Rapport

Défi : API de Confiance Open Data

Equipe : **MasterOfDL**

Damien ARONDEL
Florent MOUYSSET
Axel ROBERT
Jérémy SIMAR
Vincent TERTRE
Laurent WARIN

Table des matières

Introduction

Dans le cadre de la nuit de l'info se déroulant les 5 et 6 décembre 2013, la société Docapost[1] propose un défi intitulé API de Confiance Open Data[2]. Le présent rapport propose la solution mise en place par l'équipe MasterOfDL de l'Université Toulouse III Paul Sabatier. Avant d'aller plus loin dans le document, il est important d'expliquer les différents termes évoqués :

- ***Qu'est-ce qu'une API ?***

Une API est un ensemble de services permettant d'interagir avec un système, comme une application ou un site web. Dans notre cas, une telle interface est requise pour accéder aux données publiques, utilisable par un tiers pour leur propre utilisation personnelle. On peut par exemple fournir des statistiques sur les produits les plus vendus à Toulouse durant l'année 2013. Cette information peut ainsi être utilisée par des journalistes pour expliquer les tendances actuelles.

- ***Qu'est-ce que l'Open Data ?***

L'Open Data est un mouvement visant à rendre disponible des données d'origine publique ou privée. Pour avoir l'étiquette Open Data, ces données doivent être libres d'accès et d'utilisation sans aucune restriction juridique, technique ou commerciale. Grâce à ce mouvement, des entreprises ou des particuliers ont pu initier des projets d'utilisation de ces données tels que des applications d'informations sur les réseaux de transport ou de localisation.

- ***Comment se décompose le rapport ?***

Il est nécessaire d'expliquer les différentes sections disponibles dans le rapport. En voici la liste :

- ✓ Définition du contexte : dans quoi s'inscrit notre solution ?
- ✓ Présentation de l'API mise en œuvre : quelle utilisation ?
- ✓ Choix techniques : comment l'API a-t-elle été implémentée concrètement ?
- ✓ Corrélation entre le défi et notre solution : en quoi répond-elle aux contraintes ?
- ✓ Exemple d'utilisation de l'API : comment s'en servir ?

[1] http://www.docapost.com/
[2] http://www.nuitdelinfo.com/nuitinfo/defis:api_de_confiance_open_data:start

Définition du contexte

Le défi principal de la nuit de l'info 2013, proposé par CDiscount, consiste à développer un site Y-commerce. Le sigle *Y*, pour *Your* introduit le paradigme selon lequel ce n'est plus le produit mais le consommateur qui est au cœur des préoccupations. L'époque des sites catalogues référençant des milliers d'articles non conformes avec les attentes des consommateurs est révolue. Place désormais à des contenus dynamiques capables de s'adapter aux préférences et aux habitudes des utilisateurs.

Pour répondre à cette problématique, l'équipe MasterOfDL a mis en place un système innovant d'échanges entre utilisateurs. Le jeu est simple : on attribue un objet virtuel à chacun d'entre eux. S'ils le souhaitent, ils peuvent se les échanger pour arriver à obtenir le produit de leur choix. Ce système a un objectif double :

- Donner du contenu interactif à l'utilisateur du Y-commerce, couplé à un aspect social (échanges).
- Etablir des statistiques utiles au commerçant, comme le pourcentage d'utilisateurs désirant une certaine catégorie de produits.

EXTRAIT DU RAPPORT SUR LA SOLUTION Y-COMMERCE (MasterOfDL)

- L'utilisateur à séduire : Cet utilisateur est sur l'application car il veut trouver un produit sans cependant avoir une idée précise. Nous devons donc trouver des mécanismes permettant à au client de trouver un produit qui lui plaît parmi notre catalogue, sans passer par les étapes habituelles et fastidieuses de filtres / champs de recherche.

Nous avons conçu une application qui met en place un jeu d'échange. Il s'agit d'une fonctionnalité complémentaire d'un site marchand normal : de CDiscount à Amazon. L'idée est d'avoir en plus des menus "rubriques" horizontaux, un menu latéral où le Trade-Game pourra prendre place.

Le trade-show est un jeu d'échange de produit entre les utilisateurs connectés et authentifiés à l'application.

Le trade-show démarre à un instant T et se termine à la fin d'un timer qui se veut long, une semaine par exemple. A l'instant T un Produit est attribué à l'utilisateur, provenant du catalogue de nos produits. Nous verrons par la suite que ce produit pourra être échangé plusieurs fois.

