

Applications distribuées

Protocole HTTP et architectures REST

Module RCPI01

Sebastien.Kramm@univ-rouen.fr

IUT R&T Rouen

2018-2019

- REST : un style de conception pour les application distribuées :
 - client-serveur, basé sur HTTP,
 - indépendant de tout langage,
 - plus souple, moins de couplage client-serveur que d'autres approches (SOAP/XML).

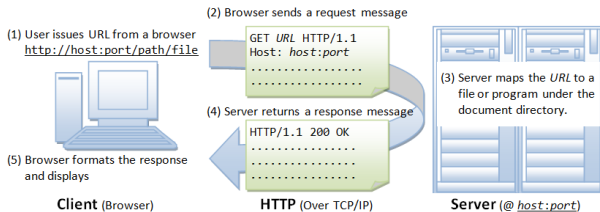
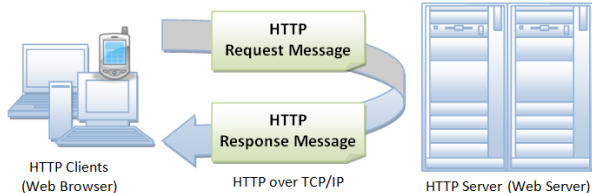
1 Protocole HTTP

2 Architecture REST

- Format des données échangées
- REST en pratique

Contenu

- Protocole textuel (ASCII), s'appuie sur TCP port 80 par défaut (443 pour HTTPS)



Requête du client au serveur

```
1 GET /docs/index.html HTTP/1.1
2 Host: www.nowhere123.com
3 Accept: image/gif, image/jpeg, */*
4 Accept-Language: en-us
5 Accept-Encoding: gzip, deflate
6 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
7
```

Analyse :

- ligne 1 : Requête, composée d'un **verbe** (*GET*), d'une URL, et du protocole utilisé (*HTTP/1.1*). Champs séparés par une espace (ASCII 0x20).
- lignes suivante (*request headers*) : paires de chaînes "*nom : valeur*". Les valeurs peuvent être multiples en les séparant par des virgules.
- certains champs sont obligatoires, d'autres optionnels.
- dernière ligne vide : obligatoire.

Description complète : [RFC 7230](#)

Réponse du serveur

```
1 HTTP/1.1 200 OK
2 Date: Sun, 18 Oct 2009 08:56:53 GMT
3 Server: Apache/2.2.14 (Win32)
4 Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
5 ETag: "10000000565a5-2c-3e94b66c2e680"
6 Accept-Ranges: bytes
7 Content-Length: 44
8 Connection: close
9 Content-Type: text/html
10 X-Pad: avoid browser bug
11
12 <html><body><h1>It works!</h1></body></html>
```

Analyse :

- ligne 1 : Réponse serveur, avec un **code de statut**
- lignes suivante (*response headers*) : paires de chaînes "*nom : valeur*".
- la plupart des champs sont standardisés mais on peut en avoir des non-standards, qui seront en général ignorés ([voir page WP](#)).
- ligne vide, puis contenu de la page.
- la nature des données est indiquée par le **type MIME** : `content-type`

- internet \neq web !
- internet : interconnexion de **réseaux autonomes** (1970-1980)
- 1989 : Tim Berners-Lee invente HTTP, avec les adresses Web et le langage HTML
⇒ World Wide Web (www)
- HTTP/0.9 : 1991
- HTTP/1.0 : 1996
- HTTP/1.1 : 1997, standard de l'IETF
- HTTP/2.0 : 2015
 - objectif principal : amélioration des performances
 - compatibilité ascendante avec 1.1
 - supporté par tous les grands acteurs, mais encore peu utilisé

Type MIME

- Le logiciel client a besoin de savoir ce que lui envoie le serveur : HTML, image (png, jpeg, ...) son, vidéo, fichier archive, etc.
- Cette information est indiquée par le serveur avec le champ `Content-Type` qui indique le **type MIME** des données.
- MIME : *Multipurpose Internet Mail Extensions*, crée initialement pour l'email.
- Constitué d'un **type** et d'un **sous type**
Exemples : text/plain, text/css, text/html, audio/mpeg, image/png, video/H264
- Pour les échanges de données entre applications, on utilise : application/json, application/javascript, application/xml
- On peut y ajouter un **encodage** :
`Content-Type: text/plain; charset=utf-8`
- Plus : [page WP](#)

Le serveur répond à la requete avec un code de statut à 3 chiffres :

- 1xx (Informational) : Request received, server is continuing the process.
- 2xx (Success) : The request was successfully received, understood, accepted and serviced.
- 3xx (Redirection) : Further action must be taken in order to complete the request.
- 4xx (Client Error) : The request contains bad syntax or cannot be understood.
- 5xx (Server Error) : The server failed to fulfill an apparently valid request.

