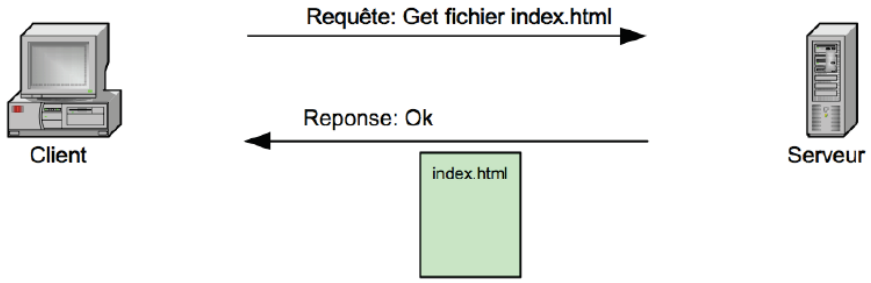
Les services Web REST

Protôcole HTTP

HyperText Transfert Protocol

- HTTP permet d'accéder aux fichiers situés sur le réseau Internet. Il est notamment utilisé pour le *World Wide Web*
- HTTP se place au dessus de TCP et fonctionne selon un principe de requête/réponse
  - Le client transmet une requête comportant des informations sur le document demandé
  - Le serveur renvoie le document si disponible ou, le cas échéant, un message d'erreur
- HTTP est un protocole synchrone initialement **sans connexion** et chaque couple requête/réponse est de ce fait indépendant



« Adresses » HTTP

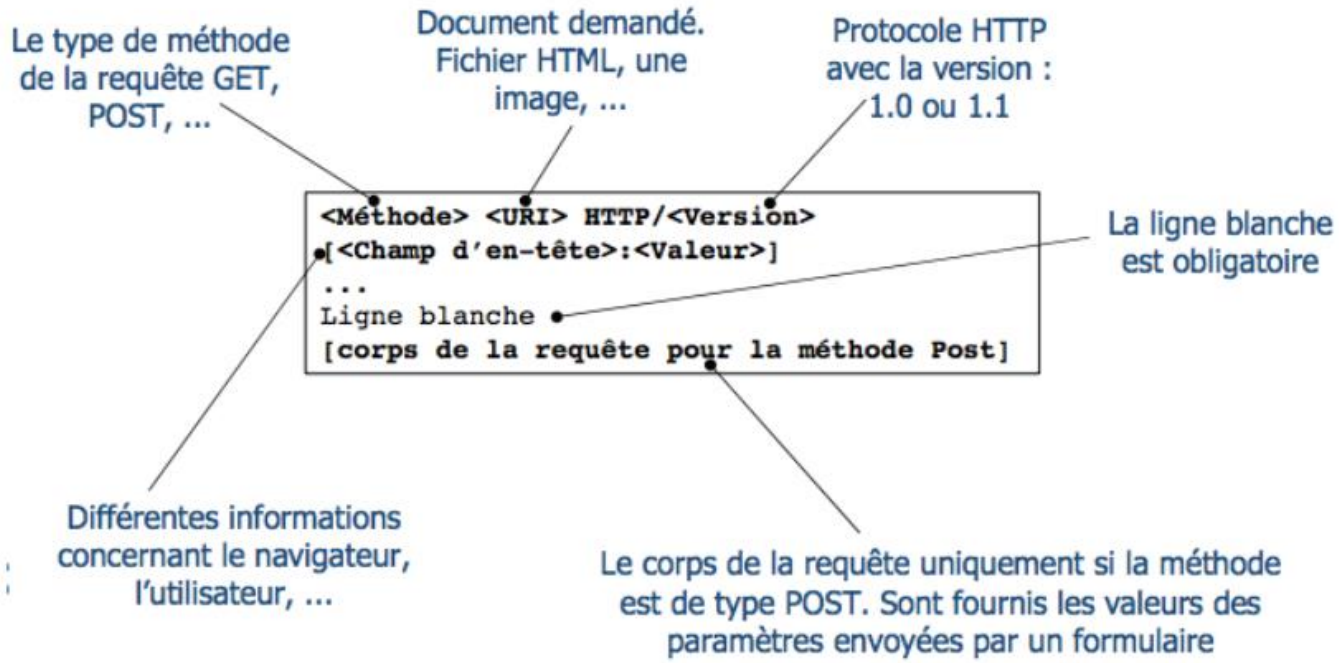
- *Uniform Resource Identifier URI*
  - chaîne de caractères structurée permettant d'**identifier de manière unique une ressource dans un espace de nom défini**
  - Cette ressource peut être désignée soit par un URN, soit par un URL
  - URN et URL sont des sous-ensembles de URI
- *Uniform Resource Name ou URN*
  - permet d'**identifier une ressource par son nom même lorsque** celle ci n'est plus disponible
- *Uniform Resource Locator ou URL*
  - permet de **localiser une ressource**
  - Dans le cas du protocole HTTP, une URL permet de localiser une page HTML, un fichier texte, un script cgi, une image…

Format d'un URI

```
HTTP://<host>:<port>/<path>?<query>#<fragment>
```

Champ	Description
host	Permet de spécifier le FQDN (ou adresse IP) du serveur possédant la ressource à accéder
port	Permet de spécifier le numéro de port à utiliser pour atteindre le serveur possédant la ressource. Si sa valeur est 80 (port par défaut du protocole HTTP), il n'est pas nécessaire de spécifier le numéro de port dans l'URL.
path	Permet de spécifier l'emplacement du fichier sur le serveur. Ce champ est en général constitué d'une suite de répertoires séparés par des '/' puis du nom du fichier à accéder.
query	Permet de passer un, ou plusieurs, paramètre(s) à un script PHP, Perl etc .
fragment	Permet d'indiquer une « position » (ancre, *fragment*) dans une page

