SOFIP
SE FORMER POUR L'AVENIR

Qualiopi
processus certifié
RÉPUBLIQUE FRANÇAISE

Api Rest

Introduction

REST n'est pas un protocole, mais une architecture. Créé par Roy Fielding en 2000, REST est un acronyme pour **Re**presentational **S**tate **T**ransfer. Souvent associée à l'architecture orientée service (Service Oriented Architecture), cette architecture est un moyen de présenter et manipuler des ressources.

I L'API

Une API (**A**pplication **P**rogramming **I**nterface) est une application à laquelle on peut faire effectuer des actions via le protocole HTTP :

- récupérer des données concernant des utilisateurs ;
- ajouter des produits ;
- supprimer l'auteur d'un article ;

Cela ressemble franchement à ce que nous pourrions retrouver dans un site web habituel… hormis le fait qu'il ne s'agit pas d'afficher des pages web HTML.

Les utilisateurs d'API sont d'autres développeurs (ou vous-même), et ils s'attendent à ce que votre API REST soit architecturée d'une certaine manière.

C'est une bonne nouvelle ! En effet, savoir à qui vous vous adressez et comment vous devez le faire facilite grandement le travail ! Plus besoin de réfléchir à "la meilleure façon de faire".

Nous ne parlons pas de code ici, mais d'une structure d'application.

Concrètement, vous aurez à créer une liste d'actions possibles avec votre application (récupérer une liste d'utilisateurs, en ajouter, en supprimer…), et une manière d'effectuer ces actions grâce à HTTP. Cette manière de faire doit respecter les nombreuses contraintes de REST.

Commençons par passer en revue ces contraintes en rapport avec ce que votre API devra refléter – les six contraintes de REST :

- Contrainte n° 1 : Client/Server (Client-Serveur).
- Contrainte n° 2 : Stateless (sans état).
- Contrainte n° 3 : Cacheable (cachable).
- Contrainte n° 4 : Layered system (système à plusieurs couches).
- Contrainte n° 5 : Uniform interface (Interface uniforme) :
    - une ressource doit posséder un identifiant ;
    - une ressource doit avoir une représentation ;
    - une ressource doit être autodécrite.
- Contrainte n° 6 : Code on demand (du code sur demande).

II Le protocole HTTP

Nous allons faire quelques rappels concernant HTTP : il s'agit d'un protocole d'échange entre deux machines. Une API n'est rien d'autre qu'une application capable de recevoir une requête HTTP et de rendre une réponse HTTP. Ce qui change d'un site web "classique" est le fait qu'il s'agit de faire des actions plus atomiques, contrairement à une page web HTML. Une API va se charger de la gestion des utilisateurs (ajout, suppression…) ou d'une gestion de produits, ou de toute autre ressource.

Requête HTTP

Une requête HTTP émane d'un client (tout logiciel dans la capacité de forger une requête). Une requête est constituée des éléments suivants :

1. La première ligne (*request line*) doit contenir :
   - la **méthode HTTP** (`GET` , `POST` , `PUT` , `PATCH` , `DELETE` , `OPTIONS` , `CONNECT` , `HEAD` ou `TRACE`) ;
   - l'**URI**, c'est-à -dire ce qu'il y a après le nom de domaine (exemple : `/users/1`) ;
   - la **version du protocole** (exemple : `HTTP/1.1`).
2. Les **entêtes** (*headers*), une entête par ligne, chaque ligne étant finie par le caractère spécial "retour à la ligne" (`CRLF`).
3. Le **contenu de la requête** (*body*), qui doit être séparé de deux caractères spéciaux "retour à la ligne" (`CRLF CRLF`) – optionnel.

SOURCE / OpenClassroom

Méthodes HTTP

- `GET` : utilisée pour récupérer des informations en rapport avec l'URI ; il ne faut en aucun cas modifier ces données au cours de cette requête. Cette méthode est dite *safe* (sécuritaire), puisqu'elle n'affecte pas les données du serveur. Elle est aussi dite *idempotent*, c'est-à-dire qu'une requête faite en GET doit toujours faire la même chose (comme renvoyer une liste d'utilisateurs à chaque fois que la requête est faite – d'une requête à l'autre, on ne renverra pas des produits si le client s'attend à une liste d'utilisateurs !)

- `POST` : utilisée pour créer une ressource. Les informations (texte, fichier…) pour créer la ressource sont envoyées dans le contenu de la requête.

