RESTFUL Web Services

1. Architecture orientée ressource

ROA ou ***Resource Oriented Architectures*** est une structure de conception supportant l'interconnexion de réseaux de ressources. Une ressource, dans ce contexte, désigne toute entité qui peut être identifié et attribué dans un URI (Uniform Resource Identifier).

Voici donc les **concepts** décrivant une architecture orientée ressource :

1.1 Ressources

Une ressource est quelque chose d'assez important pour être référencé comme étant une chose en elle-même, créer un lien hypertexte vers elle, faire ou réfuter des affirmations à son sujet, rechercher ou mettre en mémoire cache une représentation la concernant, l'inclure en partie ou dans son ensemble en la référençant dans une autre représentation, l'annoter ou encore effectuer d'autres opérations. [2]

Généralement, une ressource est quelque chose qui peut être stocké sur un ordinateur, un document, une ligne dans une base de données, ou le résultat de l'exécution d'un algorithme. [2]

1.2 URI (Uniform Resource Identifier)

Mais pour qu'une ressource soit connue, il faut qu'elle soit accessible et pour ça on a créé les URI qui sont à la fois le nom et l'adresse d'une ressource, sans URI pas de ressource. C'est ce qui fait la force du Web (HTTP), et ce qui lui a permis d'écraser une bonne partie des autres protocoles. Aujourd'hui tout passe par ce protocole au détriment des autres, on télécharge via un navigateur et non un client FTP par exemple. [2]

Comme les ressources sont identifiées par leurs URI, ça consiste généralement à fournir une liste d'URI possibles. Celles-ci pouvant être infinies comme par exemple dans le cas de Google : *http://www.google.com/search?**q=REST***. Grâce à cette propriété, on peut indiquer une adresse à quelqu'un sans lui dire : va sur google.com et taper « REST » dans le formulaire [2].

L'un des points importants de l'architecture orientée ressource est qu'une URI doit être descriptive. On doit savoir ce qui se cache derrière une URI rien qu'en la lisant, par exemple

en allant sur *http://www.monsite.com/liens/* vous savez que vous allez accéder à une page listant des liens.

Une ressource doit au moins avoir une URI mais rien n'empêche une ressource d'avoir plusieurs URI.

1.3 Représentation des ressources

Une représentation est une encapsulation de l'information (état, données, ou balisage) de la ressource, encodée en utilisant un format tel que XML, JSON, ou HTML.

1.4 Relation entre les ressources

Les représentations ne contiennent bien souvent pas seulement des données mais aussi des liens vers d'autres ressources. Résume bien le fait que nous naviguons sur internet grâce aux liens entre les ressources.

2 Web services RESTful

2.1 Architecture REST

2.1.1 Historique

REST estl'acronyme de "**RE**presentational**S**tate **T**ransfer" inventé par Roy T. Fielding danssa dissertation "an architecture style of networked systems". REST décrit les caractéristiques du web qui en ont fait son succès. L'explication de la signification de REST telle que donnée par Roy T. Fielding est la suivante : ‘Representational State Transfer’ évoque l'image du fonctionnement d'une application web bien construite : « un réseau de pages web où l'utilisateur progresse dans l'application en cliquant sur des liens (transition entre états) ce qui provoque l'affichage de la page suivante (représentant le nouvel état de l'application) à l'utilisateur qui peut alors l'exploiter. » [4]

2.1.2 Définition

REST est un style d'architecture, pas un standard. Il n'existe donc pas de spécifications de REST. Est un style d'architecture réseau pour web services qui met l'accent sur la définition de ressources identifiées par des URI, et utilise les messages du protocole HTTP pour définir la sémantique de la communication client/serveur.

Roy T. Fielding dit dans sa Thèse de doctorat : REST est un modèle hybride dérivé de plusieurs modèles basés sur les concepts réseau.

2.1.3 Les contraintes de REST défini par Roy T. Fielding

Voici les six (6) contrainte de REST sont défini par Roy T. Fielding [3] :

- **Client-Serveur :**

Les responsabilités sont séparées entre le client et le serveur. L'interface utilisateur est séparée de celle du stockage des données. Cela permet aux deux d'évoluer indépendamment.