A la fin du temps imparti, une récompense s'applique sur le TypeDeProduit que l'utilisateur possède à ce moment précis.

Exemple : L'utilisateur Vincent, démarre une session de trade Game, le timer indique une semaine. On lui attribue le produit suivant : Clé-USB de 4Go de marque A. Si le timer se termine et qu'il possède ce produit à ce moment-là, une récompense liée au produit de Type : Clé-USB lui sera délivré, nous n'avons en l'occurrence pas défini de récompense précise (réduction en %, bon d'achat etc…).

Présentation de l'API

Méthode : Get Score List

Description

Retourne la liste des produits et leur évaluation.

Syntaxe URL

/api/{version}/score

Paramètres

Aucun paramètre.

Exemple

En tant que comparateur de produits,
Je veux connaître l'évaluation de tous les produits
*Afin d'*établir un classement des sites e-commerce.


```
[
	{
		"productScoringList": [{
		"productName": "Asus e-300",
		"score": 60
	},
	{
		"productName": "Emachines b500",
		"score": -3
	},
	{
	"productName": "HP-BC9856 + 22 inch screen",
		"score": 0
	}
]
```

Méthode : Get Product Score

Description

Retourne l'évaluation pour un produit donné.

Syntaxe URL

/api/{version}/score?product_name={product_name}

Paramètres

Nom du paramètre	Type	Mandatory
product_name	Integer	yes

Exemple

En tant que service marketing de la marque X,
Je veux connaître l'évaluation d'un de mes produits
Afin de connaître sa notoriété.

```
{
        "score": 60
}
```

Méthode : Get Product List

Description

Retourne la liste des produits en vente.

Syntaxe URL

/api/{version}/product

Paramètres

Nom du paramètre	Type	Mandatory
scoreMoreThanEquals	Integer	no
scoreLessThan	Integer	no

Exemple

*En tant qu'*étudiant participant à la nuit de l'informatique,
Je veux récupérer une liste de produits
*Afin d'*alimenter ma base de données de site Y-commerce.

En tant que comparateur de produits,
Je veux connaître la liste des produits correspondant à une évaluation prédéfinie
Afin de conseiller mes utilisateurs en fonction de critères préétablis.


```
{
  "productRepresentationList": [{
      "productName": "Asus e-300",
      "description": null,
      "price": 0,
      "type": "Laptop"
  }, {
      "productName": "Emachines b500",
      "description": null,
      "price": 0,
      "type": "Laptop"
  }]
}
```

Méthode : Get Multi Account User

Description

Retourne le nombre d'utilisateurs ayant plusieurs comptes (même code postal, sexe, date de naissance).

Syntaxe URL

/api/{version}/user

Paramètres

Aucun paramètre.

Exemple

En tant que Docapost,
Je veux connaître le nombre d'utilisateurs susceptibles de frauder en utilisant le multi-compte
Afin de mener des investigations approfondies envers ces utilisateurs.


```
{
  "productRepresentationList": [{
      "productName": "Asus e-300",
      "description": null,
      "price": 0,
      "type": "Laptop"
  }]
}
```

Méthode : Get Users By Action

Description

Retourne les caractéristiques moyennes d'un utilisateur.

Syntaxe URL

/api/{version}/user?operation={action_name}

Paramètres

Nom du paramètre	Values	Mandatory
action_name	{avg-age, sex}	yes

Exemple

En tant que faculté de sociologie,
Je veux connaître l'âge moyen des utilisateurs d'un site Y-commerce
Afin de dresser le profil d'utilisateurs appartenant à une catégorie d'âge.

Méthode : Get Command Number (non implémentée)

Description

Retourne le nombre d'utilisateurs ayant effectué plus de X commande(s) lors des Y derniers mois.

Syntaxe URL

/api/{version}/command?action={action}&nb={x}&time={y}

Paramètres

Nom du paramètre	Type	Mandatory
action	Integer	yes
x	Integer	yes
y	Integer	yes

Exemple

En tant que Docapost,
Je veux connaître le nombre d'utilisateurs ayant effectué des commandes multiples
*Afin d'*être en mesure de détecter d'éventuelles fraudes.

Méthode : Get Command Statistics (non implémentée)

Description

Retourne des informations relatives aux commandes des X derniers mois.

Syntaxe URL

/api/{version}/command?action={action}&time={x}

Paramètres

Nom du paramètre	Type	Values	Mandatory
action		{express,same,litigation}	yes
x	Integer		yes

Exemple

En tant que Docapost,
Je veux connaître le pourcentage de livraisons express demandées sur les x derniers mois
Afin de déterminer si une fraude a été commise.