- La norme prévoit différentes requêtes, que le serveur doit interpréter différemment
- Les plus courantes :
 - GET : demande au serveur de renvoyer la ressource. Sans effet sur le serveur.
 - POST : utilisée pour transmettre des données en vue d'un traitement à une ressource (le plus souvent depuis un formulaire HTML)
 - PUT : pour remplacer ou ajouter une ressource sur le serveur.
- Il en existe d'autres :
HEAD, OPTIONS, CONNECT, TRACE, PATCH, DELETE, ...
- Un navigateur Web ne fait que du GET, sauf pour les formulaires si la méthode POST est précisée dans le code :

```
<form method="post" action="page.php">  
</form>
```

1 Protocole HTTP

2 Architecture REST

- Format des données échangées
- REST en pratique

REST : REpresentational State Transfer

- Roy Fielding, 2000.
- Pas un standard, mais un style d'architecture, basé sur le succès du WWW.
- Basé sur le concept de **ressource** : entité conceptuelle représentant ce que le client attend :
 - Une commande en cours identifiée par son n° ,
 - le dernier billet du blog,
 - le billet/article daté du 1/02/2016,
 - ...
- Utilisation de HTTP pour la couche transport.
- Données renvoyées en XML ou JSON.
- Conçu comme une alternative plus légère à SOAP, qui utilise XML pour envoyer une spécification complète de la requête.

- Inspiration : *www (World Wide Web)*

Exemple : un humain cherche un billet d'avion :

- se connecte sur un site de voyageur ;
- tape sa requête ;
- clique sur un des liens qui lui est proposé.

Mais : difficilement utilisable par une machine : pages web conçues pour les humains, difficile d'extraire l'info pertinente.

- API REST = « **page web pour les machines** »

- un seul point d'entrée, qui reste fixe
- permet de naviguer dans les ressources
- données reçues du serveur dans un format normalisé

1 Protocole HTTP

2 Architecture REST

- Format des données échangées
- REST en pratique

- Aujourd'hui, deux types de formats basés texte sont dominants
 - XML : *Extensible Markup Language*
 - Hérite de SGML
 - Normalisé en 1998 (1.0) et 2004 (1.1)
 - JSON : *JavaScript Object Notation*
 - Hérité de Javascript
 - Normalisé en 2014, mais utilisé depuis bien plus longtemps
- Permettent de transférer de l'information **structurée**
- Comparaisons :
 - JSON est plus "jeune", plus souple et moins "verbeux" que XML
 - XML est très utilisé pour de la description d'IHM
 - XML est extensible, pas JSON (pas besoin)
 - XML est plus puissant : permet la transformation et l'extraction des données directement via le langage d'interrogation XPATH
 - JSON : format d'échange **dominant** dans les échanges entre applications aujourd'hui

XML vs. JSON : des personnes

- JSON :

```
{ "employees": [                                ( un tableau )
  { "firstName": "John", "lastName": "Doe"},
  { "firstName": "Anna", "lastName": "Smith"},
  { "firstName": "Peter", "lastName": "Jones"}
]}
```

- XML :

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</
      lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</
      lastName>
  </employee>
</employees>
```


XML vs. JSON : un livre

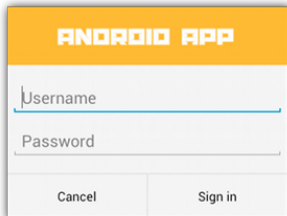
- JSON (140 caractères) :

```
{
  "id": 123,                un entier
  "title": "Object Thinking", une chaîne
  "author": "David West",
  "published": {           un "sous-objet"
    "by": "Microsoft Press",
    "year": 2004
  }
}
```

- XML (167 caractères) :

