APIs REST

Introduction et Utilisation

Jean-René Rouet <rouet@in2p3.fr>

Introduction

- Building restful web services, like other programming skills is part art, part science. As the Internet industry progresses, creating a REST API becomes more concrete, with emerging best practices. As RESTful Web services don't follow a prescribed standard except for HTTP, it's important to build your RESTful API in accordance with industry best practices to ease development and simplify client adoption.
- Presently, there aren't a lot of REST API guides to help the lonely developer. RestApiTutorial.com is dedicated to tracking REST API best practices and making resources available to enable quick reference and self education for the development crafts-person. We'll discuss both the art and science of creating REST Web services.
- —Todd Fredrich, REST API Expert

Thèse de Roy T. Fielding

- Uniform Interface
- Stateless
- Cacheable
- Client Server
- Layered System
- Code on Demand (optional)

Basé sur les ressources

- Les ressources individuelles sont identifiées dans les requêtes par l'URI.
  - http://host.com/users/1
- Les ressources sont séparées conceptuellement de la représentation qui est retournée au client.
- En général, on retourne du XML ou du JSON en UTF8.

```
<person>
   <name>Förster</name>
   <surname> James</surname>
   <mail>james.forster@mycompany.com</mail>
</person>
```

La manipulation des ressources est faite au travers des représentations

- Le client possède toute l'information nécessaire grâce à la représentation et les méta-données pour modifier ou supprimer une ressource.
- si il possède les privilèges.

Les messages sont auto descriptifs

- Le messages inclut toute l'information nécessaire au client pour traiter le messages.
  - par exemple le mime-type pour une format vidéo
- Les réponses peuvent indiquer la politique de cache

Hypermedia est le moteur de l'état de l'application

- Le client délivre l'état des données en utilisant le "content body", la "query string", les "headers" et l'URI (l'identifiant de la ressource)
- Le serveur délivre l'état des données en utilisant le "content body", le "response code" et les "response headers".
- Les accès aux ressources ou objets relatifs est sous forme d'URI

L'interface uniforme est essentielle dans l'implémentation d'un service REST

ne pas confondre les APIs basés sur HTTP et un service REST

REST est REpresentational State Transfer

pas de conservation d’état.

cela signifie que les données nécessaires au traitement du statut sont toutes présentes dans la requête elle-même.

l'URI identifie la ressource et le corps du message contient le statut (ou le changement de statut) de la ressource.

Nous sommes habitués aux applications web avec gestion de session.

- Ce n'est pas le cas avec les APIS rest.
- Meilleure scalabilité.
- Les frontaux d'équilibrage de charge n'ont pas à gérer l'ID de session.

Quelle est la différence entre la représentation et la ressource ?

- La représentation est l'ensemble d'informations nécessaire pour répondre à une requête
- La ressource est l'ensemble des données représentant l'information (par exemple le contenu d'une ligne dans une table dans une base de données

Si dans une application web, le fait d'utiliser le bouton back du navigateur vous perd, c'est que l'application ne respecte pas le principe de "statenessless".

Les limites d'appel d'API aussi.

Les réponses d'un service REST doivent inclure la politique de cache et sont donc potentiellement mises en cache par le client

L'architecture Client REST et Serveur REST est une architecture client-serveur.

- Pas de dépendance dans le développement du code du client vis à vis du code du serveur.
- Et vice et versa.
- À condition que l'interface ne change pas évidemment.

Le client ne sait pas si il est connecté au serveur final ou à un intermédiaire

- Cela permet d'étendre les performances et la scalabilité du serveur sans impacter le code du client

Code on demand (optionnel)

- Le serveur peut envoyer du code au client à exécuter.

Et?

- Applications
- Développeurs
- Pragmatisme
- Adoption

faire une api REST pour son application en respectant tous les principes est très compliqué.