- `PUT` : utilisée pour remplacer les informations d'une ressource avec ce qui est envoyé dans le contenu de la requête

- `PATCH` : utilisée pour modifier une ressource. La différence avec une requête avec la méthode PUT est que l'action à effectuer sur la ressource est indiquée dans le contenu de la requête. Prenons un exemple : nous souhaitons rattacher un utilisateur à une organisation ; dans le contenu de la requête, il sera indiqué qu'il s'agit d'un rattachement à une organisation, en plus des informations à mettre à jour.

- `DELETE` : utilisée pour supprimer une ou plusieurs ressources. Les ressources à supprimer sont indiquées dans l'URI.

- `OPTIONS` : utilisée pour obtenir la liste des actions possibles pour une ressource donnée (suppression, ajout…).

- `CONNECT` : utilisée pour établir une première connexion avec le serveur pour une URI donnée.

- `Head` : même principe que pour la méthode GET, mais seuls les entêtes devront être renvoyés en réponse.

- `TRACE` : utilisée pour connaître le chemin parcouru par la requête à travers plusieurs serveurs. En réponse, un entête `via` sera présent pour décrire tous les serveurs par lesquels la requête est passée.

RÉPONSE HTTP

Une réponse HTTP émane d'un serveur (tout logiciel dans la capacité de forger une réponse HTTP). Une réponse est constituée des éléments suivants :

- La première ligne (status line) doit contenir :
  - La version du protocole utilisée;
  - le code status
  - l'équivalent textuel du code status

- Les entêtes (headers), une entête par ligne, chaque ligne étant finie par le caractère spécial "retour à la ligne" (CRLF)

- Le contenu de la réponse (body) doit être séparé de deux caractères spéciaux "retour à la ligne"(CRLFCRLF) - optionnel

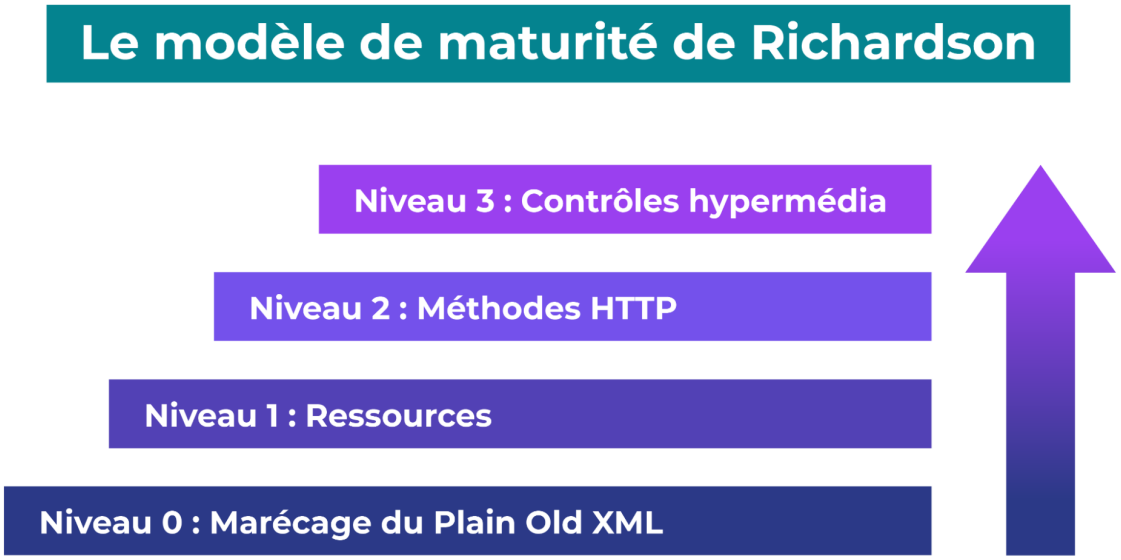
Exemple :

```
HTTP/1.1 200 OK Date:Tue, 31 Jan 2017 13:18:38 GMT Content-Type:
application/json {
 "current status" : "Everything is ok!" }
```

Le Modèle de maturité de Richardson

Le modèle de Richardson est aussi connu que la reine d'Angleterre, toutes proportions gardées, dans le monde des API REST.

