COMPRENDRE L'ARCHITECTURE DES WEB SERVICES REST

Amosse EDOUARD, Doctorant

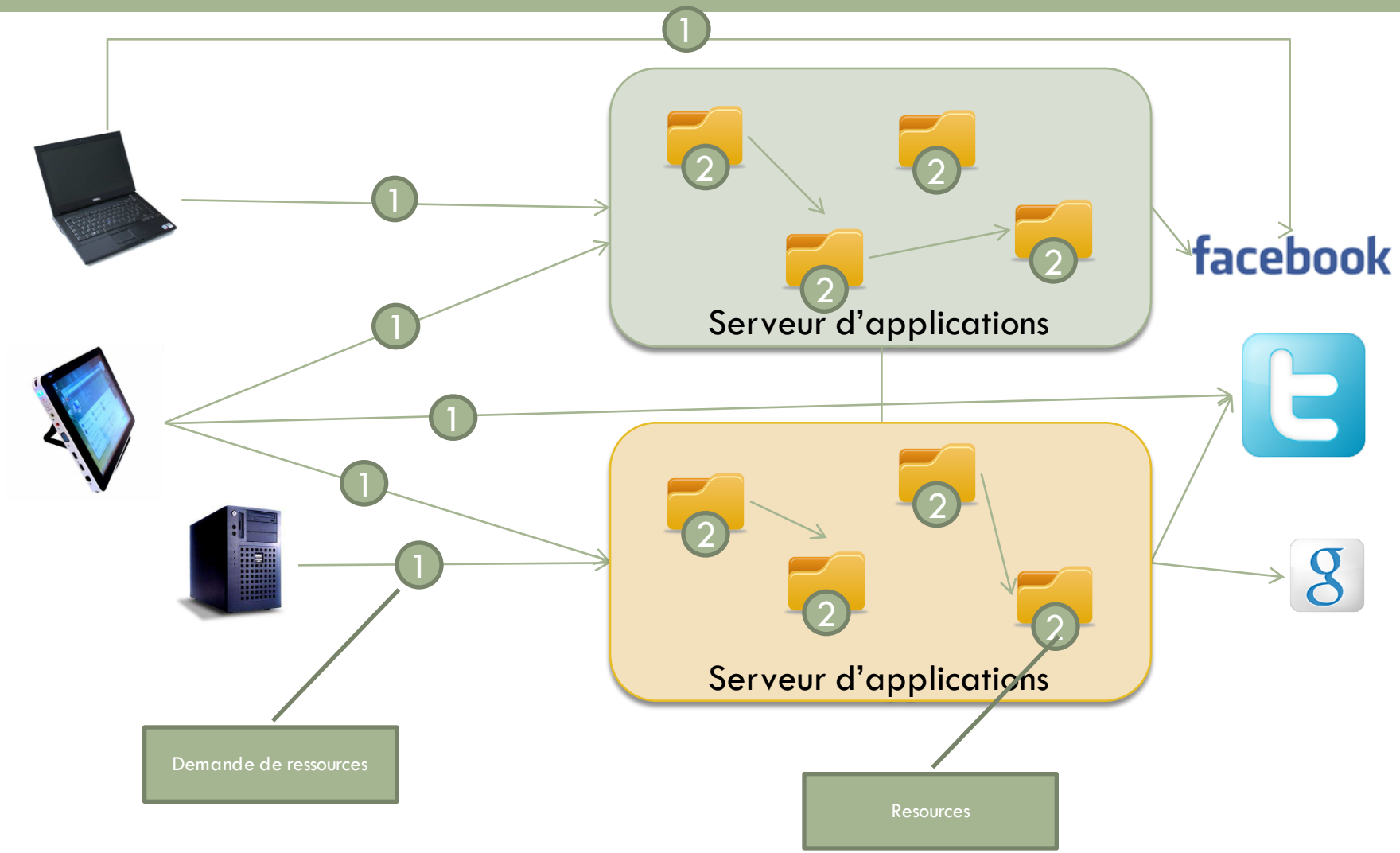
Evaluation

- Présence, Participation➔ 20%
- Mini Projet ➔ 40%
- Examen Théorique ➔ 40%

Objectifs du cours

- SOAP
  - Initiation au protocole et les Web services SOAP
  - Comprendre les enveloppes SOAP
- REST
  - Comprendre l'architecture des services web REST
  - Créer et exposer des services REST
  - Consommer des services REST