Requête HTTP



Exemple - Requête HTTP

- Accepte tous les types de document en retour
- Préfère les documents en français
- Utilise un navigateur (*browser) compatible Mozilla 4.0 sur un* système *Windows NT 5.1 (Windows XP)*
- Signale au serveur qu'il faut garder la connexion TCP ouverte à l'issue de la requête (car il a d'autres requêtes à transmettre)

```
GET /index.html HTTP/1.1
Host: www.example.com
Accept: */*
Accept-Language: fr
User-Agent: Mozilla/4.0 (MSIE 6.0; windows NT 5.1)
Connection: Keep-Alive
```

Type de méthodes

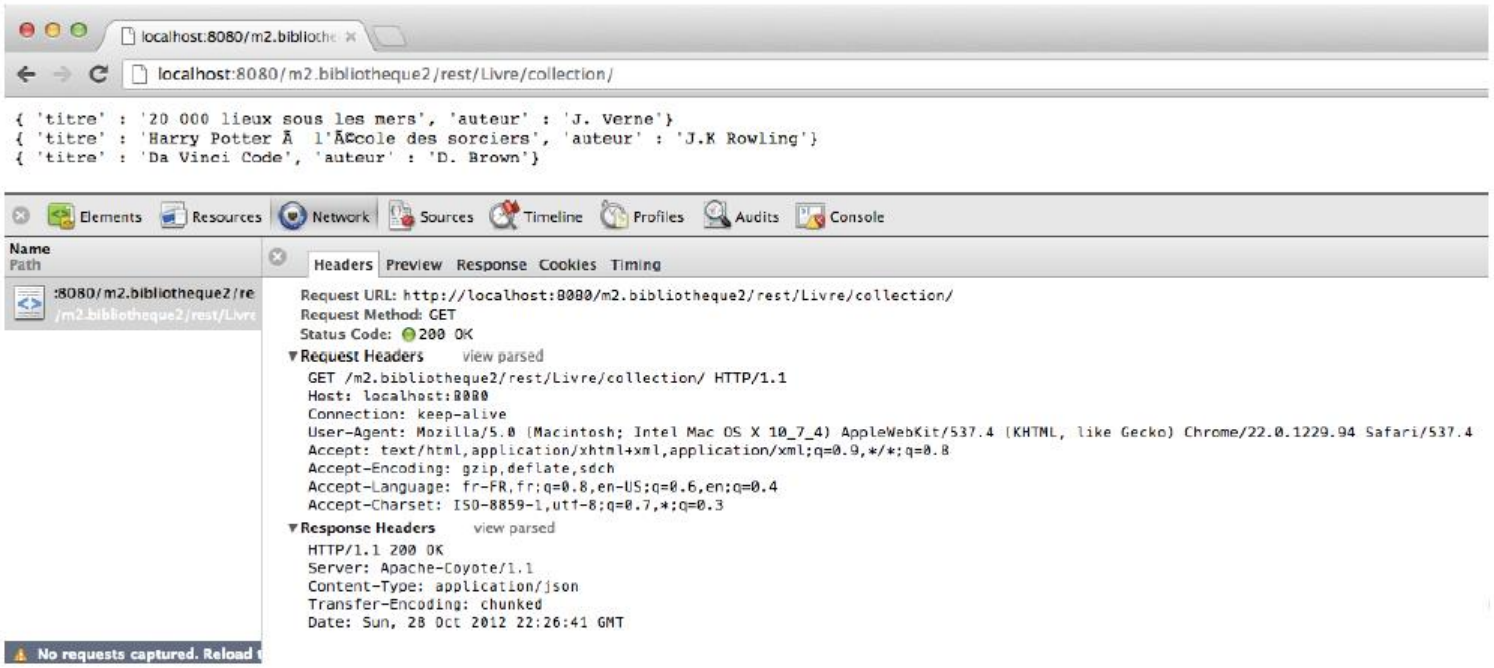
- Requêtes de type GET
  - Pour extraire des informations
    - Document, graphique
  - Intègre les données de formatages de l'URI (chaîne d'interrogation)
    - www.toto.com/hello?key1=titi&key2=tata&…
- Requêtes de type POST
  - Pour poster des informations secrètes (au sens pas visibles dans l'en-tête), des données graphiques…
  - Transmises dans le corps de la requête

Type de méthodes

Méthodes	1.0	1.1	Description
Get			Permet de demander un document
Post			Permet de transmettre des données (d'un formulaire par exemple) à l'URL spécifiée dans la requête. L'URL désigne en général un script Perl, PHP
Head			Permet de ne recevoir que les lignes d'en-tête de la réponse, sans le corps du document
Options			Permet au client de connaître les options du serveur utilisables pour obtenir une ressource
Put			Permet de transmettre au serveur un document à enregistrer à l'URL spécifée dans la requête
Delete			Permet d'effacer la ressource spécifiée
Trace			Permet de signaler au serveur qu'il doit renvoyer la requête telle qu'il la reçue
Connect			Permet de se connecter à un *proxy* ayant la possibilité d'effectuer du *tunneling*

