REST

Olivier Coupelon

2021-2022

Technologies de communication inter-applications

- API : on embarque directement l'application à appeler (dépendance directe, JAR)
- EJB : on appelle l'application avec un protocole de niveau transport (TCP/IP) ou directe (si même JVM)
- REST : échange niveau HTTP, on exploite à 100% le protocole
- SOAP : au dessus d'HTTP, on encapsule les messages
- MQTT : sur TCP/IP, pour l'IoT
- ...

REST

> Representational State Transfer

Roy Fielding http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm → Exploitation de la sémantique du Web, de la mécanique HTTP

Principe

On utilise les méthodes HTTP sur les ressources exposées afin d'effectuer toutes les opération voulues. On est donc toujours sans état.

Technologies utilisées :

- HTTP
- URI
- XML / JSON (représentation des ressources)
- Types MIME

XML vs JSON

Les deux, mais rarement en même temps : * XML est validé, mais très verbeux * JSON est plus souple

→ On peut utiliser aussi du protobof ou tout autre format de contenu, ce sont simplement les plus utilisés.

Utilisation de REST

Figure 1: Utilisation de REST

Communication REST

CRUD	REST	
CREATE	POST	Create a sub resource
READ	GET	Retrieve the current state of the resource
UPDATE	PUT	Initialize or update the state of a resource at the given URI
DELETE	DELETE	Clear a resource, after the URI is no longer valid

Figure 2: CRUD

Exemple : Création d'un sondage

```
POST /sondage
<options>A,B,C</options>
```

```
201 Created
Location: /sondage/672609683
```

Exemple : Récupération

```
GET /sondage/672609683
```

```
200 OK
<options>A,B,C</options>
```

Exemple : Vote au sondage

```
POST /sondage/672609683
```

```
<name>O. Coupelon</name>
<choice>B</choice>
```

```
201 Created
Location: /sondage/672609683/vote/1
```

Exemple : Récupération

```
GET /sondage/672609683
```

```
200 OK
<options>A,B,C</options>
    <votes>
        <vote id='1'>
        <name>O. Coupelon</name>
        <choice>B</choice>
    </vote>
</votes>
```

Exemple : Modification d'un sondage

```
PUT /sondage/672609683/vote/1
<name>O. Coupelon</name>
<choice>C</choice>
```

```
200 OK
```

Exemple : Récupération

```
GET /sondage/672609683
```

```
200 OK
<options>A,B,C</options>
    <votes>
        <vote id='1'>
        <name>O. Coupelon</name>
        <choice>C</choice>
    </vote>
</votes>
```

Exemple : Suppression d'un sondage

```
DELETE /sondage/672609683
```

```
200 OK
```

Exemple : Récupération

```
GET /sondage/672609683
```

```
404 Not Found
```

URL

`http://server:port/path/to/resource?p1=v1&p2=v2`

- Ressource : ce que l'on cherche
- Paramètres : les modes de recherche
- Une fois définie, une URL ne doit plus changer

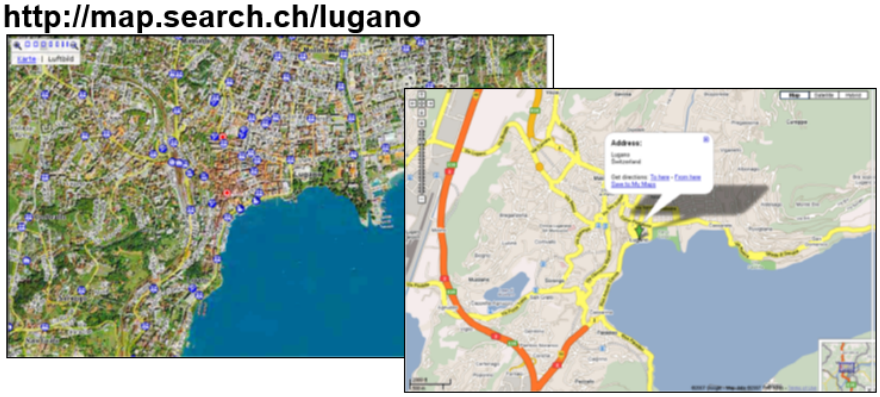

Figure 3: URL Propre

HATEOAS

REST utilise des URL opaques : HATEOAS

> Hypermedia As The Engine Of Application State

- Concept unique, relativement aux autre protocoles
- Un client n'a besoin d'aucune information pour dialoguer avec le service.
- Concept au cœur de la thèse de Roy Fielding : Representational State Transfer