Utilisation du web aujourd'hui (1)



Web Services (Définition)

- Services informatiques de la famille des technologies web permettant la communication entre des applications **hétérogènes** dans des environnements distribués *(Wikipédia)*.
- Ils ont été proposé à la base comme solution d'intégrations de différents logiciels développés par des entreprises (ERP, SCM, CRM) leur permettant de communiquer entre eux.
- Basés sur XML (déscription et échange) et utilisant en général les protocoles du web comme canal de communication;

Types de Services Webs

- Deux principaux types
  - SOAP
  - REST

Web Services SOAP

- **S**imple **O**bject **A**ccess **P**rotocol

- Protocole d'échanges d'informations dans un environnement distribué basé sur XML
  - Interopérabilité entre applications d'une même entreprise (Intranet)
  - Interopérabilité inter entreprises entre applications et services web

- Similaire au protocole RCP,

Web Services SOAP

- SOAP peut être utilisé de concert avec plusieurs autres protocoles : HTTP, SMTP, POP

→ HTTP est le plus utilisé

Web Services SOAP

	RMI	DCOM	CORBA	SOAP
Défini par	SUN	Microsoft	OMG	W3C
Plate-forme	Multi	Win32	Multi	Multi
Langage de Développement	Java	C++, VB, VJ	Multi	Multi
Langage de définition	Java	ODL	IDL	WSDL
Transport	TCP, HTTP, IIOP	IP/IPX	GIOP, IIOP	HTTP, HTTPR, SMTP
Transaction	Non	Oui	Oui	Oui
Sécurité	SSL, JAAS	?	SSL	SSL

Web Services SOAP

- SOAP est principalement composé de trois parties:
  - Les enveloppes SOAP (ou Message)
  - Les règles d'encodages
  - La représentation RPC

Messages SOAP

- L'Enveloppe SOAP → Obligatoire
  - Une en-tête (Header) → Optionnel
  - Le corps (Body) → Obligatoire