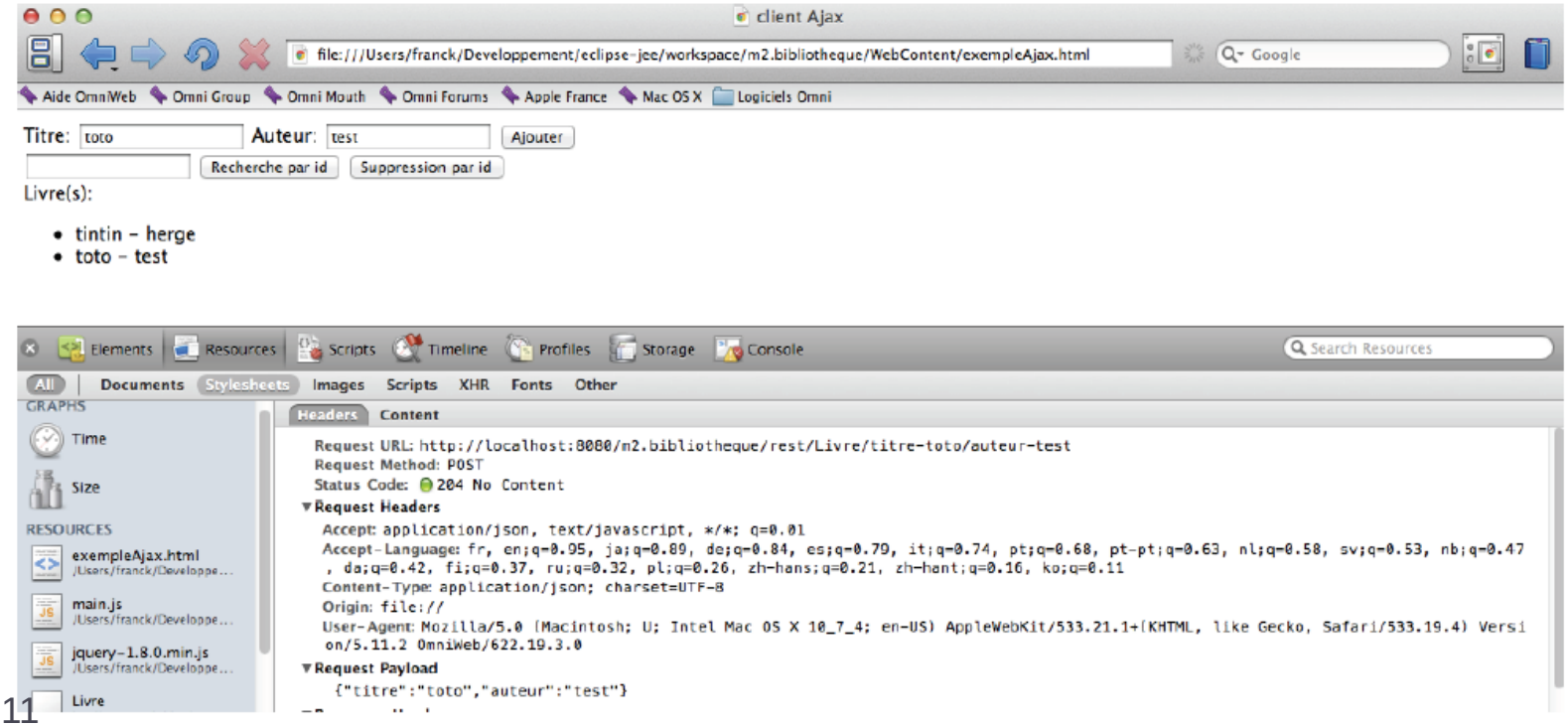
Principales méthodes – GET

- La méthode **GET est une requête d'information sur une ressource**
  - L'information fournie en réponse est sous forme
    - D'un ensemble d'headers et d'une représentation
  - Le client n'envoie jamais de représentation avec la requête (corps de la requête vide)