En tant que Docapost,
Je veux savoir si plusieurs articles identiques ont été commandés lors d'un même achat
Afin de déterminer si une fraude potentielle a été commise.

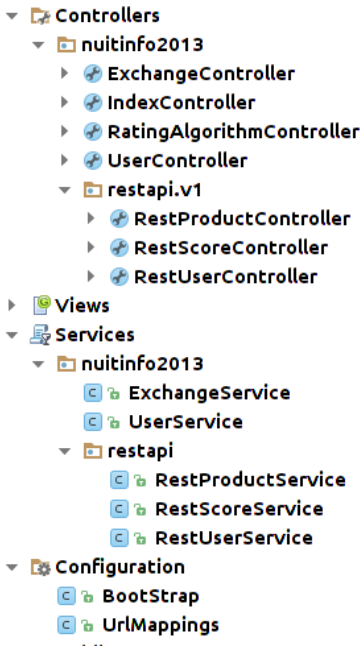
En tant que Docapost,
Je veux connaître le nombre de litiges sur les commandes des X derniers mois
Afin de déterminer si une fraude a été commise.

Aspects techniques

Technologies utilisées

Au niveau du défi principal, le choix technique s'est porté sur le framework Grails[3] couplé au langage Groovy. Etant donné que nous avons décidé d'intégrer le développement du défi API de confiance Open Data à notre projet principal, nous avons gardé les mêmes technologies.

Nous avons donc exposé notre API dans un fichier de mapping URL, mis à disposition par le framework Grails. Ce mapping permet au framework de rediriger toutes les requêtes REST qui sont envoyées vers un contrôleur associé. De cette manière, nous avons pu mettre en place un système de paramètres nommés permettant de définir des requêtes. Chacun de ces paramètres peut être ensuite récupéré au niveau des contrôleurs puis être soumis à un traitement interne afin d'aller récupérer les résultats dans notre base de données. Le résultat de la requête est enfin retourné dans un format dépendant du paramètre de même nom spécifié dans l'URL. JSON est le format par défaut.



L'image ci-dessus montre l'architecture type de Grails adapté à notre projet. Le contrôleur s'attribue la mise en forme des données. Les services s'occupent de la logique métier : construction des requêtes, récupération des données en base, ….

[3] http://grails.org/

Voici un exemple de contrôleur :

```
class RestProductController {

    static allowedMethods = [retrieve: 'GET']

    RestProductService restProductService

    def retrieve(String scoreMoreThan, String scoreLessThan, String format) {
        Map representation = restProductService.buildRequestRepresentation(scoreMoreThan, scoreLessThan)

        render(representation as JSON)
    }
}
```

Voici un exemple de service construisant la réponse associée à une requête :

```
            def buildRequestRepresentation(String productName) {
                Map representation

                if (!productName) {
                    def productList = Product.findAll()
                    representation = [
                            productScoringList:
                            productList.collect { def product ->
                                def score = 0
                                def ratingList = Rating.findAllByProduct(product)
                                ratingList.each { score += it.elo }
                                return [
                                        productName: product.name,
                                        score: score
                                ]
                            }
                    ]
                } else {
                    def score = 0
                    Product product = Product.findByName(productName)

                    def ratingList = Rating.findAllByProduct(product)
                    ratingList.each { score += it.elo }
                    representation = [
                            score: score
                    ]

                }

                return representation
            }
        }
```

Comme on peut le constater, le code est minimal et optimisé.

Démarche méthodologique

Notre équipe a pris le parti d'adopter une philosophie agile pour cette nuit de l'info 2013. Nous avons donc mis en place la méthode Scrum[4] afin de cadrer notre processus de développement.

Après une phase de brainstorming qui a amené à un découpage en features et user stories de notre projet, nous avons mené successivement des sprints de 2h environ. Ainsi, nous nous sommes répartis le travail en tâches atomiques tout en procédant à des revues de sprint afin de mettre en commun notre travail et de définir ce qu'il reste à faire pour le prochain sprint.