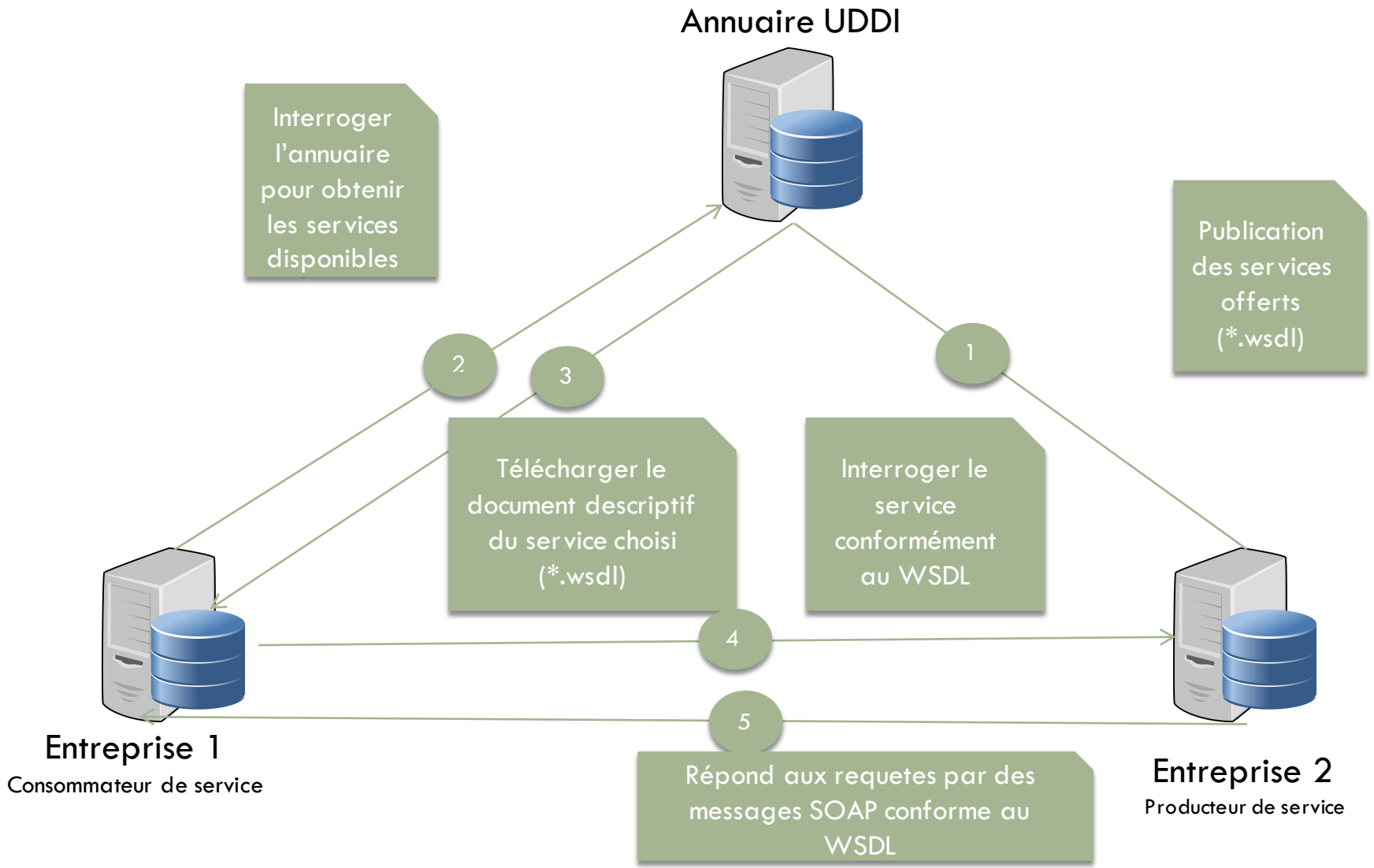
Messages SOAP

- Les messages SOAP sont utilisés pour envoyer (requête) et recevoir (réponse) des informations d'un consommateur vers un producteur
- Un message SOAP peut être transmis à plusieurs récepteurs intermédiaires avant d'être reçu par le récepteur final (➔ chaîne de responsabilité)
- Le format SOAP peut contenir des messages spécifiques correspondant à des erreurs identifiées par le récepteur
- Un message SOAP est véhiculé en utilisant un protocole de transport (HTTP, SMTP, …)

SOAP : WSDL

- Web Service Description Language
- Fichier au format XML
- Décrit les actions exposées par le web service

SOAP – Exemples d'utilisation



SOAP : Enveloppe

- Constitue la racine d'un message SOAP
- Identifié par la balise <namespace:Envelop>
- La balise doit être **obligatoirement** associé à un espace de noms [spec W3C]
- SOAP définit deux espaces de noms
  - Enveloppe SOAP : http://schemas.xmlsoap.org/soap/envelope/
  - Serialization SOAP: http://schemas.xmlsoap.org/soap/encoding/
- Requête et Réponse ont la même structure

SOAP : En-tête

- Balise optionnelle identifié par <namespace:Header>
- Quand il est présent, il doit être avant le Body
- Utilisé pour transmettre des informations supplémentaires entre le consommateur et le fournisseur du service
- Usages possibles
  - Informations d'authentification
  - Contexte d'une transaction
  - Transiter des informations intermédiaires

SOAP : Corps

- Identifié par la balise <namespace:Body>
- Contient la réponse à l'appel d'une action du service
  - Une erreur <namespace:Fault>
  - Réponse de l'action
- L'encodage est des informations est précisé par les bindings du WSDL

SOAP : Requête

- Appeler les opérations d'un web service SOAP

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:sayHelloToTheWorld
xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/"/>
    </S:Body>
</S:Envelope>
```

Appel à la méthode sayHelloToTheWorld sans paramètre

SOAP : Réponse

- Réponse du service à l'appel de la méthode

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:sayHelloToTheWorldResponse
xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/">
            <return>Hello World</return>
        </ns2:sayHelloToTheWorldResponse>
    </S:Body>
</S:Envelope>
```