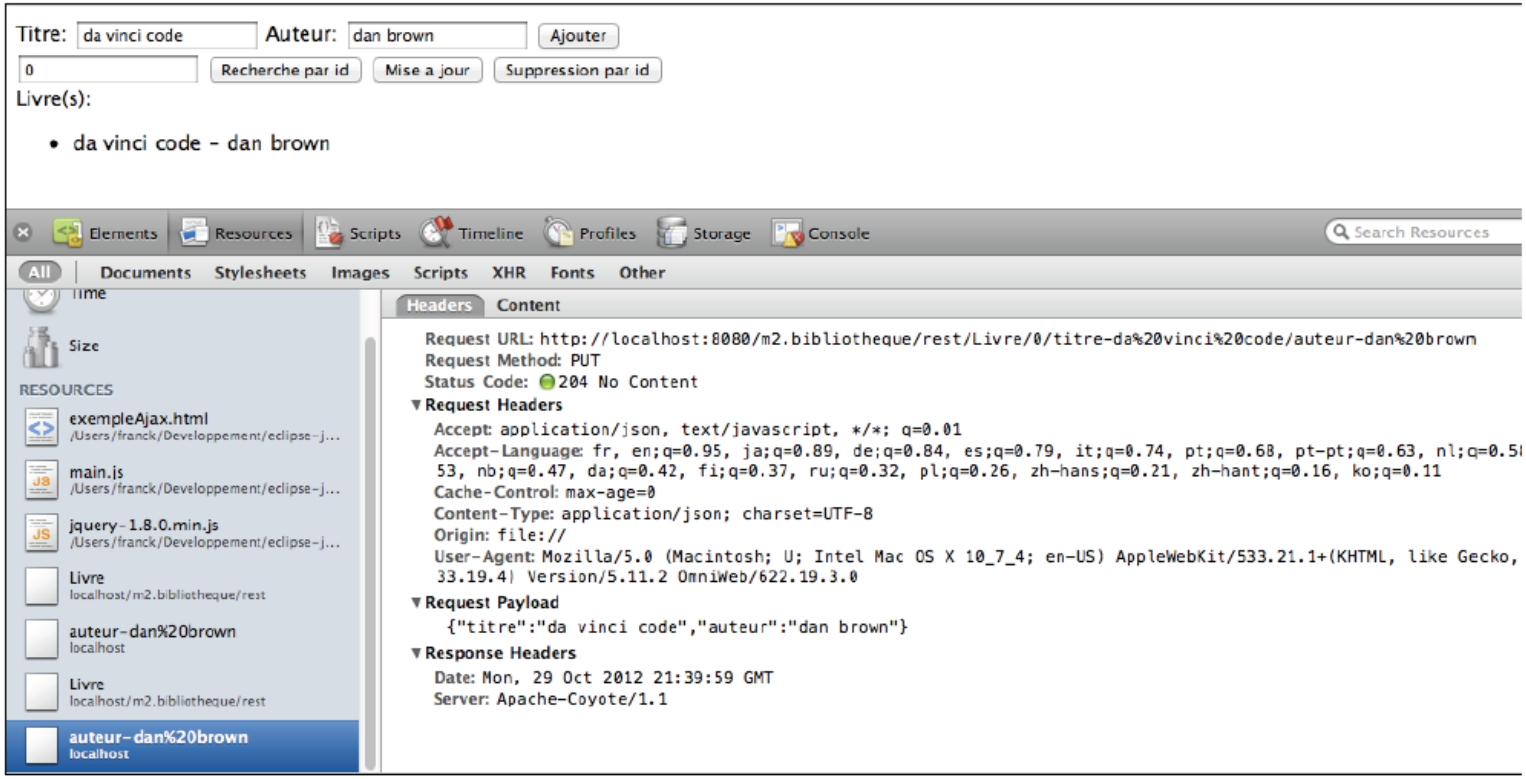
Principales méthodes – POST

- La méthode **POST permet de créer / ajouter une nouvelle ressource**
  - Tous les paramètres à transmettre au services peuvent être dans le header
  - Aucune donnée attendue en réponse (mais ceci est toutefois possible)



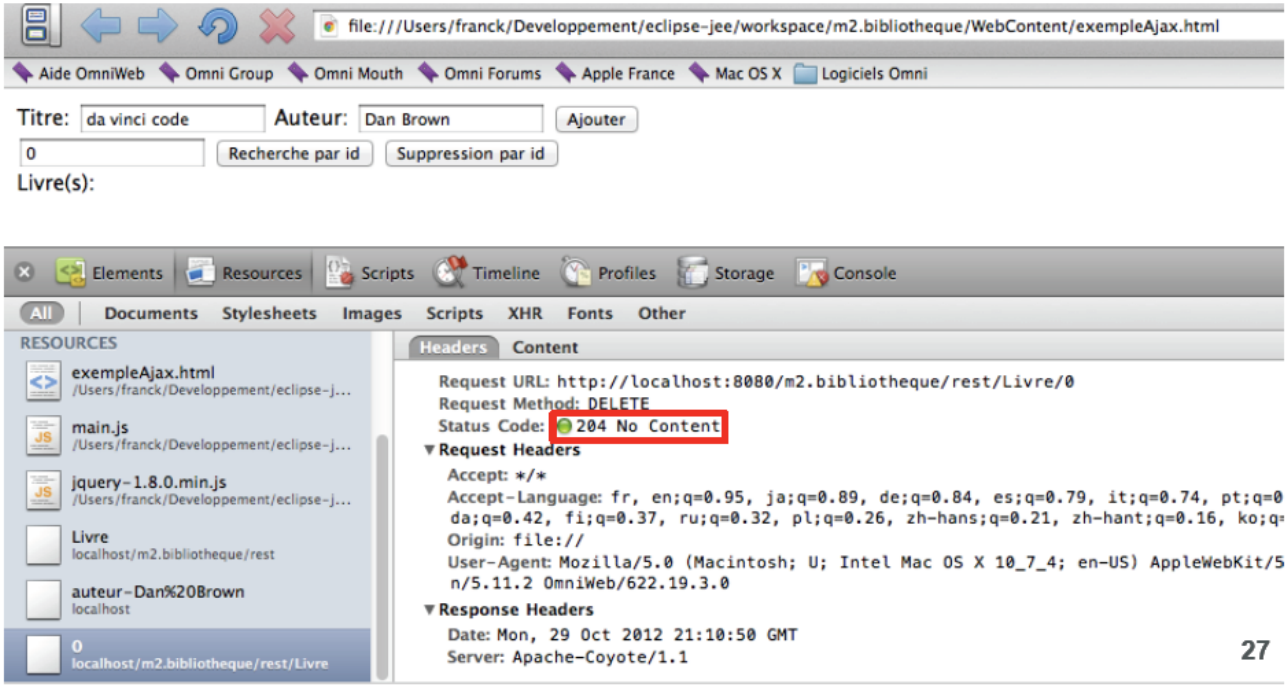
Principales méthodes – PUT

- La méthode **PUT permet de mettre à jour une ressource**
  - Inclut l'ajout d'une sous-ressource
  - Tous les paramètres à transmettre au services peuvent être dans le header
  - Aucune donnée attendue en réponse (mais ceci est toutefois possible)