Ce modèle donne un moyen d'évaluer son API. Plus on monte dans les niveaux, plus notre API est considérée RESTful (pleinement REST, autrement dit). Plus votre API adhère aux contraintes REST, plus elle est RESTful, ce qui est plutôt bien du point de vue des bonnes pratiques : en effet, étant donné que rien n'est imposé, le fait de respecter les règles permet de faire en sorte que ceux qui connaissent et aiment utiliser les API REST, et/ou développer des API REST, adopteront plus facilement votre travail.



source / OpenClassroom

Le Marécage du Plain Old XML

Une API qui ne fait que respecter le niveau 0 n'est pas une API REST. Ce niveau ressemble plus à ce que l'on peut retrouver dans une API SOAP (type d'API plutôt *old school*).

Il s'agit de :

- n'utiliser qu'un seul point d'entrée pour communiquer avec l'API, c'est à dire une seule URI, comme par exemple /api
- n'utiliser qu'une seule méthode HTTP pour effectuer ses demandes à l'API, avec POST

Ces deux règles ne sont pas à suivre selon les contraintes de REST énoncées : en effet, chaque ressource devrait avoir son point d'entrée. Si par exemple notre API gère des utilisateurs et des produits, il devrait y avoir au moins deux URI différentes pour récupérer ces listes, `/users` et `/products` . Par ailleurs, l'utilisation de la méthode HTTP `POST` pour toutes les actions que nous souhaitons effectuer sur l'API n'est vraiment pas une bonne idée : comme expliqué plus haut, la méthode POST n'est réservée qu'à la création de ressources. Si nous souhaitons récupérer une liste d'utilisateurs, il faudra utiliser la méthode `GET` pour la requête.

Les ressources

Le niveau 1 concerne les ressources et demande dans un premier temps que chaque ressource puisse être distinguée séparément. Cela ne vous rappelle rien ? Mais si, la contrainte n° 5 ! Je ne vais pas me répéter, du coup. Simplement, rappelez-vous qu'il faut que vos URI correspondent à la ressource que le client de votre API souhaite manipuler.

J'ajouterai encore un petit quelque chose, en prenant un exemple concret : une API permettant de manipuler des articles. Disons que nous souhaitons développer un CRUD (CRUD est un acronyme anglais pour **C**reate, **R**ead, **U**pdate et **D**elete, soit Créer, Lire, Mettre à jour et Supprimer). La première question qu'il faut se poser est la suivante :

Quelles vont être les URI par lesquelles les utilisateurs de mon API vont pouvoir effectuer ces actions ?

Nous devons réfléchir aux **points d'entrée** dès le début. D'après ce que dit le niveau 1 du modèle de Richardson, les solutions possibles devraient être :

- Création : POST / articles
- Lecture : GET/ articles ou GET / articles/{identifiant-unique}
- Mise à jour : PUT /articles/{identifiant-unique}
- Suppression : DELETE / articles/{identifiant-unique}

Comme vous pouvez le voir, les actions ne font plus partie de l'URI. C'est primordial de les retirer. L'information concernant l'action est désormais contenue dans la méthode HTTP. Nous utilisons la pleine capacité du protocole HTTP !

Autre point abordé par le niveau 2 : le ***code status***. Si vous décidez de respecter le niveau 2 du modèle de Richardson, il faut choisir avec sagesse le code retour HTTP dans vos réponses. Voici un petit tour des *codes status* les plus courants que vous serez amené à retourner en fonction des situations.
cf liste des status : https://httpstatusdogs.com/

Retenez simplement que plus vous serez explicite dans la réponse générée, meilleure sera la qualité de votre API.

Les Contrôles hypermédia

Et enfin le niveau 3, *The glory of REST* ! Le niveau 3 est simplement l'idée de rendre votre API auto découvrable, en imitant les liens hypertextes d'une page web classique. Concrètement, nous devons indiquer au client de votre API ce qu'il est possible de faire à partir d'une ressource.

Reprenons l'exemple de l'API de gestion d'articles : si le client demande un article, non seulement il obtiendra les informations de l'article en question (titre, contenu…), mais aussi la liste des liens (URL) pour effectuer d'autres actions sur cette ressource, comme la mettre à jour, ou voir un article associé, par exemple.

Voici un exemple de contenu de réponse offrant une auto découverte de l'API :

```
{
   "id" : 1,
   "title" : "Le titre de l'article",
   "content" : "<p> Le contenu de l'article.</p>",
   "links" : {
      "update" : "http://domain.name/article/1",
      "associated" : "http://domain.name/article/16"
   }
}
```