Réponse du web service à l'appel de la méthode sayHelloToTheWorld

SOAP : Requête

- Appeler les opérations d'un web service SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:sayHelloTo
xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/">
            <name>Miage NTDP</name>
        </ns2:sayHelloTo>
    </S:Body>
</S:Envelope>
```

Appel à la méthode sayHelloTo du service avec une valeur en paramètre

SOAP : Réponse

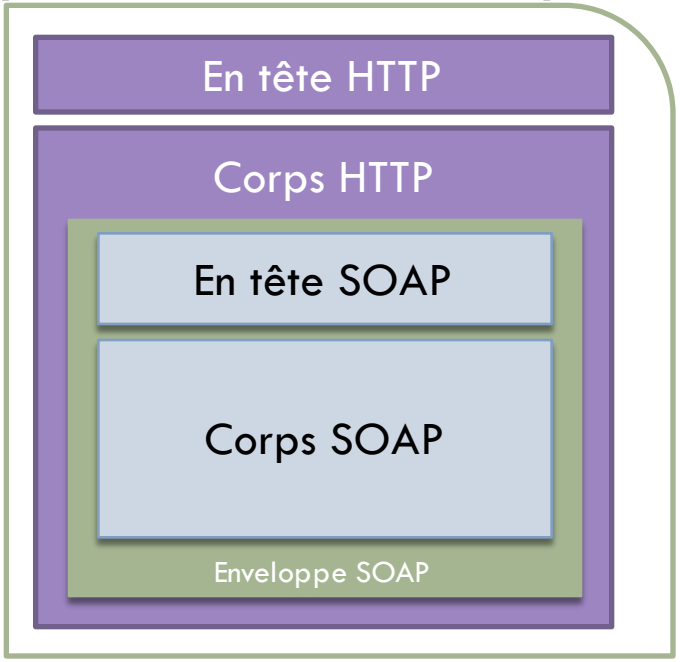
- Réponse du service à l'appel de la méthode

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:sayHelloToResponse
xmlns:ns2="http://soap.bibliotheque.android.mbds.fds.edu.ht/">
            <return>Hello Miage NTDP !</return>
        </ns2:sayHelloToResponse>
    </S:Body>
</S:Envelope>
```

Réponse du web service à l'appel de la méthode sayHelloTo

SOAP : Transport HTTP

- Structure d'une requête HTTP
  - En-tête (http header)
  - Corps (http body)
- Les messages SOAP sont encapsulés dans le corp de la requête HTTP



Services Web RESTFul

Web Service REST

Définition

- Acronyme de **RE**presentational **S**tate **T**ransfert défini dans la thèse de Roy Fielding en 2000.
- REST n'est pas un protocole ou un format, contrairement à SOAP, HTTP ou RCP, mais un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP
- Il n'est pas dépendant uniquement du web et peut utiliser d'autre protocoles que HTTP

Web Service REST

Ce qu'il est :

