Architecture et Services Web RESTful

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-11-01

Table des matières

1 L'adressage des ressources 2
2 Le protocole d'échange 3
3 La représentation des ressources 5
4 Les code de retours 6
5 Un échange REST = un échange HTTP 7
6 Quelques exemples complets 7
7 Utilisation 8

```
[INFO] Target directory: /home/jovyan/work/github/ebpro/sample-jaxrs
[INFO] Cloning new repository to /home/jovyan/work/github/ebpro/sample-jaxrs
[INFO] Repository cloned successfully
```

Les exemples suivants sont accessibles dans le dépôt :

: ebpro/sample-jaxrs

Détails
- **URL**: https://github.com/ebpro/sample-jaxrs
- **Branch**: develop
- **Local Path**: `/home/jovyan/work/github/ebpro/sample-jaxrs`
- **Latest Commit**: 7331765 - chore: add _quarto-utils submodule + .devcontainer (SOTA pattern)
- **Timestamp**: 2026-11-01T12:18:09Z

Clone Command:

```
git clone -b develop https://github.com/ebpro/sample-jaxrs
```

Ce document présente les service Web REST en général et par la pratique en Java.

Il s'appuie sur un exemple simple d'application : https://github.com/ebpro/sample-jaxrs qui servira à illustrer les notions et sera étudiée en détail dans la partie pratique.

L'idée générale de l'approche REST (REpresentational State Transfer) pour construire une interface de programmation (API) est d'offrir un accès distant à des ressources via une interface commune construite ~~au dessus de~~ « **en** » http. On parle d'approche RESTful quand l'interface d'une application est conforme à une certaine philosophie (il ne s'agit pas d'une norme).

En particulier, le fait que l'API est sans état côté serveur permet de l'utiliser de façon transparente même en cas de serveur proxy/cache.

> ! Important
>
> RESTfull est une approche d'API client/serveur suivant la logique de navigation dans un hypermedia. On parle d'HATEOS (Hypermedia As The Engine Of Application State).

Pour définir un protocole de communication, il faut généralement définir :

- un système d'identification (d'adressage) des ressources manipulées,
- un protocole de communication,
- un format d'échange de données éventuellement typées,
- un système de gestion des erreurs.

La logique RESTfull est d'utiliser tout ce que propose HTTP pour écrire une API **en** HTTP.

1 L'adressage des ressources

> ! Important
>
> Les ressources (ou ensembles de ressources) de l'application sont identifiées par des URI. Les URL sont une sorte particulière d'URI qui indique un moyen d'accès en plus de les identifier de façon unique.

Il n'y a pas de standard pour les API REST. Il vaut généralement mieux rester simple et cohérent. Quelques pratiques sont utilisées classiquement :

- On utilise des noms (pas des verbes) au pluriel pour les ressources :

```
# Toutes les personnes
http://MyServer/MyApp/Persons
```

- La ou les informations qui permettent d'identifier une ressource sont inclues dans l'URL et le plus possible dans le chemin plutôt que dans la Query String (par exemple un identifiant) :

```
# La personne d'identifiant 1
http://MyServer/MyApp/Persons/1
```

- On évite les « jointures » dans les chemins. Si on le fait, l'ordre doit être constant et logique (la fin du chemin correspond toujours à la ressource retournée).
  - Les chiens de la personne 1
    - `http://MyServer/MyApp/Persons/1/Dogs`
    - `http://MyServer/MyApp/Dogs?master_id=1`
- On utilise la pagination, le filtrage et le tri (via les Query Strings) pour les requêtes complexes et pour contrôler le volume des données retournées. A noter la possibilité d'utiliser les Matrix Params même s'ils ne sont pas standards.
  - La deuxième page de personnes en utilisant des pages de 10 personnes.
    - `http://MyServer/MyApp/Persons?page=2&page_size=10`
    - `http://MyServer/MyApp/Persons;page=2;page_size=10` (avec des Matrix Params)
  - filtre qui trie par ordre decroissant de date de création, puis par titre
    - `http://MyServer/MyApp/Persons?page=&page_size=10&sort=name,firstname,-created,title`
- Une projection explicite de certains champs des données peut être envisagée dans la ressource.
  - La personne d'identifiant 1 restreinte uniquement à certains champs
    - `http://MyServer/MyApp/Persons/1?fields=email,firstname,lastname`
- Des éléments « administatifs » peuvent/doivent être proposés (par exemple pour gérer la version d'une API en ajoutant /api/v1, /api/v2, ... au début du chemin).

