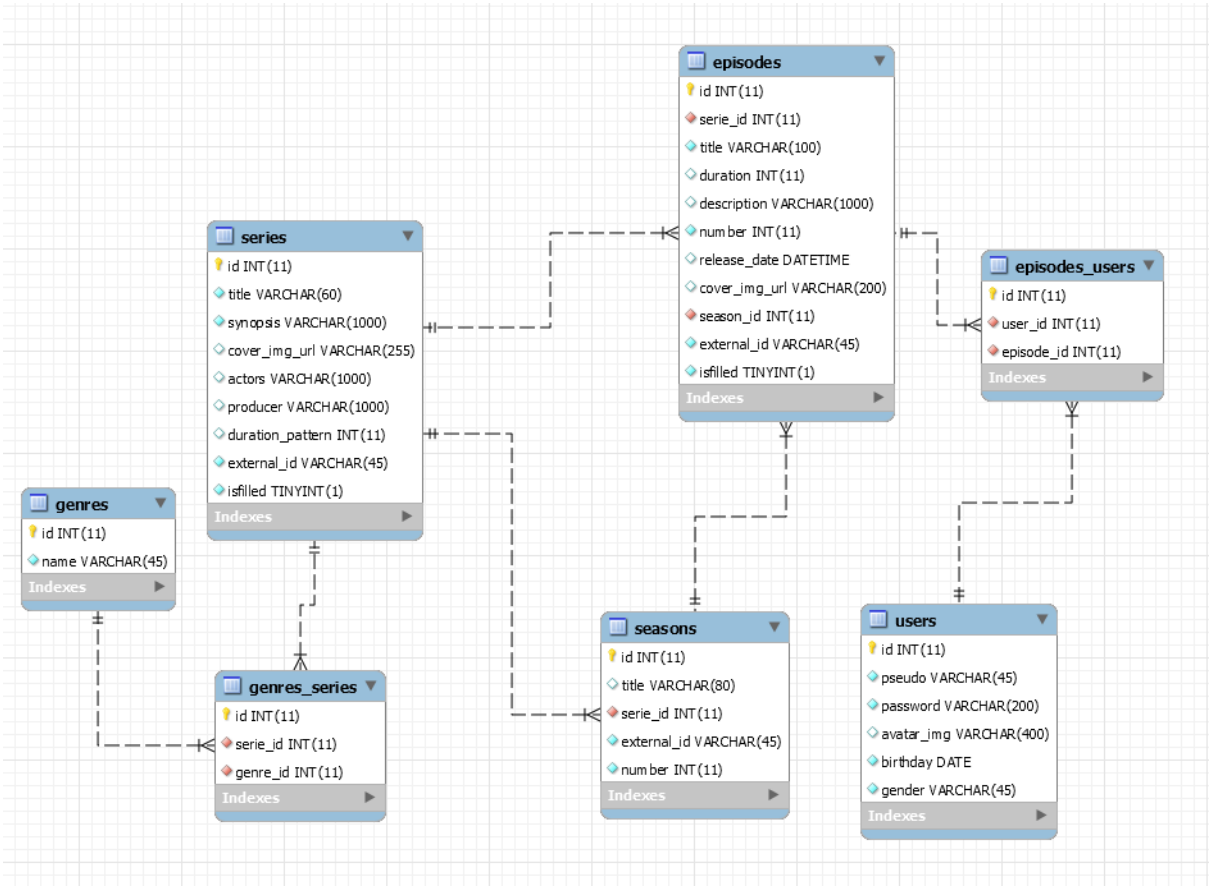
- Contracts
  - Toutes les interfaces de notre application
- Exceptions
  - Toutes les exceptions liées à l'application.
- http
  - Controllers
    - Contient tous les controllers qui sont directement associés aux routes.
  - Middleware
    - Contient tous les middlewares utilisés dans les routes Laravel
  - Services
    - Cette partie permet de faire le lien entre les Models et les Controllers. Il contient aussi un service qui permet d'uniformiser les JSON envoyés.
  - Utils
    - Contient toutes les fonctions utilitaires nécessaires à l'application. Permet entre autres d'extraire des nombres, convertir des dates, effectuer des appels à une API externe, mapper les résultats de l'API externe.
- Routes.php
  - Contient toutes les routes RESTfull de l'application.
- Models
  - Contient tous les modèles représentant nos objets métier.
- Resources
  - Views
    - Contient le fichier index.php, qui est chargé par défaut si l'utilisateur ne demande aucune route connue par l'application (des routes commençant par /API/…). Ce fichier permet de charger toutes les dépendances liées à AngularJS.
- Public
  - Css
    - Contient tous les fichiers css pour le design du front-end.
  - Img
    - Dossier contenant toutes les images.