- Un système d'architecture
- Une approche pour construire une application

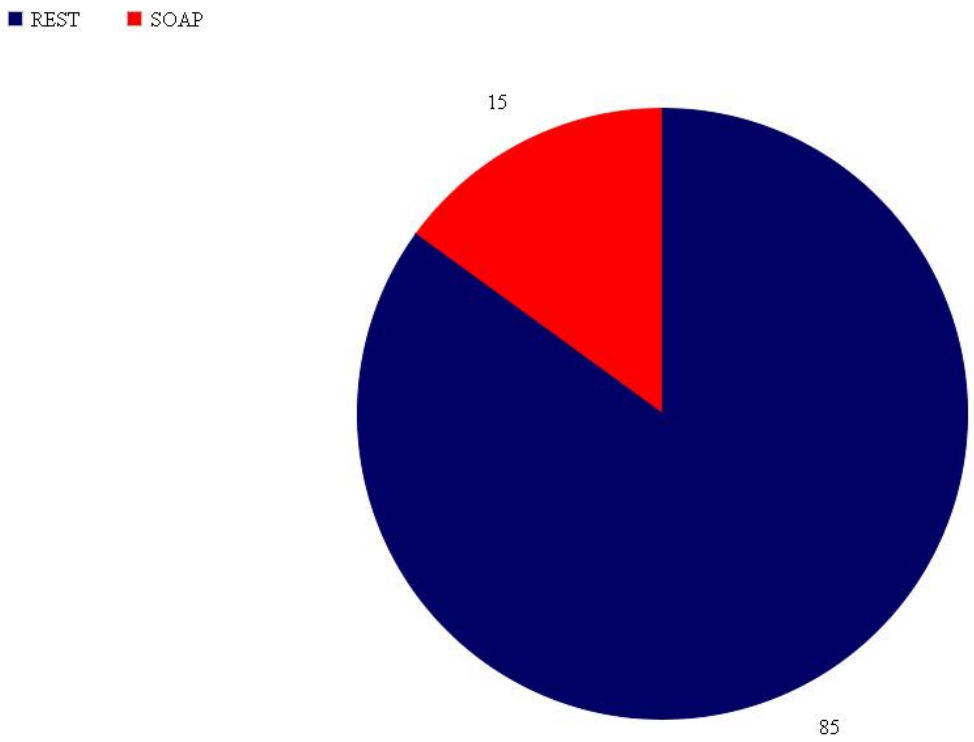
Ce qu'il n'est pas

- Un protocole
- Un format
- Un standard

Qui font du REST?

REST → Statistics

Statistique d'utilisation des services web REST et SOAP chez AMAZON



REST: Une Architecture Simple

Basée sur 7 propriétés principales

- **Performance** : Interaction simple entre les composants
- **Evolutivité** : Supporte une large variété de composants
- **Simplicité** : Entre les interfaces
- **Modification** : Peut être modifié sans impacter les clients
- **Visibilité** : Communication claire entre les composantes
- **Confiance** : Reprise sur panne

REST → Caractéristiques

- Séparation Client/Serveur
- Sans Etats (Stateless)
  - Chaque requête envoyée au serveur doit contenir toutes les informations relatives à son état et est traitée indépendamment de toutes autres requêtes
  - Aucune gestion de session ou d'état des ressources par le serveur
- Possibilité de mise en Cache
- Orienté Ressources
- Messages auto-descriptifs
- Hypermedia : Aucune autre action n'est supportée outre celles qui sont décrites

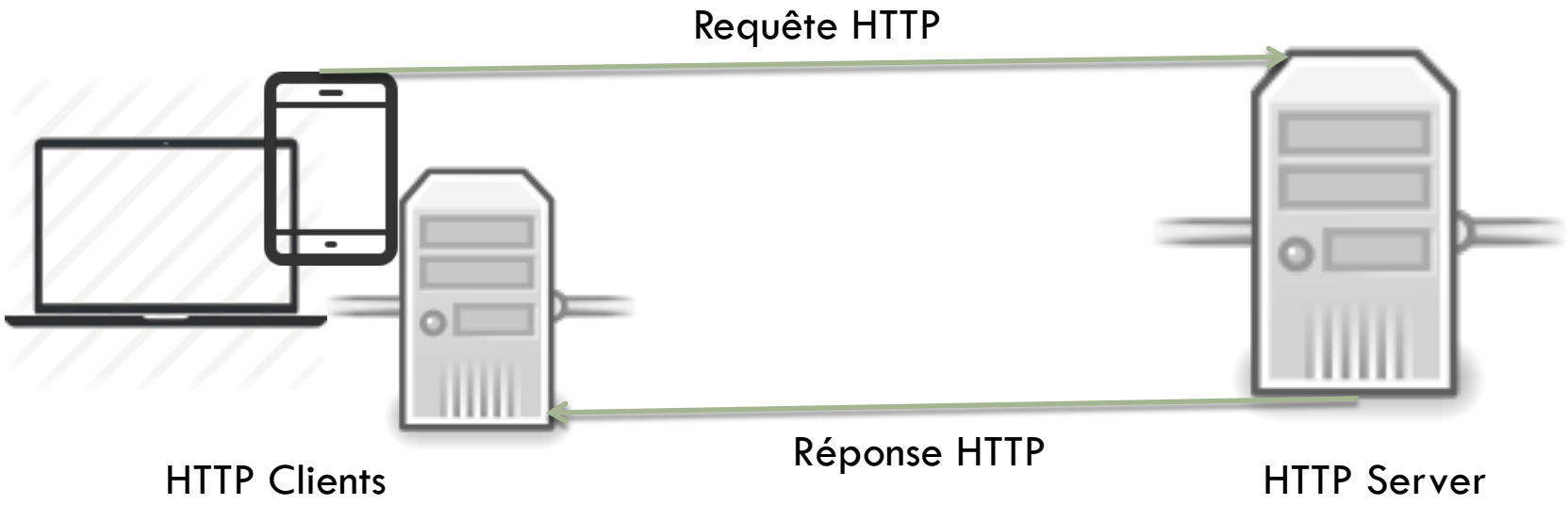
REST → utilisation

- Utiliser dans le développement des applications orientés ressources (ROA) ou orientées données (DOA)
- Les applications respectant l'architecture REST sont dites RESTful

