EPSI I4 Web Services / **David Gayerie** - 2013-2014

INTRODUCTION AUX WEB SERVICES

TENTATIVE DE DÉFINITION

Une première tentative : **celle du W3C (http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice)**

Une seconde tentative : **celle de Wikipedia (http://en.wikipedia.org/wiki/Web_service)**

Web = HTTP

service = client/serveur

Un Web service s'appuie sur le protocole HTTP selon les principes d'une architecture Client/Serveur pour échanger de l'information.

HTTP (HYPERTEXT TRANSFER PROTOCOL)

HTTP est un protocole applicatif qui est le fondement de l'échange de données pour le World Wide Web.

Comme tous les protocoles applicatifs de l'Internet, HTTP est orienté texte.

HTTP version 1.1 est défini dans la **RFC 2616 (http://tools.ietf.org/html/rfc2616)** datant de 1999.

Exemple de l'envoi d'une requête

Un requête HTTP GET

```
GET /Programmes/Formation-informatique HTTP/1.1
Host: www.epsi.fr
```

La réponse du serveur

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 420

<html>
...
</html>
```

Structure d'une requête HTTP

```
[méthode] [URI (chemin de ressource)] HTTP/1.1
[Nom de l'en-tête]: [Valeur de l'en-tête]
[ligne blanche]
[entité envoyée]
```

En HTTP 1.1, seul l'en-tête *Host* est obligatoire.

Structure d'une réponse HTTP

```
HTTP/1.1 [code statut] [message]
[Nom de l'en-tête]: [Valeur de l'en-tête]
[ligne blanche]
[entité reçue]
```

Les **méthodes HTTP (http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html)** :

- OPTIONS
- GET
- HEAD
- PUT
- PATCH
- DELETE
- POST
- TRACE
- CONNECT

OPTIONS
Obtenir les options de communication (par exemple : les méthodes autorisées pour l'URI)

GET
Demander au serveur une entité identifiée par l'URI

HEAD
Comme un GET sauf que la réponse ne contient *jamais* l'entité. Utile pour connaître uniquement les en-têtes des réponses.

PUT
Stocker ou mettre à jour l'entité envoyée à l'adresse de l'URI.

***PATCH** (http://tools.ietf.org/html/rfc5789)*
Appliquer des modifications partielles à l'entité identifiée par l'URI.

DELETE
Détruire l'entité identifiée par l'URI.

POST
Annoter des ressources existantes,
Poster un message dans un système de centralisation d'articles,
Fournir un bloc de données (formulaire) à un processus de traitement,
Étendre une base de données par une opération d'ajout.

Les **en-têtes (http://en.wikipedia.org/wiki/List_of_HTTP_header_fields)** HTTP :

A noter : le nom des en-têtes est insensible à la casse.

Pour les requêtes et les réponses contenant une entité, on connaît la taille de l'entité et le type de contenu grâce aux en-têtes.

Pour la taille, on peut utiliser l'en-tête *Content-Length* ou *Transfer-Encoding*

Content-Length
Indique la taille de l'entité en octets

Transfer-Encoding
Avec la valeur *chuncked* indique que l'entité est transmise par morceau. Très utile quand le serveur ne peut pas connaître la taille de la réponse avant de l'avoir émise.

Pour identifier le contenu, on utilise principalement l'en-tête *Content-Type*.

Content-Type
Indique le type **MIME (http://en.wikipedia.org/wiki/MIME)** de l'entité : text/plain, text/html, application/json, image/jpeg, ...

Content-Language
Indique la langue ou une liste de langues naturelles nécessaire pour comprendre l'entité

Les familles de **codes de statut (http://en.wikipedia.org/wiki/List_of_HTTP_status_codes)** de la réponse :

1XX
Information

2XX
Succès (à connaître : 200, 201, 204)

3XX
Redirection (à connaître : 301, 303, 304, 307)

4XX
Erreur du client (à connaître : 400, 404, 405, 409, 410, 412, 415)

5XX
Erreur du serveur (à connaître : 500, 501, 502, 503)

Cas d'utilisation :

- Création d'une entité dont on connait l'URI
- Création d'une entité dont on ne connait pas l'URI
- Suppression d'une entité
- Requête conditionnelle (dernière modification)
- Requête conditionnelle (Etag)
- Redirections
- Traitement asynchrone

Création d'une entité dont on connait l'URI

La requête

```
PUT /individu/David+Gayerie HTTP/1.1
Host: exemple.fr
Content-Type: application/json; charset=utf-8
Content-Length: 91

{'name' : 'Gayerie',
 'firstname' : 'David',
 'email' : 'david.gayerie.epsi@mailoo.org'}
```

La réponse du serveur en cas de succès

```
HTTP/1.1 201 Created
```

Création d'une entité dont on ne connait pas l'URI

La requête