- Scripts
  - Contient tous les scripts AngularJS ainsi que toutes les librairies.
- Views
  - Contient tous les templates utilisés par AngularJS.

Les models



Modèle de base de données :



Episode

Les épisodes contiennent deux éléments important : le flag "isfilled" qui indique si l'épisode a été pleinement développé. En effet, lors de la recherche d'une saison de série, depuis l'api, nous avons déjà des informations sur les épisodes, ce qui nous évite de faire une requête par épisode au moment de la génération de la saison. Cette requête n'est faite que lorsque l'on rentre dans le détail d'un épisode et la base de données est mise à jour à ce moment-là.

Le deuxième élément est le "serie_id" qui est présent pour faciliter grandement l'algorithme de génération des vues "ToWatch" et "Calendar".

EpisodesUser

Cette table a pour but de lier l'utilisateur à chaque épisode pour indiquer lesquels il a vu.

GenresSeries

Table mise à disposition pour les futures recherches par catégorie, en plus du titre.

Season

Table faisant le lien entre épisodes et séries, cette information devient surtout utile pour la fiche de série, afin d'indiquer les épisodes de chaque saisons.

Series

Table contenant simplement les informations de la série.

User

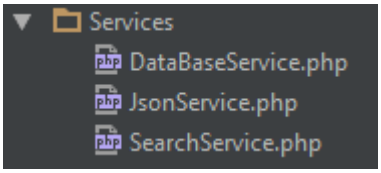
Table contenant les informations personnelles de l'utilisateur.

UsersSeries

Table de liaison qui permet d'indiquer toutes les séries que l'utilisateur suit. L'information depuis la table "EpisodesUser" n'aurait pas été suffisante dans le cas où l'utilisateur suit une série dans laquelle il n'a regardé aucun épisode.

Les services

Une couche service a été ajoutée au modèle de base, afin d'y placer les méthodes qui traitent les informations entre les controllers et les models, lorsque c'est nécessaire. L'application est liée à une base de données mais également à une API externe, ce qui a rendu cette couche obligatoire.



DataBaseService

Contient les méthodes de liason à la base de données, notamment pour les requêtes spécifiques ou encore lorsqu'il faut la remplir.

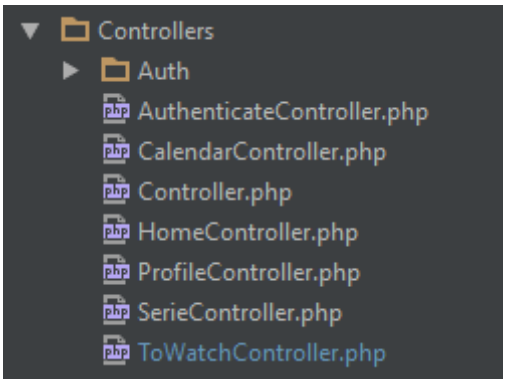
JsonService

Contient les méthodes de traitement des JSON récupérés depuis l'API.

SearchService

Contient toutes les méthodes faisant appel à une api externe (ici, OMDB).

Les controllers



AuthenticateController

Ce controller gère toute la partie "login" de l'utilisateur et permet de le récupérer depuis les autres fonctionnalités.