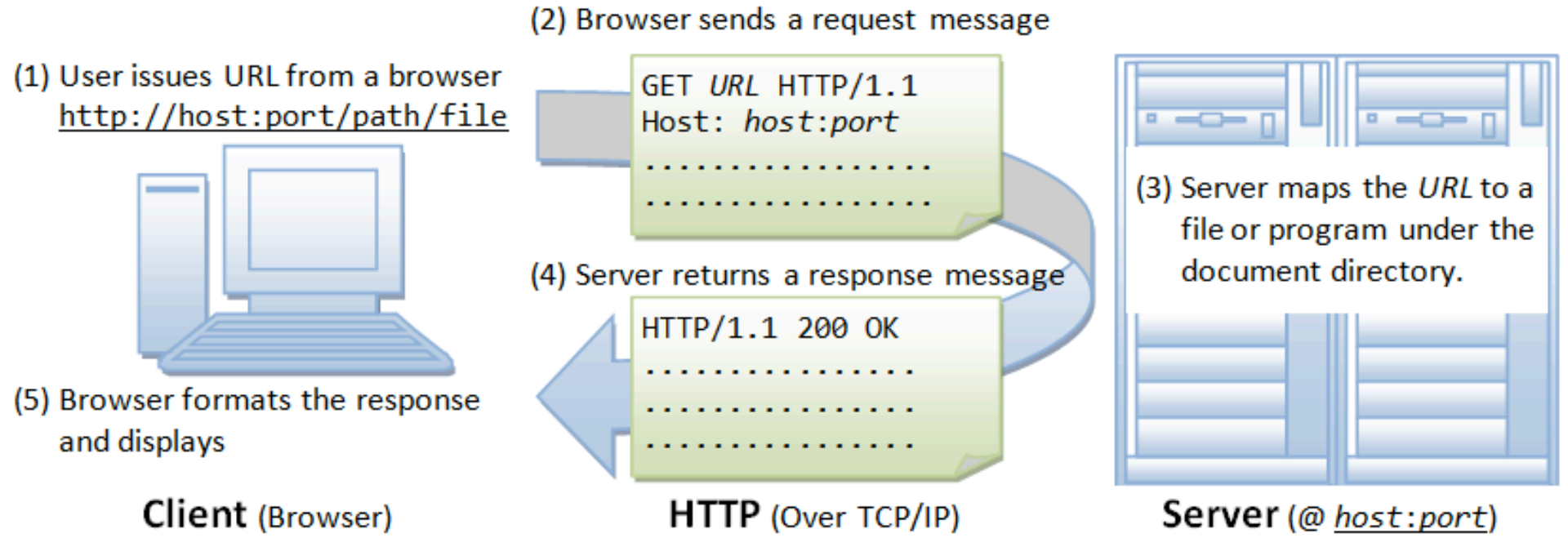
Rappel → Protocol HTTP

Le Protocole HTTP

- HyperText Transfer Protocol
- Protocole d'échanges d'information sur le web
- Basé sur TCP/IP



Enchainement Client Serveur



URL

- Unique Resource Location
- Identifie les ressources de manière unique sur le Web
- 4 parties
  - Protocole (http, ftp, mail, …)
  - Host (google.com)
  - Port (8080, 80)
  - Path (Chemin vers la ressource sur le serveur)

Requêtes HTTP

- Permet à un client de demander une ressource sur un serveur
- Format d'un message HTTP
  - **Header**
    - Request Line
    - Request Headers [Optional]
  - **Body**

Entête des requêtes HTTP

- Request Line

```
POST /bibliotheque/faces/views/categorie/Create.xhtml HTTP/1.1
```