Quelques conseils

- schéma de l'url : http://monservice.com/api/v2/ ou http://api.monservice.com/v2/
- les verbes HTTP sont signifiants
- le nommages des ressources est important et ne doit pas être obscur
  - /users/rouet et non /?type=user&id=rouet
  - pas de verbe, utiliser le pluriel et pas /user_list
  - si pas de pluriel, c'est un singleton
  - url le plus court possible

Quelques conseils

- les codes réponses HTTP sont utiles et doivent être utilisés
- offrir à minima du json et du xml en sortie
- les ressources doivent de fine granularité
  - éviter "/users/rouet" qui renvoie aussi la liste de ses commandes puis les lignes de commandes puis la description des produits
- pensez déconnecté (utilisez les liens href pour lier les données entre elles)

Implémentation

CRUD repose sur les verbes HTTP : GET, POST, PUT, DELETE

GET

GET /users


```
[
  {
    "id": 1,
    "username": "john_smith",
    "name": "John Smith",
    "state": "active",
    "avatar_url": "http://localhost:3000/uploads/user/avatar/1/cd8.jpeg",
  },
  {
    "id": 2,
    "username": "jack_smith",
    "name": "Jack Smith",
    "state": "blocked",
    "avatar_url": "http://gravatar.com/../e32131cd8.jpeg",
  }
]
```


retourne 200

GET

GET /users/john_smith

```
{
  "id": 1,
  "username": "john_smith",
  "name": "John Smith",
  "state": "active",
  "avatar_url": "http://localhost:3000/uploads/user/avatar/1/cd8.jpeg",
}
```

retourne 200

POST

POST /users

```
{
  "id": 1,
  "username": "john_smith",
  "name": "John Smith",
  "state": "active",
  "avatar_url": "http://localhost:3000/uploads/user/avatar/1/cd8.jpeg",
}
```

retourne 201 avec un lien vers l'URI

PUT

PUT /users/1

```
{
  "username": "john_smith",
  "name": "John Smith",
  "state": "active",
  "avatar_url": "http://localhost:3000/uploads/user/avatar/1/cd8.jpeg",
}
```

retourne 200 (OK) ou 204 (No Content), 404 (Not Found)
idempotent

DELETE

DELETE /users/john_smith

retourne 200 (OK), 404 (Not Found)
idempotent

PATCH

PUT est sensé mettre à jour la ressource complète (tous les attributs)

PATCH permet de mettre à jour partiellement une ressource

```
[
  {
    "op": "replace",
    "path": "/email",
    "value": "new.email@example.org"
  }
]
```

Pourquoi faire une API REST à son application/service

Pour un service, cela parait naturel car il doit être utilisé par d'autres services.

Pour une application, vous voulez fournir un moyen d'intéragir avec votre application par programme.

Pourquoi utiliser une API REST dans son application/service

Etendre les fonctionnalités sans réimplémenter.

Un exemple

- Traitement d'une erreur
- L'utilisateur rencontre une erreur sur votre application, vous voulez automatiquement insérer un rapport de bug dans votre gestion de projet.
- L'application de gestion de projet (redmine) ou de code (gitlab) posséde une API REST.
- Il est donc très facile de le faire

Client

curl (libcurl) avec un parseur JSON suffit

Javascript : $.ajax

exemple : mozaik (http://mozaik.herokuapp.com)

Serveur

tous les frameworks ont ou peuvent intégrer un plugin-bundle REST

- Symfony : https://github.com/FriendsOfSymfony/FOSRestBundle
- Django : http://www.django-rest-framework.org
- Java : https://jersey.java.net

Références

- Roy T. Fielding http://roy.gbiv.com/untangled/
- Learn REST: A RESTful Tutorial (Todd Fredrich). http://www.restapitutorial.com/
- Beautiful REST & JSON APIs (Les Hazlewood). https://youtu.be/mZ8_QgJ5mbs
- Technologies pour Web Services faciles : REST, JSON (Pierre Gambarotto). https://2009.jres.org/planning_files/article/pdf/92.pdf
- GitLab documentation (GitLab API). http://doc.gitlab.com/ce/api/