```
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

- XML est utilisé par Android pour la description de l'interface utilisateur



res/layout/dialog_signin.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/
  android:orientation="vertical"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">
  <ImageView
    android:src="@drawable/header_logo"
    android:layout_width="match_parent"
    android:layout_height="64dp"
    android:scaleType="center"
    android:background="#FFFBB3"
    android:contentDescription="@string/app_name" />
  <EditText
    android:id="@+id/username"
    android:inputType="textEmailAddress"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:layout_marginLeft="4dp"
```

JSON : format orienté "objet"

- Un objet JSON est délimité par { et } et contient des paires clé-valeur, séparées par ",",
- Les clés sont des chaînes.
- Les valeurs peuvent être :

- Une chaîne, un nombre ou un booléen :

```
"clé1": "aaa", "clé2": 1345, "clé3": true
```

- Un sous-objet :

```
"clé1": { "aaa": "bbb", "ccc": 123 }
```

- Un tableau d'objets :

```
"aaa": [ { ... }, { ... }, { ... } ]
```

- Inconvénients :
 - pas de commentaires possibles ;
 - typage faible (pas de type date, pas de distinction nombre entier & flottant) ;
 - pas optimal (qté information / nombre d'octets)
 - pas de possibilité d'extension.
- Pour aller plus loin :

https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

1 Protocole HTTP

2 Architecture REST

- Format des données échangées
- REST en pratique

REST : exemple

- Exemple : soit une entreprise de pièces détachées "parts-depot" qui met à disposition de ses clients son catalogue
- Elle propose l'URI suivante :
`http://www.parts-depot.com/parts`
- Le client reçoit via http le document XML suivant :

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part id="00345" xlink:href="http://www.parts-depot.com/parts
    /00345"/>
  <Part id="00346" xlink:href="http://www.parts-depot.com/parts
    /00346"/>
  <Part id="00347" xlink:href="http://www.parts-depot.com/parts
    /00347"/>
</p:Parts>
```

REST : exemple

- Si le client veut des détails sur une pièce, alors il envoie la requête suivante (HTTP GET) :


```
http://www.parts-depot.com/parts/00345
```

- Le serveur renverra le document :

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com"
        xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part-ID>00345</Part-ID>
  <Name>Widget-A</Name>
  <Description>This part is used within the frap assembly</
    Description>
  <Specification xlink:href="http://www.parts-depot.com/parts
    /00345/specification"/>
  <UnitCost currency="USD">0.10</UnitCost>
  <Quantity>10</Quantity>
</p:Part>
```

- Placer une commande : requête POST sur l'URL

```
http://www.parts-depot.com/place-order
```

avec comme  utile un fichier XML contenant la description de la commande.

Principes de base d'une architecture REST :

- Utiliser dans les URI des **noms**, pas des verbes

Exemple : Au lieu de

```
http://www.parts-depot.com/parts/getPart?id=00345
```

on préfère

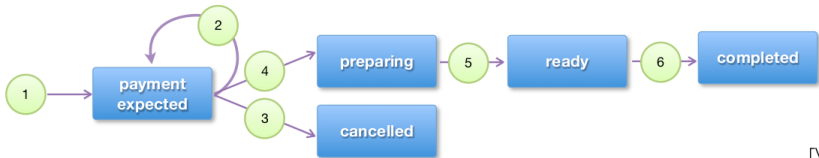
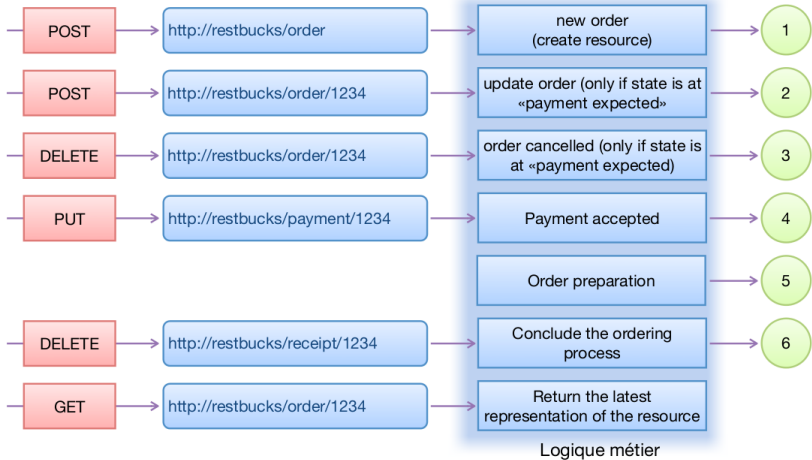
```
http://www.parts-depot.com/parts/00345
```

- Un seul point d'entrée (*endpoint*) qui permet de naviguer vers les données
- 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.

- REST utilise HTTP comme un protocole de transport, mais surtout comme **protocole applicatif** :
→ L'ensemble des méthodes du protocole peut-être utilisées (à la différence du Web, qui n'utilise que GET et POST)

Verbe (méthode)	Sémantique
GET	récupère une ressource
PUT	met à jour une ressource
DELETE	supprime une ressource
POST	créé une nouvelle ressource
...	

Réf : [HTTP sur Wikipedia](#)



REST : Exemple IRL 1 : yuml.me

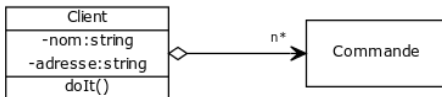
- L'URI suivante :

[http://yuml.me/diagram/nofunky/class/\[Client|-nom:string;-adresse:string|doIt\(\)\]<>-n*>\[Commande\].json](http://yuml.me/diagram/nofunky/class/[Client|-nom:string;-adresse:string|doIt()]<>-n*>[Commande].json)
renvoie (format JSON) :

