L'architecture REST

[Mise à jour le 11/11/2021]

- **Source** : Mooc Fun “Programmer l'internet des objets”
- **Vidéo** sur YouTube: Web et REST

1. Le modèle client-serveur du Web

Le Web et ses extensions sont basés sur un **modèle client-serveur**. Les serveurs possèdent des ressources et les clients peuvent y accéder ou les modifier grâce à un protocole tel que HTTP. Le **modèle client-serveur** est quelque chose de courant dans les réseaux informatiques, mais le Web suit certaines directives de conception connues sous le nom de **REST** (**RE**presentational **S**tate **T**ransfer).

Selon Roy Fielding, qui a défini ce modèle, REST est un ensemble de principes, de propriétés et de contraintes. REST utilise le modèle de communication client-serveur et utilise généralement le protocole HTTP (Hypertext Transfer Protocol).

Les **6 propriétés** qui définissent REST :

- **architecture client-serveur** : **les serveurs** sont les entités qui **fournissent des services** tels que l’heure, le partage de données, etc. ; les clients sont les entités qui interrogent ces services.
- **sans état** (ou stateless) : cela signifie que l’état de l’interaction entre le client et le serveur n’est pas stocké. Chaque requête client est traitée comme une nouvelle demande indépendante. De cette façon, **le serveur n’a pas besoin de stocker, suivre ou analyser les requêtes précédentes du même client**.
- **cacheability** : les périphériques intermédiaires ainsi que les clients peuvent mettre en cache les réponses pour une utilisation ultérieure. Cela conduit à améliorer les performances.
- **système en couches** : cela signifie que les services peuvent être réalisés en utilisant des couches telles que différents serveurs responsables de différentes parties de services (stockage, recherche, etc.), mais le client ne sait pas s’il est connecté au serveur final ou intermédiaire.
- **code à la demande** (facultatif) : les serveurs peuvent éventuellement envoyer du code exécutable aux clients.
- **interface uniforme** : elle fait référence aux principes selon lesquelles une ressource doit avoir une seule représentation telle que les **URI** (**U**niform **R**esource **I**dentifier), et suivre certaines directives pour la dénomination, le format de lien et de données, etc. Il doit avoir suffisamment d’informations sur le traitement. Les ressources initiales telles que la page d’accueil d’un site Web, devraient avoir d’autres liens leur permettant de découvrir d’autres ressources.

Le respect de ces règles permet l'évolutivité du système en ajoutant continuellement des acteurs et de nouvelles données. L'objectif principal est de simplifier le comportement du serveur afin de servir le plus grand nombre de requêtes possible.

2. Le nommage

Chaque ressource du Web est identifiée par une **valeur unique appelée URI** (**U**niform **R**esource **I**dentifier). Si l'URI contient des caractères internationaux, (comme les lettres accentuées, ...) il est appelé IRI (International Resource Identifier).

Les URI permettent de désigner une ressource de manière non ambigüe, c'est-à-dire que l'on ne retrouvera pas le même URI pour désigner deux ressources différentes. Par construction, la structure de l'URI est hiérarchique, ce qui permet de créer des identificateurs uniques de manière distribuée.

Si vous voulez identifier une ressource, vous devez posséder une séquence unique : un numéro de téléphone, un numéro de sécurité sociale, un nom de domaine. En y ajoutant quelque chose d'unique pour nous, cela crée un identifiant globalement unique.

Par **exemple**, on a une image que l'on veut identifier, on peut l'appeler

- "image"

mais il y a peu de chance que ce nom soit unique, d'autres personnes sur Terre ont sûrement eu la même idée que nous. En revanche, si je la fais précéder de mon numéro de téléphone, on aura :

- 33667789078image

qui sera unique si je ne nomme qu'une seule ressource "image". Un autre utilisateur sur le même principe pourra nommer sa ressource :

- 33667239018image

sans ambiguïté possible. Cependant, comme le numéro de téléphone est unique dans l'espace des numéros de téléphone, d'autres numéros uniques pourraient entrer en conflit dans d'autres espaces de numérotation.

Pour éviter les conflits, il est intéressant de donner, au début l'espace de numérotation, par exemple :

- tel:33667789078image

et

- ss:33667789078image

On aura donc deux identifiants uniques, même si le hasard a fait que ce numéro de téléphone et ce numéro de sécurité sociale coïncident.

Les URI formalisent ce principe. Le [RFC 3986] explique comment ils peuvent être construits.