Principales méthodes – DELETE

- La méthode **DELETE permet de supprimer une ressource**
  - Tous les paramètres à transmettre au services peuvent être dans le header
  - Aucune donnée attendue en réponse (mais ceci est toutefois possible)



Principales méthodes

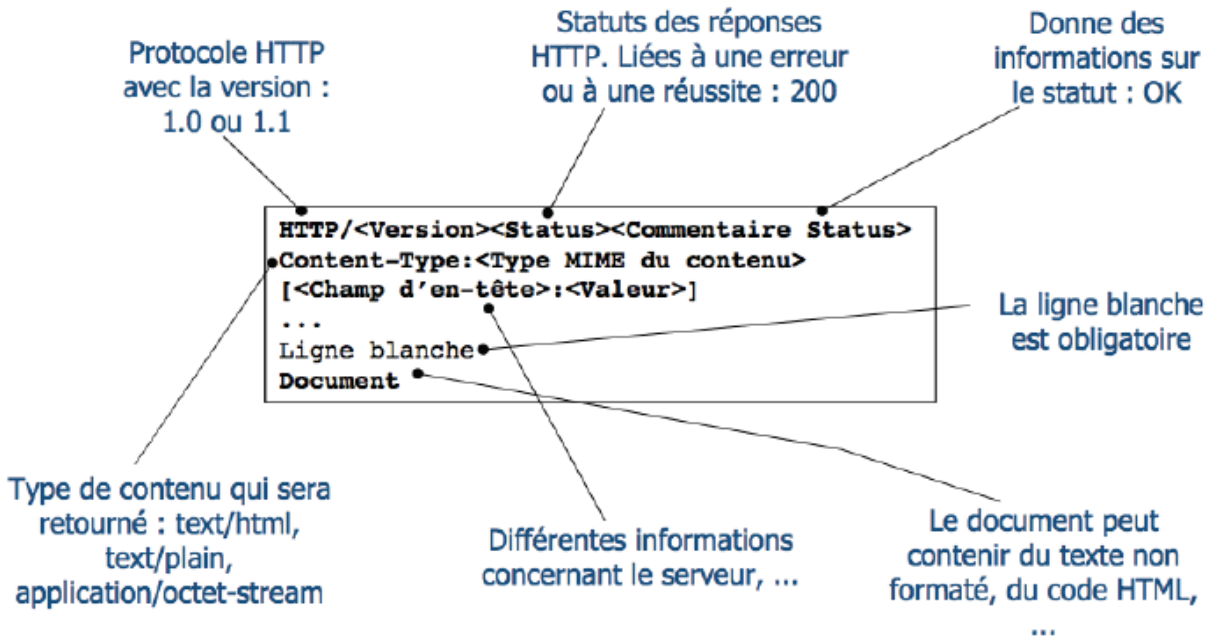
- Deux caractéristiques : la sureté et l'idempotence
- Une méthode **sûre** ne doit jamais modifier l'état de la ressource
  - Cas des méthodes GET et HEAD
  - POST, DELETE et PUT ne sont pas sûres
- Une méthode est **idempotente** dès lors qu'elle peut être répétée un nombre quelconque de fois, l'ensemble des ressources reste toujours dans le même état après l'application de la méthode
  - Autrement dit, le résultat d'une opération idempotente reste toujours le même dans un contexte donné avec des paramètres donnés

Principales méthodes

- La méthode GET est sure et idempotente
  - Un client qui fait une requête de type GET sur une ressource ne requiert aucun changement de cette ressource
  - Faire un nombre quelconque de GET ou aucun devrait avoir le même effet
- Les méthodes PUT et DELETE sont idempotentes
  - Faire plusieurs requêtes PUT (ou DELETE) sur une ressource doit avoir le même effet que d'en faire une seule
  - Problème connu :
    - PUT qui modifie l'état de la ressource en faisant un incrément de 5 sur une valeur
    - Une telle spécification n'est pas possible pour être idempotent
- La méthode POST n'est ni sure ni idempotente
  - C'est elle qui sert de « boite à outils » à divers frameworks (messages personnalisés, etc.)

Réponse HTTP

- La structure d'une réponse HHTP envoyé par le serveur



Réponse HTTP : Exemple

```
HTTP/1.1 200 OK
Date: Mon, 15 Dec 2003 23:48:34 GMT
Server: Apache/1.3.27 (Darwin) PHP/4.3.2 mod_perl/1.26
DAV/1.0.3
Cache-Control: max-age=60
Expires: Mon, 15 Dec 2003 23:49:34 GMT
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "26206-5b0-3af1f126"
Accept-Ranges: bytes
Content-Length: 1456
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
...
```

En-têtes des réponses HTTP

Champ	Description
Accept	Types MIME que le client accepte
Accept-encoding	Méthodes de compression supportées par le client
Accept-language	Langues préférées par le client (pondérées)
Cookie	Données de *cookie* mémorisées par le client
Host	Hôte virtuel demandé
If-modified-since	Ne retourne le document que si modifié depuis la date indiquée
If-none-match	Ne retourne le document que sil a changé
Referer	URL de la page à partir de laquelle le document est demandé
User-agent	Nom et version du logiciel client

