**REST (Representational State Transfer) ou RESTful** est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service). Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière. L'architecture REST utilise les spécifications originelles du **protocole HTTP**, plutôt que de réinventer une surcouche (comme le font SOAP ou XML-RPC par exemple).

- Règle n°1 : l'URI comme identifiant des ressources
- Règle n°2 : les verbes HTTP comme identifiant des opérations
- Règle n°3 : les réponses HTTP comme représentation des ressources
- Règle n°4 : les liens comme relation entre ressources
- Règle n°5 : un paramètre comme jeton d'authentification

# 5 règles à suivre pour implémenter REST

## Règle n°1 : l'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 (et donc ses URL) 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 :

Quelques exemples de construction d'URL avec RESTful :
**Liste des livres**

*NOK : http://mywebsite.com/book*
*OK : http://mywebsite.com/books*
**Filtre et tri sur les livres**

*NOK : http://mywebsite.com/books/filtre/policier/tri/asc*
*OK : http://mywebsite.com/books?fitlre=policier&tri=asc*
**Affichage d'un livre**

*NOK : http://mywebsite.com/book/display/87*
*OK : http://mywebsite.com/books/87*
**Tous les commentaires sur un livre**

*NOK : http://mywebsite.com/books/comments/87*
*OK : http://mywebsite.com/books/87/comments*
**Affichage d'un commentaire sur un livre**

*NOK : http://mywebsite.com/books/comments/87/1568*
*OK : http://mywebsite.com/books/87/comments/1568*
En construisant correctement les URI, il est possible de les trier, de les hiérarchiser et donc d'améliorer la compréhension du système.

L'URL suivante peut alors être décomposée logiquement :

*http://mywebsite.com/books/87/comments/1568* => ***un commentaire pour un livre***
*http://mywebsite.com/books/87/comments* => ***tous les commentaires pour un livre***
*http://mywebsite.com/books/87* => ***un livre***
*http://mywebsite.com/books* => ***tous les livres***

# Règle n°2 : les verbes HTTP comme identifiant des opérations

La seconde règle d'une architecture REST est d'utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource. Ainsi, généralement pour une ressource, il y a 4 opérations possibles (CRUD) :

- Créer (create)
- Afficher (read)
- Mettre à jour (update)
- Supprimer (delete)

HTTP propose les verbes correspondant :

- Créer (create) => **POST**
- Afficher (read) => **GET**
- Mettre à jour (update) => **PUT**
- Supprimer (delete) => **DELETE**

Exemple d'URL pour une ressource donnée (un livre par exemple) :

**Créer un livre**

*NOK : GET http://mywebsite.com/books/create*
*OK : POST http://mywebsite.com/books*
**Afficher**

*NOK : GET http://mywebsite.com/books/display/87*
*OK : GET http://mywebsite.com/books/87*
**Mettre à jour**

*NOK : POST http://mywebsite.com/books/editer/87*
*OK : PUT http://mywebsite.com/books/87*
**Supprimer**

*NOK : GET http://mywebsite.com/books/87/delete*
*OK : DELETE http://mywebsite.com/books/87*

# Règle n°3 : les réponses HTTP comme représentation des ressources

Il est important d'avoir à l'esprit que la réponse envoyée n'est pas une ressource, c'est la représentation d'une ressource. Ainsi, une ressource peut avoir plusieurs représentations dans des formats divers : **HTML, XML, CSV, JSON, etc.**

C'est au client de définir quel format de réponse il souhaite reçevoir via l'entête **Accept**. Il est possible de définir plusieurs formats.

Quelques exemples :

**Réponse en HTML**

```
GET /books
Host: mywebsite.com
Accept: text/html
```

**Réponse en XML**

```
GET /books
Host: mywebsite.com
Accept: application/xml
```

# Règle n°4 : les liens comme 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 spécifir sur tous les liens. Ainsi l'IANA donne une liste de relation parmi lesquelles :

- contents
- edit
- next
- last
- payment
- etc.

La liste complète sur le site de l'IANA : http://www.iana.org/assignments/link-relations/link-relations.xml

# Règle n°5 : un paramètre comme jeton d'authentification

C'est un des sujets les plus souvent abordé quand on parle de REST : comment authentifier une requête ? La réponse est très simple et est massivement utilisée par des APIs renommées (Google, AWS, etc.) : le **jeton d'authentification**.

Chaque requête est envoyée avec un jeton (token) passé en paramètre $_GET de la requête. Ce jeton temporaire est obtenu en envoyant une première requête d'authentification puis en le combinant avec nos requêtes.

Ainsi, on peut construire le scénario suivant :

## *1. demande d'authentification*

GET /users/123/authenticate?pass=lkdnssdf54d47894f5123002fds2sd360s0

Code

```
<?xml>
<user>
  <id>123</id>
  <name>Nicolas Hachet</name>
</user>
<token>
  fsd531gfd5g5df31fdg3g3df45
</token>
```

## *2. accès aux ressources*

Cet token est ensuite utilisé pour générer un hash de la requête de cette façon :

Code

```
hash = SHA1(token + requete)
```

```
2  hash = SHA1(fsd531gfd5g5df31fdg3g3df45 + "GET /books")

3  hash = 456894ds4q15sdq156sd1qsd1qsd156156
```

C’est ce hash qui est passé comme jeton afin de valider l’authentification pour cette requête:

Code

```
1  GET /books?user=123&hash=456894ds4q15sdq156sd1qsd1qsd156156
```