CalendarController

Contient la méthode getCalendarSubs qui va retourner un Json composé de toutes les informations relatives aux épisodes qui proviennent des séries auxquelles l'utilisateur s'est abonné.

HomeController

Contient les méthodes getFeaturedSeries qui sont les 10 dernières séries entrées dans la database et getFavouritesSeries qui sont les 10 premières séries de l'utilisateur connecté.

ProfileController

Lié à l'affiche de profil utilisateur, il contient la méthode permettant de modifier ses informations ainsi que celles permettant de récupérer les séries qu'il suit.

SerieController

Le controller principal qui contient toutes les méthodes liées aux séries, lors de la recherche, de l'affichage des informations ou encore des interactions avec l'utilisateur.

ToWatchController

Gère la vue "ToWatch" qui consiste à afficher à l'utilisateur tous les épisodes sortis qu'il n'a pas encore vu.

Les routes

Method	URI	Description	Attributes	Returned value	Middleware
POST	api/authenticate	Get the Token of the current user	{user} : with pseudo and password	{String} : JSON Web Tokens	-
GET	api/authenticate/user	Get the user object without password {user}	-	{User}	-
GET	api/calendar	Get an array of episodes unreleased yet from the series that the current user follows.	-	Array of Episodes	jwt.auth
POST	api/episode/seen/{id}/{seen}	Update database if the current user has seen the given episode.	{id} : database episode ID {seen} : boolean	-	jwt.auth
GET	api/episode/{id}	Get full data of an episode.	{id} : database episode ID	Episode	-
GET	api/episodes/seen/{idseason}	Get the episodes seen by the current user in the given season.	{idseason} : database season ID	Array of Episodes	jwt.auth
GET	api/favourites	Get the 10 favourites series of the current user	-	Array of Series	jwt.auth
GET	api/featured	Get the 10 featured	-	Array of Series	-

		series of the database			
GET	api/profile/personal	Get personal data of the current user.	-	User	jwt.auth
POST	api/profile/personal	Update personal data of the current user by the form informations.	-	-	jwt.auth
GET	api/profile/subscriptions	Get all series followed by the current user.	-	Array of Series	jwt.auth
GET	api/search/{string}	Get a list of series from the external api matching the given string	{string} : string to match titles	Array of Series	-
GET	api/serie/filled/{id}	Get a serie from its database ID and fill database with its full information	{id} : database serie ID	Series	-
GET	api/serie/{id}	Get a serie from its database ID	{id} : database serie ID	Series	-
POST	api/signup	Create a new user	-	-	-
POST	api/serie/subscribe/{id}	Add the given serie to the current user subscriptions	{id} : database serie ID	-	jwt.auth
GET	api/serie/subscribed/{id}	Check if the current user follows the given serie	{id} : database serie ID	Boolean	jwt.auth
GET	api/towatch	Return all episodes that the current user hasn't seen and that belongs to the series followed by him	-	Array of Series	jwt.auth

POST	api/serie/unsubscribe/{id}	Remove the given serie to the current user subscriptions	{id} : database serie ID	-	jwt.auth

Les exceptions

Toutes les exceptions levées dans Laravel sont transmise sous format JSON uniquement si la requête a été exécutée en AJAX. Le fichier *app/Exceptions/Handler.php* définit ce cas de figure. Ceci permet à notre vue AngularJS de recevoir les erreurs de Laravel.

AngularJS – Front-end

Tous les fichiers nécessaires au fonctionnement du front-end sont stockés à deux endroits. Le premier est le dossier *Resouces/Views*, qui contient le fichier index.php et qui charge toutes les dépendances (Scripts / CSS) liées au front-end. Le second est le dossier **Public** qui contient tous les fichiers (JavaScript / Librairie / CSS).