- **Sans état (Stateless) :**

Chaque requête d'un client vers un serveur doit contenir toute l'information nécessaire pour permettre au serveur de comprendre la requête, sans avoir à dépendre d'un contexte conservé sur le serveur. Cela libère de nombreuses interactions entre le client et le serveur. L'état de la session est donc entièrement détenu par le client. [3]

Chaque nouvelle page affichée contient toutes les informations nécessaires pour afficher la ressource appropriée ou effectuer les traitements nécessaires, les requêtes ne doivent pas avoir d'ordre prédéfini et sont déconnectées les unes des autres. Chaque requête est totalement déconnectée à partir des autres. Le client peut faire des requêtes pour ces ressources n’importe quel nombre des fois, dans un ordre quelconque. De cette façon, le serveur n'a jamais besoin de connaître l'état du client, il ne sait même pas où il est, il sait juste qu'une requête arrivant avec tels paramètres doit restituer telles données. [2]

- **Mise en cache :**

Le serveur envoie une réponse qui donne l'information sur la propension de cette réponse à être mise en cache, comme la fraîcheur, sa date de création, si elle doit être conservée dans le futur. Cela permet à des serveurs mandataires de décharger les contraintes sur le serveur et aux clients de ne pas faire de requêtes inutiles. Cela permet également d'améliorer l'extensibilité des serveurs.

- **Interface uniforme :**

Toutes les ressources sont accessibles par une interface générique (par exemple, HTTP GET, POST, PUT, DELETE).

- **Un système hiérarchisé par couche :**

Afin d'améliorer le comportement de l'architecture face aux besoins de grande échelle (internet), nous ajoutons des contraintes de système en couches. Le modèle de système en couches permet à une architecture de se composer de couches hiérarchiques en contraignant le comportement des composants. Chaque composant ne peut pas «voir» au delà de la couche immédiate avec laquelle il interagit. En limitant la connaissance du système à une seule couche, nous mettons une limite sur la complexité du système global et favorisons l'indépendance des couches. Les couches peuvent être employées pour encapsuler des services déjà en place et protéger les nouveaux services des clients existants, en simplifiant les composants par le déplacement de fonctionnalités rarement utilisées dans un espace intermédiaire partagé. Des intermédiaires peuvent également être employés pour améliorer la montée en charge du système en offrant un équilibrage de charges pour les services et ce à travers de multiples réseaux et processeurs [1].

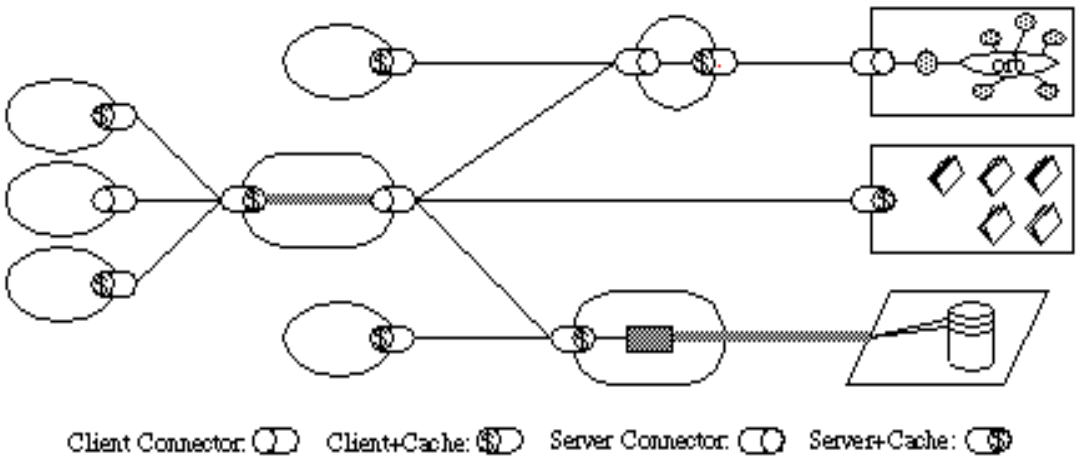

Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

- **Code à la demande (facultatif) :**

Exécution de scripts côté client obtenus par le serveur. Cela permet de rendre le client plus léger et plus générique. Permet l'extension des fonctionnalités d'un client par le biais de téléchargement et d'exécution de code sous forme d'applet ou de scripts, cela simplifie les clients en réduisant le nombre de fonctionnalités qu'ils doivent mettre en œuvre par défaut.

2.2 Les éléments de REST

Le style REST est une abstraction d'éléments architecturaux dans un système hypermédia distribué. REST ignore les détails de la mise en œuvre des composants et de la syntaxe du protocole afin de se concentrer sur les rôles des composants, les contraintes lors de leur interaction avec d'autres composants et leur interprétation à éléments de données significatifs.Il englobe les contraintes fondamentales sur les composants, les connecteurs et les données qui définissent la base de l'architecture Web.