- Request Headers

```
Host: localhost:8080
Connection: keep-alive
Content-Length: 176
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_0) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/39.0.2171.65 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost:8080/bibliotheque/faces/views/categorie/List.xhtml
Accept-Encoding: gzip, deflate
Accept-Language: fr,fr-FR;q=0.8,en;q=0.6
Cookie: JSESSIONID=d64a9484e61761662575b5d14af1
```

Corp des Requêtes HTTP

- Contient les données supplémentaires envoyées au serveur

```
j_idt13:nom:Miage
j_idt13:description:NTDP
```

Réponse HTTP

- Réponse du serveur au client

- Format d'une réponse HTTP
  - Response Message Header
    - Response Line
    - Response Headers
  - Response Message [Optional]

Entête des Réponses HTTP

- Response Line

```
HTTP/1.1 200 OK
```

- Response Headers

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.0  Java/Oracle Corporation/1.8)
Server: GlassFish Server Open Source Edition  4.0
Content-Type: text/html;charset=UTF-8
Date: Sun, 23 Nov 2014 16:05:39 GMT
Content-Length: 2274
```

Corp des Réponses HTTP

- Response Body

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><html xmlns="http://www.w3.org/1999/xhtml"><head><link
type="text/css" rel="stylesheet" href="/bibliotheque/faces/javax.faces.resource/theme.css?ln=primefaces-aristo"
/><link type="text/css" rel="stylesheet" href="/bibliotheque/faces/javax.faces.resource/css/jsfcrud.css" />
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <title>Create New Categorie</title></head><body>
        <h1>Create New Categorie
        </h1>
        <p><div id="messagePanel"><table><tr style="color: green"><td>Categorie was successfully created. </td>
</tr></table></div>
</html>
</html>
```

Méthodes HTTP

- HTTP définit un ensemble de méthode permet de caractériser les requêtes
  - GET : Récupérer des ressources à un serveur
  - POST : Envoyer des données à un serveur
  - PUT : Modifier des données
  - DELETE : Suppression de données
  - OPTIONS : Demander la liste des méthodes supportées par un serveur
  - Autres : HEAD, TRACE, CONNECT

REST et HTTP

- Des ressources Identifiées par des URIs (http://unice.fr/cursus/master/miage)
- Des actions sur les ressources correspondant aux types de requêtes HTTP : GET, POST, PUT, DELETE
- Représentation des ressources : Vue sur l'état de la ressource
  - Format d'échanges entre le client et le serveur (XML, JSON, text/plain,…)

Ressources

- Une ressource est un objet identifiable sur le système
- ➔ Livre, Client, Prêt

Une ressources n'est pas forcément une entité physique, elle peut être virtuelle (Prêt, Consultation,…)

- Une ressource est identifiée par une URI : Une URI identifie uniquement une ressource sur le système
  http://ntdp.miage.fr/bookstore/books/1

Clef primaire de la ressource dans la BDD

Actions sur les ressources

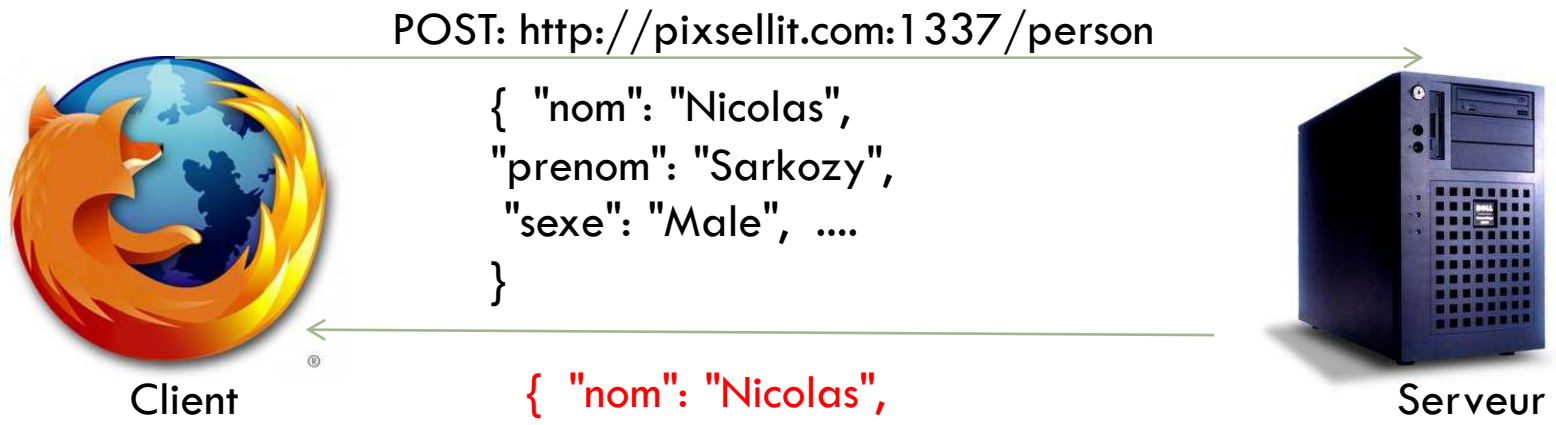
- Une ressource peut subir des actions correspondant aux opérations effectuées sur la ressource.
- Les opérations de bases (CRUD) sont identifiées par les types de requêtes HTTP (GET, PUT, POST, DELETE)