En-têtes des réponses HTTP

Champ	Description
Allowed	Méthodes HTTP autorisées pour cette URI (comme POST)
Content-encoding	Méthode de compression des données qui suivent
Content-language	Langue dans laquelle le document retourné est écrit
Date	Date et heure UTC courante
Expires	Date à laquelle le document expire
Last-modified	Date de dernière modification du document
Location	Adresse du document lors d'une redirection
Etag	Numéro de version du document
Pragma	Données annexes pour le navigateur (par exemple, no.cache)
Server	Nom et version du logiciel serveur
Set-cookie	Permet au serveur d'écrire un *cookie* sur le disque du client

Codes des réponses HTTP

Code	Nom	Description
Information 1xx		
100	Continue	Utiliser dans le cas où la requête possède un corps.
101	Switching protocol	Réponse à une requête
Succès 2xx		
200	OK	Le document a été trouvé et son contenu suit
201	Created	Le document a été créé en réponse à un PUT
202	Accepted	Requête acceptée, mais traitement non terminé
204	No response	Le serveur n'a aucune information à renvoyer
206	Partial content	Une partie du document suit
Redirection 3xx		
301	Moved	Le document a changé d'adresse de façon permanente
302	Found	Le document a changé d'adresse temporairement
304	Not modified	Le document demandé n'a pas été modifié
Erreurs du client 4xx		
400	Bad request	La syntaxe de la requête est incorrecte
401	Unauthorized	Le client n'a pas les privilèges d'accès au document
403	Forbidden	L'accès au document est interdit
404	Not found	Le document demandé n'a pu être trouvé
405	Method not allowed	La méthode de la requête n'est pas autorisée
Erreurs du serveur 5xx		
500	Internal error	Une erreur inattendue est survenue au niveau du serveur
501	Not implemented	La méthode utilisée n'est pas implémentée
502	Bad gateway	Erreur de serveur distant lors d'une requête *proxy*

Approche orientée ressource

Services Web REST

- Exploités pour les Architectures Orientées Données (DOA)
- REST n'est pas un standard, il n'existe pas de spécification W3C.
- REST est un style d'architecture basé sur un mode de compréhension du Web
- REST s'appuie sur des standards du Web
  - Protocole HTTP
  - URIs
  - Formats de fichiers
  - Sécurisation via SSL

Approche Orientée Ressources

- REST est l'acronyme de **REpresentational State Transfert**
- Ses principes ont été définis dans la thèse de Roy FIELDING en 2000
  - L'un des principaux auteurs des spécifications de HTTP
  - Membre fondateur de la fondation Apache
  - Développeur du serveur web Apache
- **REST est un style d'architecture** inspiré de l'architecture du web

Pourquoi REST?

- Les Services Web REST sont utilisés pour développer des architectures orientées ressources
- Différentes dénominations disponibles dans la littérature
  - Architectures Orientées Données (DOA)
  - Architectures Orientées Ressources (ROA)
- Les applications qui respectent les architectures orientées ressources sont respectivement nommées **RESTful**

Caractéristiques de REST

- Les services Web REST sont sans états (Stateless)
  - Le serveur n'a jamais besoin de connaître l'état du client et réciproquement
  - Le client maintient l'état de l'application de son point de vue
  - Le serveur fait de même en maintenant l'état de ses ressources

➔Ces états ne sont jamais partagés !!!

  - Tout changement d'état a lieu à la suite d'un échange de messages entre le client et le serveur ➔transfert de représentations
  - Vu que le serveur et le client ne connaissent pas leurs états respectifs
    - Chaque requête envoyée vers le serveur doit contenir toutes les informations nécessaires à son traitement
    - Le serveur ne conserve aucune information sur les clients

➔Minimisation des ressources, pas de session ni d'état

Caractéristiques de REST

- Les services Web REST fournissent une interface uniforme basée sur les méthodes HTTP
  - GET, POST, PUT et DELETE
- Les architectures orientées REST sont construites à partir de ressources qui sont uniquement identifiées par des URIs

Trois concepts du REST

- **Ressource (Identifiant)**
  - Identifiée par une URI
  - Exemple : http://localhost:8080/libraryrestwebservice/books
- **Méthode (Verbe)**
  - Permet de manipuler l'identifiant ou la ressource
  - Méthodes HTTP : GET, POST, PUT et DELETE
- **Représentation**
  - Elle donne une vue d'une ressource
  - On parle souvent de la vue de l'état d'une ressource
  - C'est l'information transférée entre le client et le serveur
  - Exemple : XML, JSON

… et 4 propriétés

- Les représentations doivent être adressables
- Les services doivent être sans état
- Les services / ressources doivent être connectés
- Les services respectent une interface uniforme (Uniform Interface ou UI)

Ressources et URI (1/3)

- Une ressource est quelque chose qui est identifiable dans un système
  - Personne, Agenda, Document, Ensemble, Carte

