Documentation de l’API WEB pour Hearthstone Collector

# Description générale de l’API

Cette API propose des services web que viendra interroger l’application Android « Hearthstone Collector ».

Les fonctionnalités nécessaires au projet sont disponibles dans la liste des fonctionnalités (fichier .txt)

Le choix de la technologie s’est porté sur Symfony dans sa version 4. Symfony est un framework PHP puissant permettant une découpe facile du code en modèle MVC. Avec des gains de performances et l’ORM (Object Relational Mapper) « Doctrine », cet outil permet de découper le code en différentes routes pour différents traitements, rendant le concept d’API Web plus simple.

Nous utiliserons aussi le bundle Symfony « JMS Serializer » pour la sérialisation et la désérialisation d’objets JSON ou d’entités (le modèle persistera en objet JSON ou PHP).

# Le MVC avec Symfony

Symfony découpe le code sous le modèle MVC. Les sources d’un projet Symfony peuvent se retrouver dans le dossier « src ». Ce dossier comporte une structure propre à Symfony :

* Controller : Dossier des fichiers « Controller », qui gèrent tous les traitements entre « Modèle » et « Vue » (les vues ne seront pas utilisées sur ce projet).
* Entity : Fichiers « Modèle » du projet. Ici sont stockés tous les fichiers de données, en relation avec la base de données via l’ORM Doctrine.
* Migrations : Doctrine > Fichiers créés par la commande « php bin/console make:migration »
* Repository : Fichiers de base de données, pour écrire des requêtes SQL particulières (en plus des requêtes de base proposées par Doctrine)
* Service : Fichiers « service » utilisés pour interroger d’autres API externes (comme celle d’Hearthstone ou de Facebook).

# Fichiers utiles

* /.env : fichier de configuration de l’**environnement**. S’assurer qu’il soit bien en mode « dev » lors du développement, et « prod » lors de la mise en production
* /composer.json : contient les **bundles** et **dépendances** nécessaires au projet. Des commandes sont disponibles via le fichier **composer.phar** (exécuté via php)
* /config/bundles.php : chargement des bundles Symfony : il faut y ajouter manuellement les bundles téléchargés, ou composer peut le faire lors de l’installation
* /config/packages/doctrine.yaml : configuration de la **base de données**
* /public/index.html : fichier chargé en **premier**. Il injecte le reste du projet via le kernel.
* /var/ : dossier utile possédant le cache (à vider de temps en temps ou via la commande) et les logs (utile aussi)
* /vendor/ : dossier avec toutes les dépendances de composer. Non géré par le versioning de Git.

# Commandes

Toutes les commandes sont disponibles en exécutant la commande suivante à la racine du projet :

**php bin/console**

**Attention :** Si tous les bundles ne s’affichent pas (make, start, …) c’est que la console croit être en environnement de production. Rajouter **« --env=dev »** à la fin de chaque commande si tel est le cas.

# Créer une entité

Avec la commande « doctrine:entity:create » il est possible de créer une entité modèle, et donc sa table dans la base de données. Il faut renseigner les champs, et leur type (type primitif ou entité pour les relations)

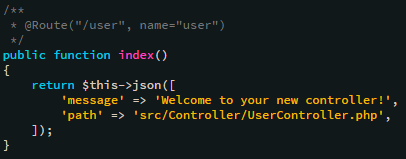
En effet, Doctrine gère bien les relations entres entités, et permet donc facilement les ManyToMany, ManyToOne, OneToMany, et OneToOne.

Une fois l’entité créée, la commande « make:migration » permet de créer les fichiers de migration, et la commande « doctrine:migrations:migrate » permet d’exécuter les migrations en base de données. Si tout se passe bien, les tables sont là.

# Créer une route

Pour les routes, cela se passe dans les « Controllers ». En se basant sur un controller existant, on remarque la définition du namespace, l’import des différentes resources (services, entités, …).

Via des annotations, on peut créer une « route ». Cette route redirigera la requête de l’URL sur la fonction suivante. La fonction qui suit se termine généralement par le mot-clé « Action ».



La route ci-dessus permet d’obtenir un objet JSON si on requête l’URL (<http://mon-symfony.fr/user>).

Grâce à Doctrine, je peux récupérer un user en particulier via son ID en base, et retourner un JSON complet sur le modèle de l’user grâce à JMS. Exemple :