```
POST /individu/ HTTP/1.1
Host: exemple.fr
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 65

name=Gayerie&firstname=David&email=david.gayerie.epsi@mailoo.org
```

La réponse du serveur en cas de succès

```
HTTP/1.1 201 Created
Location: http://exemple.fr/individu/000001
```

Le serveur fournit l'URI d'accès dans la réponse grâce à l'en-tête *Location*.

Suppression d'une entité

La requête

```
DELETE /individu/000001 HTTP/1.1
Host: exemple.fr
```

La réponse du serveur en cas de succès

```
HTTP/1.1 204 No content
```

Requête conditionnelle (dernière modification)

La requête

```
DELETE /individu/000001 HTTP/1.1
Host: exemple.fr
If-unmodified-since: Thu, 1 Jan 2010 00:00:00 GMT
```

La réponse si l'entité a été modifiée depuis le 01/01/2010

```
HTTP/1.1 412 Precondition failed
```

Requête conditionnelle (Etag)

Récupération de l'entité et de son Etag

```
GET /individu/000001 HTTP/1.1
```

La réponse du serveur

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 91
ETag: "686897696a7c876b7e"

{'name' : 'Gayerie',
 'firstname' : 'David',
 'email' : 'david.gayerie.epsi@mailoo.org'}
```

Etag est un en-tête optionnel : le serveur peut ne pas le retourner.

La requête conditionnelle

```
DELETE /individu/000001 HTTP/1.1
Host: exemple.fr
If-match: "686897696a7c876b7e"
```

L'en-tête *If-match* retransmet l'ETag fourni par le serveur.

La réponse du serveur si l'entité a changé d'ETag depuis

```
HTTP/1.1 412 Precondition failed
```

Les redirections :

La requête

```
GET /individu/000001 HTTP/1.1
Host: exemple.fr
```

La réponse du serveur si l'entité n'est plus accessible à cette URI

```
HTTP/1.1 301 Moved permanently
Location: http://exemple.fr/individu/2013/000001
```

L'en-tête *Location* dans la réponse fournit l'URI de redirection.

S'il n'existe plus d'URI pour atteindre l'entité, le serveur devrait retourner le code statut 410 (gone) ou 404 (not found).

Les codes de redirection :

301 Moved Permanently
Le serveur connaît l'entité mais ne peut plus la servir à l'URI donnée. Ce code signale au client qu'il doit maintenant utiliser une nouvelle URI.

302 Found
Code "historique" de redirection. Il est ambigu (utiliser plutôt 303 ou 307).

303 See Other
La requête a été exécutée mais son résultat est disponible avec un GET sur une autre URI.

307 Temporary redirect
La requête n'a pas pu être exécutée. Le client doit émettre à nouveau sa requête vers l'URI temporaire fournie par le serveur.

Traitement asynchrone :

La requête

```
DELETE /individu/000001 HTTP/1.1
Host: exemple.fr
```

La réponse du serveur si la requête est traitée en asynchrone

```
HTTP/1.1 202 Accepted
Location: http://exemple.fr/jobs/1948321
```

Les méthodes HTTP peuvent être classées selon deux critères : la sûreté et l'idempotence.

Sûreté (safety)
Une méthode est sûre si elle n'entraîne *aucune* modification d'état sur le serveur. Une méthode sûre peut être utilisée entre 0 et N fois d'affilée sans que cela affecte le serveur.

Les méthodes GET, HEAD et OPTIONS doivent être sûres.

Idempotence (idempotent)
Une méthode est idempotente si l'effet obtenu est le même qu'elle soit exécutée 1 ou N fois.

Les méthodes GET, HEAD, OPTIONS, PUT et DELETE doivent être idempotentes.

Classement des méthodes HTTP

	Sûre	**Idempotente**
OPTIONS	oui	oui
GET	oui	oui
HEAD	oui	oui
PUT	non	oui
PATCH	non	non
DELETE	non	oui
POST	non	non

Pour aller plus loin:

HTTP Made Really Easy
http://www.jmarshall.com/easy/http/

La RFC 2616 de l'IETF
http://tools.ietf.org/html/rfc2616

La RFC 5789 définissant la méthode PATCH
http://tools.ietf.org/html/rfc5789

CLIENT/SERVEUR

L'architecture **Client/Serveur (http://en.wikipedia.org/wiki/Client_server)** est une architecture logique centralisée pour le développement d'applications distribuées.

Un programme serveur se met en attente de requêtes de programmes clients vers lesquels il retourne une réponse.

LES GRANDES FAMILLES DE WEB SERVICES

- Web services RESTful
- WS-* (SOAP, WSDL)
- *-RPC (XML-RPC, JSON-RPC)

Java EE supporte le développement de Web services pour ces trois familles.