Mauvaise URI

- http://cours-rest.fr/api?method=findStudent&userid=nLegoff&sessionid=06102015

Bonne URI

- http://cours-rest.fr/students/nicolas-legoff

D'un point de vue architecture

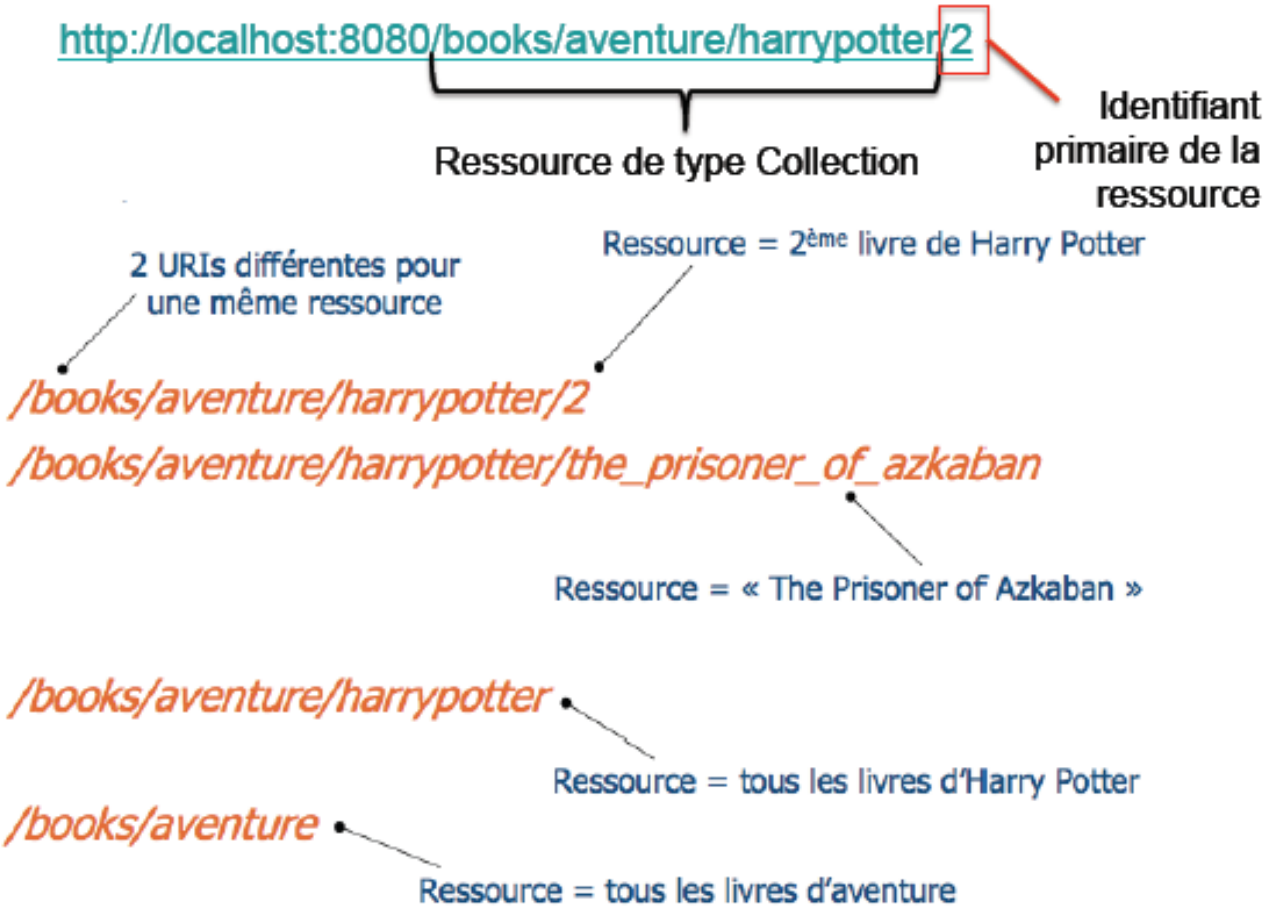
- Première solution = choix d'implémentation
  - ici l'appel de méthode sur un service distant
  - HTTP simplement utilisé comme transporteur de message uniquement
- Seconde solution
  - Moins l'impression d'invoquer une opération distante
  - URL qui traduit un concept : un étudiant
  - ET aucune action

Ressources et URI (2/3)

- Une ressource est tout élement identifiable. Une URI identifie une ressource de manière unique
  - Attention, une ressource peut avoir plusieurs URIs
  - La représentation de la ressource n'est pas liée à l'URI. Elle peut évoluer avec le temps et le client
  - *Une URI doit **avoir une structure***

```
http://www.example.com/software/releases/1.0.3.tar.gz
http://www.example.com/software/releases/latest.tar.gz
http://www.example.com/weblog/2006/10/24/0
http://www.example.com/map/roads/USA/AR/Little_Rock
http://www.example.com/wiki/Jellyfish
http://www.example.com/search/Jellyfish
http://www.example.com/nextprime/1024
http://www.example.com/next-5-primes/1024
```

Ressources et URI (3/3)



Ressources et URI (3/3)

http://localhost:8080/books/aventure/harrypotter/2

/books/aventure/harrypotter/2

/books/aventure/harrypotter/the_prisoner_of_azkaban

/books/aventure/harrypotter

/books/aventure

Méthodes CRUD

- Une ressource peut subir quatre opérations de base désignées par le terme CRUD pour
  - *Create – Créer*
  - *Retrieve – Lire*
  - *Update – Mettre à jour*
  - *Delete – Supprimer*
- REST s'appuie sur le protocole HTTP pour exprimer directement ces 4 opérations de base via les méthodes de HTTP
  - *Create par la méthode POST*
  - *Retrieve par la méthode GET*
  - *Update par la méthode PUT*
  - *Delete par la méthode DELETE*
- Des possibilités supplémentaires peuvent être exprimées via d'autres méthodes HTTP (HEAD, OPTIONS)