Implémentations : JSON-LD (Linked Documents), HAL (Hypertext Application Language)

```
{
    "_links": {
        "self": { "href": "https://api.example.com/player/1234567890" },
        "friends": { "href": "https://api.example.com/player/1234567890/friends" }
    },
    "playerId": "1234567890",
    "name": "Kevin Sookocheff",
    "alternateName": "soofaloofa",
    "image": "https://api.example.com/player/1234567890/avatar.png"
}
```

Création d'une ressource REST

- Identification des ressources
- Définition d'url propres
- Pour chaque url, prévoir les actions pour GET, POST, PUT & DELETE
- Documenter chaque ressource
- Définir les liens entre ressources
- Déployer et tester chaque ressources sur un serveur web

Comment gérer le versionning ?

Négociation de contenu

- On peut ajouter en entête dans la requête le type de contenu qu'on souhaite recevoir, par ordre de référence :

```
GET /resource
Accept: text/html, application/xml, application/json
```

```
200 OK Content-Type: application/json
```

Exemple de gestion de version :

```
Content-Type: application/x.example.product+json; version=2
```

Bonnes pratiques - GET

- Operation de lecture
- Idempotente
- Possibilité de cache

Bonnes pratiques - POST

- Operation de création
- **Effet de bord** attendu sur le serveur

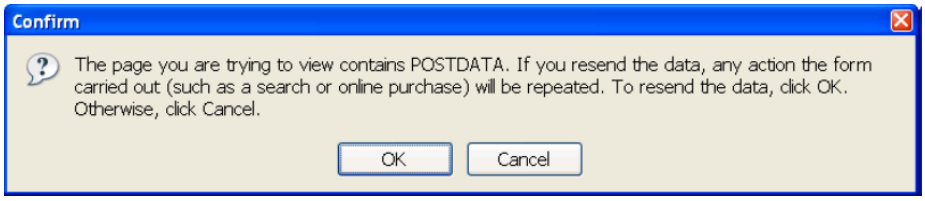

Figure 4: Annulation d'un POST

Bonnes pratiques - POST vs PUT

- On ne créer pas une ressource avec PUT. Pour la création d'un vote, préferer :

```
POST /sondage/672609683
```

```
201 CREATED
Location: /sondage/672609683/vote/1
```

À

```
PUT /sondage/672609683/vote/1
```

```
200 OK
```

Bonnes pratiques - Retours d'information

- Toujours respecter les codes retour HTTP.
- Doivent être interprétés correctement par le client, et émis correctement par le serveur.

WADL

Web Application Description Language

- Interface décrivant les opérations disponibles d'un service REST, ses entrées/sorties...
- Créé automatiquement par les frameworks REST (Jersey...)

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://research.sun.com/wadl/2006/10 wadl.xsd"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ex="http://www.example.org/types"
     xmlns="http://research.sun.com/wadl/2006/10">

    <resources base="http://www.example.org/services/">
        <resource path="getStockQuote">
            <method name="GET">
                <request>
                    <param name="symbol" style="query" type="xsd:string"/>
                </request>
                <response>
                    <representation mediaType="application/xml"
                       element="ex:quoteResponse"/>
                    <fault status="400" mediaType="application/xml"
                       element="ex:error"/>
                </response>
            </method>
        </resource>
    </resources>
</application>
```

Figure 5: wadl

RSDL

RESTful Service Description Language

- Description des points d'entrée des services (URI) uniquement