Un URI commence par un **schéma** indiquant l'autorité de nommage, suivi d'une **valeur d'autorité** puis d'**un chemin** dans l'espace d'autorité. Des caractères comme les ":" ou les "/" sont utilisés pour améliorer la lisibilité de l'URI.

Par **exemple** :

- mailto:mduerst@ifi.unizh.ch
- ssh://utilisateur@example.com
- ftp://ftp.is.co.za /rfc/rfc1808.txt

Ainsi, si je mets une ressource sur mon site Web, celui-ci est identifié par un **nom de domaine**, par exemple example.com. Je suis propriétaire de ce nom. Je peux donc l'utiliser pour identifier de manière unique ma ressource. Si on reprend le principe de construction d'un URI, j'aurai :

- http://example.com/ma_ressource

Personne d'autre dans l'univers ne pourra identifier ses ressources avec cette chaîne de caractères puisque *example.com* m'appartient. Je dispose donc d'un espace de nommage infini sous *example.com* qui me permet de désigner l'ensemble infini de ressources sans que personne d'autre ne puisse prendre les mêmes noms.

Un URI est une construction administrative permettant d'attribuer un identifiant unique global à une ressource spécifique.

L'URI a pour but de facilement nommer une ressource, de pouvoir lier les ressources entre elles pour former cette toile d'araignée mondiale. Le schéma définit à la fois l'espace de nommage de l'autorité et son format. Une adresse IP ou un nom de domaine comme autorité est à la fois un moyen d'assurer l'unicité globale, mais également de savoir comment accéder à la ressource.

Un sous-ensemble d'URI peut être directement utilisé pour localiser la ressource, c'est-à-dire trouver sur quel serveur se trouve la ressource et comment y accéder. Il s'agit d'une **URL** (**U**niform **R**esource **L**ocator) utilisée par les navigateurs Web. URI vs URL

Le schéma http est bien pratique car il peut se lire également comme un URL. Ce schéma donne :

- le **protocole** à utiliser pour accéder à la ressource (http),
- l'**autorité** qui indique l'**adresse du serveur** (et son **port**), et enfin,
- le **chemin d'accès** de ce que l'on va demander au serveur et qui peut parfois correspondre à une arborescence de fichiers sur un serveur.

3. Serveur sans état

Le principe REST permet de concevoir des serveurs évolutifs. **Un serveur doit être sans état**, ce qui signifie qu'**il ne conserve pas d'information après avoir répondu à une demande d'un client**. Cela permet de simplifier le traitement dans le serveur qui doit traiter les requêtes d'un grand nombre de clients.

Cela impose que l'état soit situé du côté du client. Cet état est alimenté à partir des données structurées que le client reçoit du serveur. Ainsi, lorsqu'un client demande une page Web, celle-ci peut contenir d'autres URI pour la compléter, par exemple des images, des feuilles de style, des scripts, etc.

Le client doit donc comprendre les données que le serveur lui envoie et donc connaître le format de représentation de la ressource qu'il reçoit pour y retrouver les URI. Donc, **en plus de la ressource elle-même, le serveur ajoute des informations complémentaires, appelées métadonnées**. Elles intègrent entre autres le format du contenu (content format). Il peut s'agir de texte pur, d'une image ou d'un format de texte structuré tel que **HTML** ou **JSON**.

HTTP est un protocole qui peut être utilisé pour mettre en œuvre un serveur Web en appliquant les principes de REST (qualifié en anglais de RESTfull).

HTTP définit différentes **méthodes** permettant au client d'interagir avec les ressources sur le serveur :

- **GET** est utilisée pour récupérer la représentation d'une ressource (par exemple page Web, valeur de température d'un capteur, etc.). Par exemple, la figure ci-dessous donne le format d'en-tête HTTP GET pour récupérer ma page Web :

```
▾ Hypertext Transfer Protocol
  ▸ GET /site/kamalsingh25/ HTTP/1.1\r\n
    Host: sites.google.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/45.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://sites.google.com/site/kamalsingh25/]
    [HTTP request 1/1]
```

- **HEAD** est utilisée pour récupérer uniquement les métadonnées présentes dans les en-têtes de réponse sans le corps de réponse ;
- **POST** est utilisée pour indiquer au serveur une nouvelle ressource ;
- **PUT** est utilisée pour stocker une ressource à l'endroit identifié par l'URI dans la requête. Si la ressource existe déjà, elle sera modifiée ;
- **PATCH** permet au client de ne modifier qu'une partie de la ressource ;
- **DELETE** est utilisée pour supprimer la ressource définie.