API de gérération aléatoire de ressources

Cahier des charges

Version 1.1 du lundi 16 avril 2018

Par Jordan JUVENTIN de 2dtension.fr

en coopération avec

Florian RAMBUR florianrambur.com

Pour ECA Digital

Sommaire

Définition des termes employés

Nous allons voir ici la définition des termes courramment employés dans ce document. Veuillez noter que les définitions ne s'appliquent qu'à ce document.

Ressource	Un ensemble de champs
Champ	Une clef, une valeur
Clef	Nom unique dans la ressource
Valeur	Un type et une donnée typée
Type	Un ensemble de règles
Règle	Permission ou restriction

Expression des besoins

Préambule

Le but premier de cette application est de fournir un service distant permettant de récupérer des ressources formatées selon une convention définie plus bas et générées aléatoirement. Elle doit simuler le fonctionnement conventionnel d'une API.

Cette API sera à la fois à buts utilitaire et éducatif. Elle servira à récupérer un grand jeu de données formatées rapidement, utile pour simuler une population de données. Elle servira également à tester les fonctionnalités d'une application dépendante d'une API dont ladite API n'existe pas encore. Elle servira enfin à apprendre à utiliser une API, car le but secondaire est la formation à l'usage des API.

Besoins fonctionnels

Cette application est une API (pour Application Programming Interface). Sa nature définit qu'elle doit tourner en autonomie, en permanence et à distance. Comme exprimé plus haut, son but premier est de fournir un service distant permettant de récupérer des ressources formatées.

A propos du format

Le format qui a été choisi pour servir de base est le JSON (pour JavaScript Object Notation). Il s'agit d'un format reposant sur des couples clef-valeur, permettant l'accès à l'information en connaissant le nom de ladite information. Qui plus est, ce format est nativement géré par nombre de technologies.

Cependant, il doit être possible d'ouvrir l'application à d'autres formats de données.

A propos des ressources

Les utilisateurs doivent pouvoir définir les ressources qu'ils souhaitent utiliser. Il doit être possible de créer, modifier et supprimer des ressources.

Ce qui doit être accessible à l'utilisateur dans les ressources inclut : le nom de celle-ci, ses clefs, ses valeurs et les types des valeurs.

A propos des types

Les utilisateurs doivent pouvoir définir les types avec lesquels ils souhaitent travailler. Il doit être possible de créer, modifier et supprimer des ressources.

Ce qui doit être accessible à l'utilisateur dans les types inclut : le nom de celui-ci, l'ensemble de ses règles.

A propos des règles

Les règles sont définies à l'intérieur même du code. Aussi, elles sont fixées au préalable et ne sont donc pas modifiables par l'utilisateur.

Contraintes

L'application devra fonctionner et répondre aux besoins exprimés plus haut. Elle devra en outre respecter les standards de programmation ainsi que les bonnes pratiques techniques, tout en permettant une maintenabilité et une évolutivité non contraignantes.

Il y a aussi une contrainte de développement, qui est l'open source. Cette application devra héberger son code source sur un gestionnaire de version accessible en ligne (par exemple : github) et permettre à des utilisateurs d'y contribuer s'ils le souhaitent.

Documentation

Il va sans dire que le produit final doit être fourni avec une documentation la plus explicite possible. Il va s'agir, pour cette documentation, d'un mélange :

- D'une documentation générée automatiquement par un outil tel que Swagger
- D'une documentation écrite par les développeurs du projet
- D'un code source auto-documenté

Liste des fonctionnalités

Une API

Cette application est une API. Elle doit donc fournir un ensemble de routes nécessaires à son fonctionnement. Elle doit, au maximum, s'approcher de la norme REST (pour REpresentationnal State Transfer) tout en respectant ses propres problématiques.

Elle doit être accessible 24h/24, 7j/7, y compris en ligne de commande.

Aléatoire

L'aléatoire de cette application respecte plusieurs principes :

- La structure de la ressource n'est jamais altérée par l'aléatoire
- Les clefs des champs ne sont jamais altérées par l'aléatoire
- Les types des champs ne sont jamais altérés par l'aléatoire
- Les valeurs sont, tout en respectant les principes précédants, aléatoires

Gestion par l'utilisateur

L'utilisateur doit pouvoir créer la ressource qu'il souhaite. Pour cela, il va avoir à sa disposition des routes pour :

- Gérer la ressource en elle-même en CRUD (pour Create, Read, Update, Delete)
- Ajouter des champs à sa ressource
- Gérer les types en CRUD
- Typer les valeurs de sa ressource

Evolutivité

Ce principe a d'ores-et-déjà été évoqué, mais le produit final devra être évolutif. Il est en effet amené à être partagé, et notamment sur un gestionnaire de versions. Il sera donc en constant développement.

Aussi, les fonctionnalités évoquées plus haut représentent les fonctionnalités minimales de l'application, à savoir une API fonctionnelle et accessible.

Vous pourrez retrouver ci-après une liste des évolutions possibles, voire recommandées.

L'implémentation et la gestion des règles rentre dans la livraison dite « optimale ». Dans cette livraison optimale viennent aussi les évolutions n°s 1, 2 et 3 cités juste en dessous.

Prévisions des évolutions

Voici une liste, non exhaustive et non ordonnée, d'évolutions envisageable.