2.2.1 Les éléments de données

Les éléments de données de REST sont résumés comme suit :

- Ressource
- Identifiant de ressource
- Représentation
- Métadonnées de représentation
- Métadonnées de ressource
- Données de contrôle

a) URI comme identifiant des ressources

REST se base sur les URI (Uniform Resource Identifier) afin d'identifier une ressource. Ainsi une application se doit de construire ses URI de manière précise, en tenant compte des contraintes REST. Il est nécessaire de prendre en compte la hiérarchie des ressources et la sémantique des URL pour les éditer.

En construisant correctement les URI, il est possible de les trier, de les hiérarchiser et donc d'améliorer la compréhension du système. [5]

L'URL suivante peut alors être décomposée logiquement :

`/serviceEtud/etudiants`→ tous les étudiants

`/serviceEtud/etudiants/39` → un étudiant (numéro 39)

`/serviceEtud/etudiants/39/messages`→ tous les messages d'un étudiant

`/serviceEtud/etudiants/39/messages/12`→un message de l'étudiant N° 39

b) Représentation des ressources

Le principe en pratique est de passer de la représentation utilisée en interne, que ce soit sur le client ou le serveur, à la représentation utilisée dans le message HTTP, requête ou réponse.

Pour les Web Services RESTful, les encodages les plus utilisés sont **XML** et **JSON** (JavaScript Object Notation).

Remarque JSON

C'est un format texte qui permet de représenter des données et de les échanger facilement à l'instar d'XML. Ce sous ensemble de JavaScript permet de décrire le modèle objet de JavaScript. Deux types de structures sont disponibles :

- Objet : une collection de paire nom/valeur, i.e. un tableau associatif.
- Tableau : une liste ordonnée de valeurs.

La syntaxe est celle de JavaScript, simpliste. Voici par exemple la description d'un étudiant avec format JSON :

```
[ {
      "id":7,
      "nom":"Bendahi",
      "prenom": "mohamed",
      "sexe": "homme",
      "date_naissanse": "10/10/1991",
      "telephone": "066666666",
      "adresse":"Biskra",
      "email":"bmohamed1991@gmail.com",
}]
```

c) Données de contrôle

Les données de contrôle définissent le but d'un message entre les composants, tels que l'action demandée ou la signification d'une réponse. Il sert également à paramétrer les requêtes et à remplacer le comportement par défaut de certains éléments de connexion. Par exemple, le comportement du cache peut être modifié par les données de contrôle incluses dans le message de la requête ou de la réponse.

Selon les données de contrôle d'un message, une représentation donnée peut indiquer l'état actuel de la ressource demandée, l'état souhaité pour la ressource demandée ou la valeur d'une autre ressource, telle qu'une représentation des données d'entrée dans le formulaire d'une requête d'un client, ou une représentation d'une condition d'erreur pour une réponse.

2.3.4. Relation entre ressources

Les liens d'une ressource vers une autre ont tous une chose en commun : ils indiquent la présence d'une relation. Il est cependant possible de la décrire afin d'améliorer la

compréhension du système. Pour expliciter cette description et indiquer la nature de la relation, l'attribut *rel*doit être spécifié sur tous les liens. Ainsil'IANA (*Internet Assigned Numbers Authority*) donneuneliste de relation (self, next, edit, last, payment, content…) [5].

Exemple

```
<?xml>
<search>
  <link rel="self"title="self"href="http://mywebsite.com/
serviceEtud/etudiants?q=formation&nm=STIC&nv=M1"/>
  <link rel="next"title="next"href="http://mywebsite.com/
serviceEtud/etudiants?q=formation&nm=STIC&nv=M2"/>
    <Etudiants>
     //...
</search>
```

2.2.2 Les Connecteurs

REST utilise différents types de connecteurs, pour encapsuler les activités d'accès aux ressources et le transfert des représentations de ressources. Les connecteurs présentent une interface abstraite pour la communication des composants, améliorant la simplicité en fournissant une séparation propre des préoccupations et cachant l'implémentation sous-jacente des ressources et des mécanismes de communication. Les différents types de connecteurs sont:

- Client
- Serveur
- Cache : cache du navigateur, ….
- Résolveur : DNS
- Tunnel : SOCKS, SSL ..