```
<link rel="get" href="/api/clusters">
    <request>
        <http_method>GET</http_method>
        <headers>
            <header required="false">
                <name>Filter</name>
                <value>true|false</value>
            </header>
        </headers>
        <url>
            <parameters_set>
                <parameter context="query" type="xs:string" required="false">
                    <name>search</name>
                    <value>search query</value>
                </parameter>
                <parameter context="matrix" type="xs:boolean" required="false">
                    <name>case_sensitive</name>
                    <value>true|false</value>
                </parameter>
                <parameter context="matrix" type="xs:int" required="false">
                    <name>max</name>
                    <value>max results</value>
                </parameter>
            </parameters_set>
        </url>
        <body/>
    </request>
    <response>
        <type>Clusters</type>
    </response>
</link>
```

Figure 6: rsdl

OpenAPI

Implémentation historique : Swagger

Le standard de description des API *REST*.

Démo : https://petstore.swagger.io/

Gestion des états dans REST

- HTTP est Stateless
- Ajout d'information dans les ressources retournées représentant les transitions valides
- Technique HTTP classiques de sessions

→ L'état doit être porté par les Hypermedia, sinon ce n'est pas du REST.

Sécurité

- Utilisation de la sécurité HTTP
- TLS
- Authentification

Asynchronisme

- HTTP est un protocole synchrone
- On peut simuler une file simplement :

```
POST /queue
```

```
202 Accepted Location: /queue/message/1
```

```
GET /queue/message/1
```

Implémentation Java : JAX-RS

> Java API for RESTful Web Services

- Spécification technique (JSR 311) côté serveur
- JAX-RS 1.1 : Java EE 6
- JAX-RS 2.0 : http://jax-rs-spec.java.net/ → Java EE 7
- JAX-RS 2.1 : http://jax-rs-spec.java.net/ → Java EE 8

Implémentations de JAX-RS

- Jersey (Implémentation de référence)
- RESTEasy (JBoss)
- Restlet
- Apache CXF

Annotations

- JAX-RS utilise des annotations Java pour exposer les services
- Masque tous les traitements HTTP
- @Path : Chemin d'accès à la ressource

- @GET, @PUT, @POST, @DELETE : Type de requête
- @Consumes : Type de donnée en entrée
- @Produces : Type de donnée en sortie
- @PathParam, @QueryParam, @HeaderParam, @CookieParam, @MatrixParam, @FormParam

---

Clients REST

Un client REST très commun est le navigateur Web → Communication back vers front

Pour faire communiquer des applications REST ou effectuer des tests unitaires, des implémentations clientes existent : *SOAP UI* Jersey Client * Curl * ...

JSON:API

Spécification de mai 2015 : https://jsonapi.org/format/

- API destinée principalement à la lecture de ressources
- Déclaration

```
Content-Type: application/vnd.api+json
```

Exemple

```
GET /articles?include=author HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "JSON:API paints my bikeshed!",
      "body": "The shortest article. Ever.",
      "created": "2015-05-22T14:56:29.000Z",
      "updated": "2015-05-22T14:56:28.000Z"
    },
```

```
    "relationships": {
      "author": {
        "data": {"id": "42", "type": "people"}
      }
    }
  }],
  "included": [
    {
      "type": "people",
      "id": "42",
      "attributes": {
        "name": "John",
        "age": 80,
        "gender": "male"
      }
    }
  ]
}
```

Fonctionnalités pragmatiques

- Sparse Fieldsets : on choisi les champs que l'on souhaite se voir retourner
- Pagination Links : on récupère un certain nombre de résultats et les liens de pagination
- Error Objects : format standardisé de gestion des erreurs

Implémentations de JSON:API

Existe pour beaucoup de langages

- https://jsonapi.org/implementations/#client-libraries-java

GraphQL

- Langage pour requêter des API
- On n'obtient que ce dont on a besoin
  - Idéal pour les traitements de données personnelles (RGPD)
  - À coupler avec des mécanisme d'autorisation côté métier

Demo : https://developer.github.com/v4/explorer/

Pour aller plus loin

- https://introspected.rest/