Description du dossier public :

Dossier scripts :

- **Controllers** : Ce dossier contient tous les controllers angular. Chaque controller est lié à un template.
- **Filters** : Contient tous les filtres angular. Pour le moment il n'y a qu'un seul filtre. Ce filtre permet de formater une date (utilise moment.js).
- **Lib** : Contient toutes les librairies (ainsi que le Framework AngularJS) utilisées par la vue.
- **Directives**: Contient toutes les directives utilisées dans le projet (nav-bar ...).
- **Services** : Il s'agit de tous les services utilisés par les controllers. Ces services permettent par exemple de s'authentifier, d'effectuer des appels AJAX …
- **App.js** : Fichier principal, il définit l'application Angular principale nommée *serialWatcherApp*. Il contient toutes les routes (AngularJS) ainsi que toute la configuration liée à AngularJS.

Description des services :

Les services sont directement appelés par les Controllers.

Voici la liste de tous les services ainsi qu'une brève description de leurs fonctionnalités.

- **search** : Permet de rechercher une série en fonction de son titre.
- **seriesService** : Permet de récupérer toutes les informations concernant une série (saisons & épisodes).
- **homeService** : Permet de récupérer les séries favorites ou en vedette.
- **calendarService** : Permet de récupérer une liste d'épisodes pour les séries dont l'utilisateur est abonné et qui ne sont pas encore sortis.
- **cacheService** : Fonctionne comme un HashMap. Il permet de stocker des valeurs dans le localStorage.
- **subscribeService** : Permet de s'abonner ou de se désabonner d'une série. Il permet également de déterminer si l'utilisateur et abonné.
- **episodeService** : Permet de récupérer une liste des épisodes que l'utilisateur a déjà vus pour une saison donnée. Il permet aussi de définir si l'utilisateur a vu ou non un épisode.
- **profileService** : Permet de récupérer toutes les informations liées à l'utilisateur authentifié. Il permet aussi de mettre à jour ses informations.
- **toWatchService** : Permet de récupérer tous les épisodes que l'utilisateur authentifié n'a pas encore vus selon les séries qu'il suit
- **signUpService** : Permet de créer des nouveaux utilisateurs. (il faut obligatoirement le pseudo, le mot de passe, la date de naissance (YYYY-MM-DD) et le genre).
- **authenticateSevice** : Ce service permet de récupérer les informations d'authentification, c'est lui qui récupère le token et qui définit la variable user dans le localStorage (uniquement pour la première connexion).
- **modalService**: Permet de This is the main controller for all modal .

Description des Controllers :

HomeController

- **Fichier**: homeController.js
- **Template(s) associé(s)** : homeView.html
- **Route(s) associée(s)** : /

AuthController

Permet d'effectuer l'authentification de l'utilisateur à l'aide de son pseudo et son mot de passe. Il contrôle principalement la vue de login.

- **Fichier** : authController.js
- **Template(s) associé(s)**: authView.html
- **Route(s) associée(s)** : /auth

CalendarController

Affiche un calendrier des futures sorties concernant les séries que l'utilisateur suit.

- **Fichier** : calendarController.js
- **Template(s) associé(s)** : calendarView.html
- **Route(s) associée(s)** : /calendar

LandingController

Affiche la liste de la série recherchée par l'utilisateur. La barre de recherche se situe dans la nav bar. Lorsque l'utilisateur clique sur une série il est automatiquement redirigé sur une autre vue : */single*.

- **Fichier**: landingController.js
- **Template(s) associé(s)** : searchLandingView.html
- **Route(s) associée(s)** : /landing

NavController

Contrôle tous les événements liés à la barre de navigation. Affiche ou non certains boutons si l'utilisateur est authentifié. Il permet aussi à l'utilisateur de rechercher des séries.

- **Fichier**: navController.js
- **Template(s) associé(s)** : navBar.html
- **Route(s) associée(s)** : -

ProfilController

Charge toutes les données liées au profil de l'utilisateur. Il permet de modifier / mettre à jour ces informations.

- Fichier: profilController.js
- **Template(s) associé(s)** : profilView.html
- **Route(s) associé(s)** : /profil

SignUpController

Ce controller permet d'ajouter de nouveaux utilisateurs.