2.2.3 Les Composants

Les Composants de REST sont :

- Serveur d'origine
- Passerelle
- Proxy
- Agent utilisateur

Un agent utilisateur utilise un connecteur client pour lancer une demande et devient le destinataire final de la réponse. L'exemple le plus courant est un navigateur Web, qui permet d'accéder aux services d'information et rend les réponses au service en fonction des besoins de l'application.

Un serveur d'origine utilise un connecteur de serveur pour gérer l'espace de noms pour une ressource demandée. C'est la source définitive de représentation de ses ressources et doit être le destinataire final de toute demande visant à modifier la valeur de ses ressources. Chaque serveur d'origine fournit une interface générique à ses services en tant que hiérarchie des ressources. Les détails de l'implémentation des ressources sont cachés derrière l'interface.
Les composants intermédiaires agissent à la fois comme un client et un serveur afin de transmettre, avec des traductions possibles, des demandes et des réponses. Un composant proxy est un intermédiaire sélectionné par un client pour fournir l'encapsulation d'interface d'autres services, la traduction de données, l'amélioration de la performance ou la protection de sécurité. Un composant passerelle (reverse proxy) est un intermédiaire imposé par le réseau ou le serveur d'origine pour fournir une encapsulation d'interface à d'autres services, pour la traduction de données, l'amélioration des performances ou l'application de la sécurité.

2.3. Principe des web services RESTful

Régulièrement, lire REST ou RESTful lorsque la technologie est abordée. En bref, il n'y a aucune différence, RESTful est simplement l'adjectif qui qualifie une architecture de type REST.

2.3.1. Méthodes HTTP

Ainsi, généralement pour une ressource, il y a 4 opérations possibles et HTTP propose les verbes correspondant :

- Créer (Create) => POST
- Afficher (Read) => GET
- Mettre à jour (Update) => PUT
- Supprimer (Delete) => DELETE

Exemple d'URI pour une ressource donnée (une Actualité par exemple), pour ajouter ou modifier ou supprimer une ressource on peut utiliser la méthode GET de manière suivante par exemple :

a- `GET 127.0.0.1:8090/serviceEtud/actualite/afficher?id=7`

b- `GET 127.0.0.1:8090/serviceEtud/actualite/supprimer?id=7`

c- `GET 127.0.0.1:8090/serviceEtud/actualite/ajouter?titre=Constantine2015&texte=capitaleDeLaCultureArabe2015…`

Dans les cas « **a** » et « **b** » nous devons modifier les méthodes HTTP, pour respecter l'architecture REST et l'URI devenudescriptif :

`GET127.0.0.1:8090/serviceEtud/actualite/7-` Afficher l'actualité numéro 7

`DELETE127.0.0.1:8090/serviceEtud/actualite/7-` Supprimer l'actualité 7

Le cas « **c** » il y a un problème !, il faut ne pas faire circuler des informations clairement sur le réseau. La simple moyen pour éviter ce problème et de envoyer les noms et les valeurs des paramètres de la requête HTTP dans formatJSON ou balise XML à partir du « Request body ».

Par exemple le cas « **c** » ajouter actualité (utilisation de la méthode POST)

```
POST /serviceEtud/actualite  HTTP/1.1
Host :127.0.0.1 :8080
Content-Type: application/json

{
  "titre"="constantine2015"
  "texte"= "capitale de la culture arabe 2015"
}
```

Request Header

RequestBody

Résumé

Pour développer une application Web REST, vous devez utiliser :

- **L'URI comme identifiant des ressources (conception des routes et des URIs)**
- **Les verbes HTTP comme identifiant des opérations**
- **Les réponses HTTP comme représentation des ressources**
- **Les liens comme relation entre ressources**
- **Pour assurer la contrainte « stateless », utiliser un paramètre comme jeton d'authentification (Token comme JSON Web Tokens (JWT)**

En plus, vous devez prendre en considération le mécanisme de cache et l'exploitation des entêtes HTTP.

Références

[1] Traduction du chapitre 5 de la thèse de doctorat de Roy Thomas Fielding, réalisée par Frédéric Laurent, http://opikanoba.org/tr/fielding/rest/

[2] Leonard Richardson et SAM Ruby, « *RESTful Web Services* », O'Reilly Media,2007.

[3] Roy Thomas Fielding, Thèse de doctorat: « *Architectural Styles and the Design of Network-based Software Architectures, Representational State Transfer REST* », Chapitre 5, Université de California, Irvine, 2000.
[4] Jean-Paul Figer, « REST un style d'architecture », http://www.figer.com/Publications/REST.htm

[5] http://blog.nicolashachet.com/niveaux/confirme/larchitecture-rest-expliquee-en-5-regles