2 Le protocole d'échange

> ! Important
>
> Les actions sur les ressources (identifiées par des URIs) sont associées aux verbes (méthodes) standards du protocole HTTP.

REST s'appuie sur le protocole HTTP (HyperText Transfert Protocol) qui est défini dans les RFC 7230 à 7237. La RFC 7231 défini les méthodes et les codes de retour.

Une méthode est « sure » (safe) si elle ne modifie pas l'état de serveur.

Une méthode est idempotente (idempotent) si l'effet attendu par des appels multiples est identique à un appel unique de la même requête. Les méthodes sures sont donc idempotentes.

Verbe HTTP	Utilisation	Contraintes
GET	Accès à une ressource identifiée dans l'URL (il peut s'agir d'une collection).	Safe, Idempotent
HEAD	comme GET mais sans le corps de la requête (seul le header http est retourné). Utile pour savoir si une ressource a changé.	Safe, Idempotent
POST	création d'une ressource sans donner l'identifiant.	
PUT	mise à jour complète d'une ressource identifiée (voire création en donnant l'identifiant).	Idempotent
DELETE	suppression d'une ressource.	Idempotent
OPTIONS	liste les actions possibles sur une ressource.	Safe, Idempotent
PATCH	RFC 5789, mises à jour partielle d'une ressource.	

Donc en particulier GET ne modifie rien, plusieurs appels à PUT n'ajoutent ou n'appliquent une modification qu'une fois mais plusieurs appels à POST en ajoutent plusieurs.

Attention, certains proxies HTTP peuvent empêcher certaines actions en dehors de GET et POST (cf. X-HTTP-Method-Override). Cela peut donc conduire à devoir enfreindre les règles ci-dessus.

Un **endpoint** REST est défini par un verbe HTTP et une URL.

- **Obtenir toutes les personnes** :
  - `GET http://MyServer/MyApp/Persons`
- **Obtenir une personne précise par identifiant** :
  - `GET http://MyServer/MyApp/Persons/1`
- **Obtenir toutes les personnes entre 7 et 16ans** (avec un filtre) :
  - `GET http://MyServer/MyApp/Persons?ageMin=7&ageMax=16`
- **Supprimer toutes personnes**
  - `DELETE http://MyServer/MyApp/Persons`
- **Supprimer une personne**
  - `DELETE http://MyServer/MyApp/Persons/1`

3 La représentation des ressources

> ! Important
>
> Les resources sont généralement représentées et échangées à l'aide de langages autodescriptifs comme XML ou JSON.

Par exemple, une personne peut être présentée :

```xml
<?xml version='1.0'?>
<person id='1'>
    <lastname>Doe</lastname>
    <firstname>John</firstname>
</person>
```

```json
{
  "person": {
    "-id": 1,
    "lastname": "Doe",
    "firstname": "John"
  }
}
```

Les types de données envoyées ou attendues sont indiqués dans l'entête de la requête HTTP par `Content-Type:` et `Accept:`. Pour cela, on utilise les Internet Media Types (ex MIME Type - Multipurpose Internet Mail Extensions). Il s'agit d'une liste

standard de formats et de sous-formats d'échange de données (text/plain, text/xml, application/json, ...).