- **Fichier**: signUpController.js
- **Template(s) associé(s)** : signUpView.html
- **Route(s) associé(s)** : /signup

SingleController

Ce controller charge toutes les informations liées à une série (charge toutes les saisons ainsi que tous les épisodes). Il permet aussi à l'utilisateur de définir quels épisodes il a déjà vus en fonction de la saison.

- **Fichier**: singleController.js
- **Template(s) associé(s)** : singleView.html
- **Route(s) associée(s)** : /single

ToWatchController

Ce controller charge une liste de tous les épisodes que l'utilisateur n'a pas encore vus pour les séries auxquelles il est abonné.

- **Fichier**: towatchController.js
- **Template(s) associé(s)**: toWatchView.html
- **Route(s) associée(s)**: /towatch

ModalController

Il s'agit du contrôleur principal pour toutes les modales. Il gère toutes la logique ainsi que toutes les variables utilisées par les modals.

- **Fichier**: modalController.js
- **Template(s) associé(s)** : modalContent.html

Description des routes angular

Les routes angular servent d'intermédiaires entre les Controllers et les Views.

C'est dans ces routes-là que l'on va définir les URL que l'utilisateur va voir dans sa barre de recherche. C'est également ici que sont définis les templates (views) ainsi que les controllers assignés à chaque route.

Description des Views

- Une Barre de navigation qui contient toutes les informations relatives au menu (liens internes, possibilité de recherche)
- Une vue d'authentification qui permet à l'utilisateur de se connecter.
- Une vue D'inscription qui permet à l'utilisateur de s'inscrire à l'application
- Une vue de type Homepage qui contient plusieurs propositions de series
- La vue profile qui contient toutes les informations relatives au profil de l'utilisateur connecté
- Une vue de recherche qui va afficher les résultats d'une recherche
- Une Single view pour les détails d'une série
- Une vue Towatch qui correspond à tous les épisodes que l'utilisateurs doit voir selon ses abonnements
- Une vue calendrier où l'utilisateur peut consulter les prochaines sorties de ses abonnements

Problèmes rencontrés et solutions trouvées

- Problème de performance lors de la recherche d'une série pas encore connue de notre base de données.
- Problème de performance lors du remplissage d'une série (charge les saisons ainsi que les épisodes de celle-ci).

Evolutions futures

Techniques :

- Meilleure gestion des erreurs entre AngularJS et laravel
- Sécuriser la gestion du profil utilisateur. A l'heure actuelle, on peut introduire des informations farfelues.
- Optimisation des requêtes effectuée avec Eloquent.
- Optimisation du design et Responsiveness de l'application.
- Sécurisation de l'application avec le protocole https.

Fonctionnelles :

- Développer un batch qui vient mettre à jour la base de données régulièrement en vues des changements sur l'API.
- Optimisation de la gestion des épisodes (vues/pas vues) de manière à ce qu'un utilisateur puisse avoir le choix lorsqu'il clique sur un épisode, de dire qu'il a vu tout les épisodes précédents (cocher 25 épisodes en 1 clic par exemple)
- Améliorer la vue Calendar afin que celle-ci puisse afficher plus d'informations concernant les épisodes.
- Affichage d'une fiche d'infos d'un épisode
- Implémenter un système de commentaires et de rating

Conclusion

Ce projet a deux grandes qualités que l'on peut mettre en avant. Premièrement, il est très peu dépendant de l'api externe utilisé pour générer nos informations. Ceci a été possible grâce à l'utilisation du modèle MVCS. Deuxièmement, le découpage entre le front-end et le back-end est fait de telle sorte à ce que le back-end puisse être utilisé par une autre vue étant un système RESTfull service.

Ces deux éléments apportent une plus-value à la montée en compétences que l'on a déjà effectuée grâce à l'environnement Laravel. De plus, le projet implémente aussi les technologies Eloquent et AngularJS qui s'ajoutent à l'expérience que nous a apporté notre projet.