1	CRUD sur les champs	Faisabilité : 8/10 Utilité : 8/10	Il s'agira de permettre de réutiliser des champs déà existants dans d'autres ressources. Il faut donc prévoir un CRUD.
2	Librairie JavaScript	Faisabilité : 7/10 Utilité : 10/10	Il s'agira de fournir une librairie JavaScript front-end pour permettre une utilisation plus aisée de l'API
3	Interface Web	Faisabilité : 7,5/10 Utilité : 10/10 Requiert : n°2	Il s'agira de fournir un site web pour interagir avec l'API de manière visuelle. Cette interface intégrera aussi les documentations et présente un intérêt éducatif évident.
4	Téléchargement	Faisabilité : 10/10 Utilité : 5/10 Requiert : n°3	Il s'agira de fournir la possibilité de télécharger les jeux de données, et pas simplement de les récupérer en affichage.
5	Plusieurs formats	Faisabilité : 7/10 Utilité : 4/10 Requiert : n°4	Il s'agira de permettre un téléchargement en différents formats (Ex : CSV, XML, ...)
6	Statistiques	Faisabilité : 6,5/10 Utilité : 1/10 Requiert : n°3	Il s'agira de fournir des statistiques sur l'utilisation des ressources au cours du temps.
7	Iframe	Faisabilité : 8/10 Utilité : 2/10 Requiert : n°3	Il s'agira de permettre de récupérer une iframe affichant les ressources. Ceci implique qu'il est possible de gérer du HTML et du CSS pour une ressource.
8	Faire le ménage	Faisabilité : 10/10 Utilité client : -5/10 Utilité serveur: 10/10	Il s'agira de supprimer les ressources non utilisées depuis trop longtemps.
9	Champ = fonction(champ)	Faisabilité : 3/10 Utilité : 8,5/10	Il s'agira d'utiliser la valeur d'un champ pour calculer la valeur d'un autre champ.

Dans ce chapitre qu'est l'évolutivité, il faut aussi aborder deux questions qui, même si elles ne le semblent pas, sont liées : le cache et la résilience des données générées.

La question du cache engendre la question de performances du système. Car, en effet, si plusieurs personnes cherchent à générer des jeux importants d'une même donnée, le serveur va fonctionner à plein régime pour leur fournir. Il faut donc trouver un moyen d'alléger la charge.

Pour cela, un système de cache peut-être intéressant. Cependant, à une même route pour une même ressource, un cache n'est pas forcément souhaitable : il s'agit de la question de résilience des données.

Admettons qu'une personne lambda souhaite accéder à 10 ressources X. Il va accéder à une route. 10 minutes plus tard, il en veut à nouveau 10. Il va accéder à la même route. Doit-il tomber sur la version en cache de sa précédente requête, ou bien faut-il générer à nouveau 10 ressources X ?

Si les données sont en cache, elles ne sont pas toujours totalement aléatoires. Cependant on peut accéder à une ressource déjà générée (faire une espèce de SELECT). Il y a une partie conceptuelle à prendre en compte.

Si les données ne sont pas en cache, les données sont totalement aléatoires, mais le SELECT est impossible. Le serveur prend aussi le risque de subir une attaque DDoS (pour Distributed Denial of Service).

C'est une question qui reste en suspens. Toutefois, dans le cadre de la première version de l'application, le choix a été fait de fournir un système sans cache. Ceci n'empêche en rien de le développer par la suite, et permet tout de même à l'application de tourner correctement. Qui plus est, l'application sera pour commencer en huis clos, le DDoS est donc peu probable.

Liste des technologies

Git	Il s'agit d'un logiciel de gestion de version. Il est très répandu dans beaucoup de domaines du développement. Il sera utilisé ici via la plate-forme GitHub.
Go (GoLang)	Il s'agit d'un langage serveur. Il est nativement multi-threads, parallélise les opérations et est donc adapté aux lourdes opérations algorithmiques.
PHP	Il s'agit d'un langage serveur. Il a fait ses preuves tout au long de son existence. Il profite d'une immense communauté, d'une grande documentation. La facilité d'implémentation justifie aussi son usage pour l'interface web.
Code Igniter	Il s'agit d'un framework PHP extrêmement léger, qui facilite l'implémentation de code très répandu dans le web. Il est MVC, et fournit des librairies de code très utiles sans ralentir l'exécution.
HTML / CSS / JavaScript	Il s'agit des deux langages incontournables du web front-end.
Bootstrap	Il s'agit d'un framework CSS, qui facilite l'implémentation du style. Il est basé sur un système de grille à 12 colonnes.
jQuery	Il s'agit d'un framework JavaScript, massivement répandu aujourd'hui. Il permet notamment
Vue.js	Il s'agit d'un framework JavaScript. Il sert à faire le lien entre vue et modèle, côté front-end.
MySQL	C'est un système de gestion de bases de données relationnelles. Il servira à stocker les données structurées.
MongoDB	C'est un système de gestion de bases de données non relationnelles. Il servira à stocker les données non structurées.

Liens utiles

PHP	HTML
GoLang	CSS
Swagger pour GoLang	JavaScript
Code Igniter	jQuery
MongoDB	MySQL
Bootstrap	Vue.js
Fake (équivalent de Faker pour GoLang)	GitHub
DataTables	EJS Charts
Font Awesome	RoboHash
LoremPixel	Go random data
Goji	