```
{"customisations":"nofunky","type":"class","dsl":["Client|-nom:string;-adresse:string|doIt()]<>-n*>[Commande]","digest":"cc59342a","file_only":false,"mime":"image/png","extension":"png"}
```

- et la même avec "png" au lieu de "json" à la fin :

[http://yuml.me/diagram/nofunky/class/\[Client|-nom:string;-adresse:string|doIt\(\)\]<>-n*>\[Commande\].png](http://yuml.me/diagram/nofunky/class/[Client|-nom:string;-adresse:string|doIt()]<>-n*>[Commande].png)
renvoie :




Web API Endpoint Reference

Our Web API endpoints give external applications access to Spotify catalog and user data.

Web API Base URL: <https://api.spotify.com>

[User Guide](#) | [Tutorial](#) | [Code Examples](#)

Search:

Method	Endpoint	Usage	Returns
GET	<code>/v1/albums/{id}</code>	Get an album	album
GET	<code>/v1/albums?ids={ids}</code>	Get several albums	albums
GET	<code>/v1/albums/{id}/tracks</code>	Get an album's tracks	tracks*
GET	<code>/v1/artists/{id}</code>	Get an artist	

album object (simplified)

Key	Value Type	Value Description
<code>album_type</code>	string	The type of the album: one of "album", "single", or "compilation".
<code>available_markets</code>	array of strings	The markets in which the album is available: ISO 3166-1 alpha-2 countries

API Spotify : exemple d'accès

- On souhaite accéder à l'album identifié par l'identifiant 4aawyAB9vmqN3uQ7FjRGTy
- On utilisera l'entrée :
<https://api.spotify.com/v1/albums/4aawyAB9vmqN3uQ7FjRGTy>
- Réponse :

```
{
  "album_type": "album",
  "artists": [ {
    "external_urls": {
      "spotify": "https://open.spotify.com/artist/0Tn0YISbd1XYRBk9myaseg"
    },
    "href": "https://api.spotify.com/v1/artists/0Tn0YISbd1XYRBk9myaseg",
    "id": "0Tn0YISbd1XYRBk9myaseg",
    "name": "Pitbull",
    "type": "artist",
    "uri": "spotify:artist:0Tn0YISbd1XYRBk9myaseg"
  } ],
  "available_markets": [ "AD", "AT", "BE", "BG", "CH", "LT", "LU", "LV", "MC", "MT", "MX", "NL", "NO", "PL", "PT", "SE", "SI", "SK", "US" ],
  "copyrights": [ {
    "text": "(P) 2012 RCA Records, a division of Sony Music Entertainment",
    "type": "P"
  } ]
}
```

- Dans les champs renvoyés on trouve l'url du "artwork" (couverture) :
<https://i.scdn.co/image/b6be520fd1dc9feb84100be40f63de4f80694918>

- Beaucoup d'API fournissent un service payant : il faut s'**authentifier**.
Pour toute l'API, ou pour certains points d'entrée seulement
- Technique la plus courante : **OAuth**, standard ouvert de **délégation d'autorisation**
Histoire : v1 : 2007, v2 : 2012
- Utilisé par Google, Facebook, Microsoft, Twitter, ... pour accorder le droit à des applications tierces d'accéder à une API.
- Procédure complexe (multiples étapes, identifiants et clés).
- Des bibliothèques logicielles dans différents langages peuvent proposer des *wrappers*, limitant la difficulté.