Représentation (1/2)

- Fournir les données suivant une représentation pour
  - le client (GET)
  - le serveur (PUT et POST)
- Les données retournées sont sous différents formats
  - XML, JSON, HTML, CSV, ...
- Le format d'entrée (POST) et le format de sortie (GET) d'un service Web d'une ressource peuvent être différents

Représentation (2/2)

- Exemples : formats JSON et XML

GET https://www.googleapis.com/urlshortener/v1/url?shortUrl=http://goo.gl/fbsS

```
{
 'kind': "urlshortener#url",
 'id': "http://goo.gl/fbsS",
 'longUrl': "http://www.google.com/",
 'status': "OK"
}
```

Représentation des données en JSON

GET http://localhost:8080/librarycontentrestwebservice/contentbooks/string

```
<?xml version="1.0"?>
<details>
    Ce livre est une introduction sur la vie
</details>
```

Représentation des données en XML

Quatre propriétés

- L'architecture orientée ressource repose sur 4 propriétés
  - Propriété1:
    - Les représentation doivent être adressables
  - Propriété2:
    - Les services doivent être sans état
  - Propriété3:
    - Les services / ressources doivent être connectés
  - Propriété4:
    - Les services respectent une interface uniforme (Uniform Interface ou UI)

Représentation adressable

- Un service web est adressable dès lors qu'il expose certaines de ses données sous forme de ressources visibles
  - cf. annotation @Path des classes java visibles en Jersey
- Un URI ne doit jamais représenter plus d'une ressource (sinon plus d'universalité)

Exemple : Une ressource accessible en anglais et en français

- Solution fréquemment utilisé **un URI → une représentation**
  - www.mylibrary/2012/books/en ← une représentation en anglais
  - www.mylibrary/2012/books/fr ← une représentation en français
- Autre solution
  - www.mylibrary/2012/books/ ← un seul URI
  - Les deux représentations existent toujours (2 méthodes annotées GET avec des @Produces différents)
  - Au client de choisir avec **le Accept-Language du header de la requête**

Service sans état

- Quand un client fait une requête HTTP
  - Toutes les informations nécessaires à l'exécution de la requête par le serveur sont envoyées au serveur
  - Le serveur ne réutilise jamais d'information provenant de requêtes précédentes
- En pratique, on transfert les informations via les adresses (URIs)

Toute requête HTTP doit pouvoir s'exécuter de manière totalement isolée

Propriété – Service sans état

- Une application Web doit scaler / passer à l'échelle
  - Des clusters de serveurs avec gestion de l'équilibrage de charges, des proxys et des points d'entrées forment des topologies permettant aux requêtes de circuler entre les serveurs ← **ceci afin de réduire les temps de réponse pour le client**
  - Ceci implique de pouvoir transférer des requêtes indépendantes et complètes i.e. des requêtes autoportantes) → **l'état ne doit pas être propre au serveur**
  - Un requête autoportante ne doit donc stocker/utiliser aucune information sur le serveur qui lui soit propre
- Un service Web REST inclut dans le header et le corps de la requête HTTP tout ce qui est nécessaire au fonctionnement du service appelé
  - Paramètres, contexte, données nécessaires au serveur
- Être sans état simplifie la conception et l'implémentation des services côté serveur car l'absence d'état supprime le besoin de synchroniser des données de la session avec une application externe

Service sans état - Bonnes Pratiques

- **Côté Serveur**
  - Génère des réponses qui incluent des liens sur d'autres ressources pour permettre aux applications clientes de naviguer vers ces ressources
  - Génère des réponses qui indiquent si elles sont « *cacheables* » *ou non pour améliorer* les performances
- **Côté Client**
  - Utilise la valeur de Cache-Control du header de la réponse pour déterminer si la réponse peut être copiée localement ou pas
  - Le client lit aussi la valeur Last-Modified du header de la réponse et renvoit la valeur si la valeur du If-Modified-Since header a changé (*appelé GET conditionnel)*
  - Envoie des requêtes autoportantes

Service sans état

- Un service sans état n'impacte qu'un type d'état
- Pour rappel, il existe 2 types d'états dans un service REST
  - **L'état de la ressource est l'information relative à la ressource**
  - **L'état de l'application est l'information relative à chaque client**
- L'état de l'application peut apparaître quand on ne s'y attend pas
  - De nombreux sites web créent des clés uniques pour chaque utilisateur enregistré
  - Cette clé est envoyée avec chaque requête (limitation du nombre de requêtes par jour / droit d'accès)

Ressources connectées

- Le serveur peut guider le client d'une ressource à une autre en lui envoyant des liens vers d'autres resssources dans les réponses aux requêtes
  - Hypermedia as the engine of application state (Fielding's PHD thesis)
- C'est le cas du « web humain » où des liens vers d'autres pages sont présents dans quasiment toutes les pages webs
- Au contraire le « web programmable » est peu connecté

Références

- https://www.json.org
- Mickeal Baron, *SOA-Serice Web REST, Comprendre le syle d'architecture REST, ISAE-ENSMA, 2011*
- Philippe Genoud, *JSON*, Université Grenoble Alpes, 2016
- Xavier Blanc, *Web Services*, Université de bordeaux,
- Walid Gaaloul, *REpresentational State Transfert Vers les architectures orientées ressources*, Telecom SudParis, 2018