[4] https://www.scrum.org/

L'API et le défi

Docapost a fixé différents critères d'évaluation. Le but est de justifier que notre solution respecte les contraintes initiales :

Critères d'évaluation :

Seront également valorisés pour départager les meilleurs projets :

- La pertinence du choix des données ouvertes mise à disposition par rapport aux cas d'usages envisagés L'ergonomie de l'API (principe KISS)
- Le respect de la philosophie Open Data
- Le respect des concepts REST et ceux de création d'une API web
- Et le bonus ultime : la mise à disposition d'une API fonctionnelle

- ***Pourquoi l'API suit le principe KISS ?***

Le principe KISS, pour « Keep It Simple, Stupid », est « *une ligne directrice de conception qui préconise la recherche de la simplicité* » (citation provenant de Wikipedia[5]). D'un point de vue utilisateur, la simplicité est induite par l'architecture REST. D'un point de vue développeur, l'utilisation du framework Grails, couplé au langage Groovy, permet de produire un code minimal et maintenable.

- ***Pourquoi l'API repose sur l'architecture REST ?***

L'architecture REST repose sur 5 contraintes :

- ✓ L'interface est uniforme.
- ✓ Les données sont sans état.
- ✓ Le client peut mettre en cache la réponse.
- ✓ L'architecture logicielle est client-serveur.
- ✓ L'API se décompose en couches.

- ***Pourquoi les valeurs retournées par l'API sont Open Data ?***

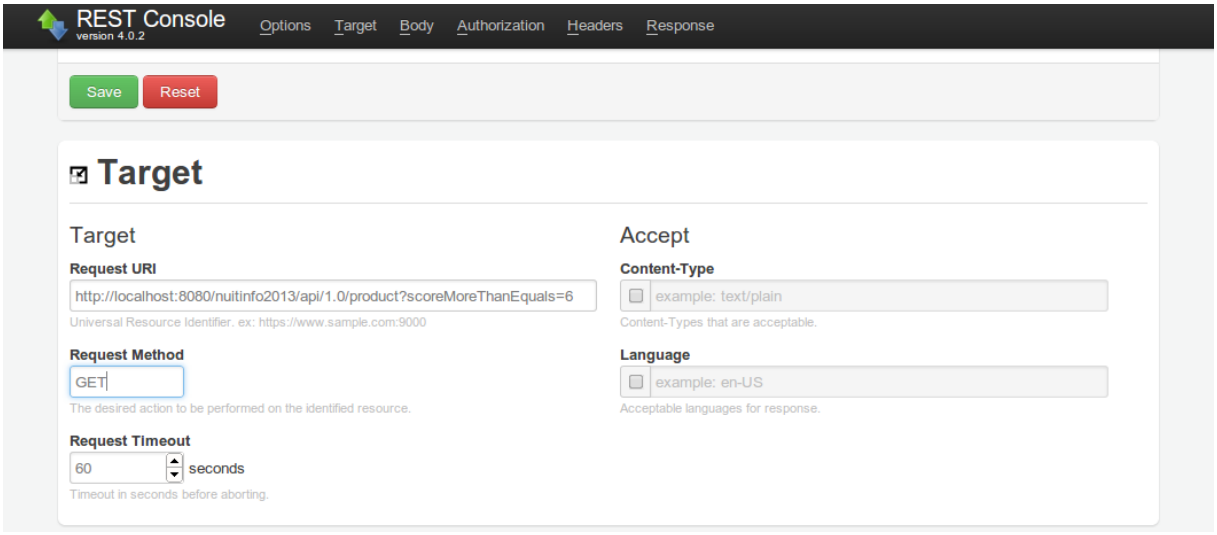
★ Available on the web (whatever format) but with an open license, to be Open Data
★★ Available as machine-readable structured data (e.g. excel vs. image scan of a table)
★★★ as (2) plus non-proprietary format (e.g. CSV instead of excel)
★★★★ as (3), plus using open standards from W3C (RDF and SPARQL) to identify things through de-referenceable HTTP URIs, to ensure effective access
★★★★★ as all the above plus establishing links between data of different sources.

D'après l'image[6] ci-dessus, on constate que non seulement les informations doivent être libres d'accès et de réutilisation, mais qu'elles doivent également être disponibles dans un format de données interopérables (XML ou JSON notamment). C'est le cas de l'API.

[5] http://fr.wikipedia.org/wiki/Principe_KISS
[6] http://www-etis.ensea.fr/WOD2013/?page_id=28

Suggestion d'utilisation

Pour pouvoir utiliser notre API, on peut par exemple se servir d'une console REST :



Ça permet d'envoyer une requête REST en ciblant une URI. Ensuite, le serveur retourne une réponse JSON ou XML, construite par les contrôleurs Grails :