L'exemple ci-dessous sérialise des objets Java qui représente un auteur et un livre en JSON et en XML. (Le détail est expliqué plus loin).

```
CompilationException:
|   import fr.univtln.bruno.samples.jaxrs.model.Library;
package fr.univtln.bruno.samples.jaxrs.model does not exist
```

4 Les code de retours

Le protocole HTTP défini un ensemble de codes de retours. Il est donc possible d'utiliser ces codes standards comme code de retour pour indiquer comment l'opération a réussi ou pourquoi elle a échoué.

> ! Important
>
> Le code de retour des méthode est un code HTTP. Il est indiqué de façon standard dans l'entête de la réponse et peut être répété dans le contenu si une enveloppe est proposée.

- Le code de retour des méthodes est un code HTTP.
- Il est indiqué de façon standard dans l'entête de la réponse.
- Il peut être répété dans le contenu si une enveloppe est proposée.

Code	Signification	Usage
200	Ok	Requête traitée avec succès.
201	Created	Nouvelle ressource créée.
204	No Content	Pas de contenu, pas exmple lors d'une requête DELETE réussie.
206	Partial Content	Seulement une partie de résultat est retourné par exemple en cas de pagination (non explicite).
304	Not Modified	Utilisation du cache possible.

Code	Signification	Usage
400	Bad Request	La requête est invalide et ne peut pas être traitée par le serveur.
401	Unauthorized	La requête nécessite que le client soit authentifié.
403	Forbidden	Le client est authentifié mais l'utilisateur n'est pas autorisé à accéder à cette ressource.
404	Not Found	La ressource demandée n'existe pas.
500	Internal Server Error	C'est une erreur générique de fonctionnement, elle devrait toujours être accompagnée d'une description

5 Un échange REST = un échange HTTP

> ! Important
>
> Un échange d'une API REST correspond donc exactement à un échange http.

- Une requête HTTP
  - composée d'un verbe, d'une URL, de la version de HTTP utilisée, d'un en-tête (une liste de couples `nom:valeur` par exemple `Content-Type: text/xml`) et un corps éventuellement vide (les données envoyées).
- Une réponse HTTP
  - composée d'un code de retour, de meta données dans l'en-tête et d'un corps qui contient les données. Le corps est éventuellement encapsulé dans une « enveloppe » qui reprend les meta données pour faciliter leur traitement.

6 Quelques exemples complets

Requête de création d'une personne :

```
POST http://MyServer:8080/MyApp/Persons/
Host: MyServer:8080
```

```
Content-Type: application/json; charset=utf-8
Content-Length: 36
{"lastname": "Doe",
 "firstname": "John"}
```

Requête de modification d'une personne (`id` dans l'URL):

```
PUT http://MyServer:8080/MyApp/Persons/1
Host: MyServer:8080
Content-Type: application/json; charset=utf-8
Content-Length: 12
{"age":"18"}
```

7 Utilisation

Une requête REST peut être envoyée par programmation ou en utilisant un programme dédié comme curl en ligne de commande, `postman` pour chrome ou `RestClient` pour firefox.

Regardez les options de la commande `curl` pour réaliser des requêtes HTTP.

```
%%shell
curl -H "Accept: application/json" \
     -H "User-Agent: Demo Script" \
      -X GET "https://nominatim.openstreetmap.org/search?q=Universit%C3%A9+de+
Toulon&format=json&limit=1"
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100   550  100   550    0     0   2807      0 --:--:-- --:--:-- --:--:--  2806
```

```
[{"place_id":82758865,"licence":"Data © OpenStreetMap contributors, ODbL 1.0.
http://osm.org/copyright","osm_type":"way","osm_id":288649901,"lat":"43.1354075","lon":"6.0161592
de Toulon","display_name":"Université de Toulon, Avenue André
Citroën, La Valette-du-Var, Toulon, Var, Provence-Alpes-
Côte d'Azur, France métropolitaine, 83160, France","boundingbox":
["43.1328214","43.1379869","6.0092534","6.0254597"]}]
```