Opérations CRUD

- Architecture simple basée sur le protocole HTTP
- Les actions s'auto-décrivent et s'identifient aux types de requetés HTTP (ou méthode)
- **POST** : Créer une nouvelle ressource → Ajout de données
- **GET** : Récupérer une ressource sans la modifier
- **PUT** : Mettre à jour une ressource identifié par l'URI
- **DELETE** : Supprimer la ressource identifiée par l'URI

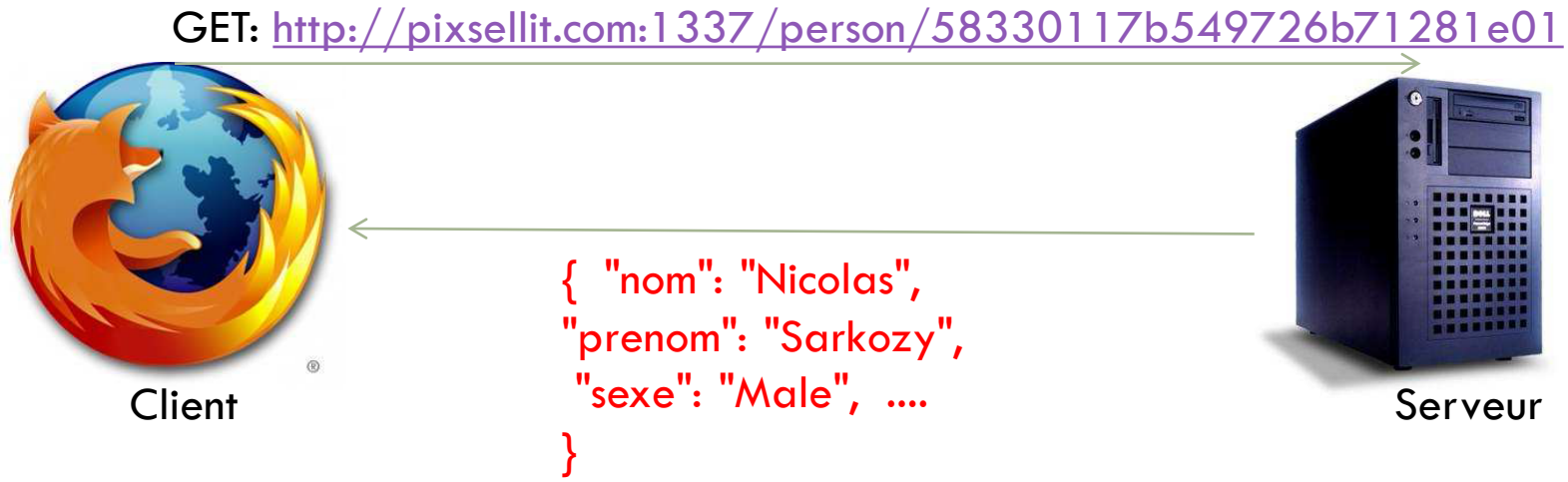
Méthode POST

- Crée une nouvelle ressource sur le système



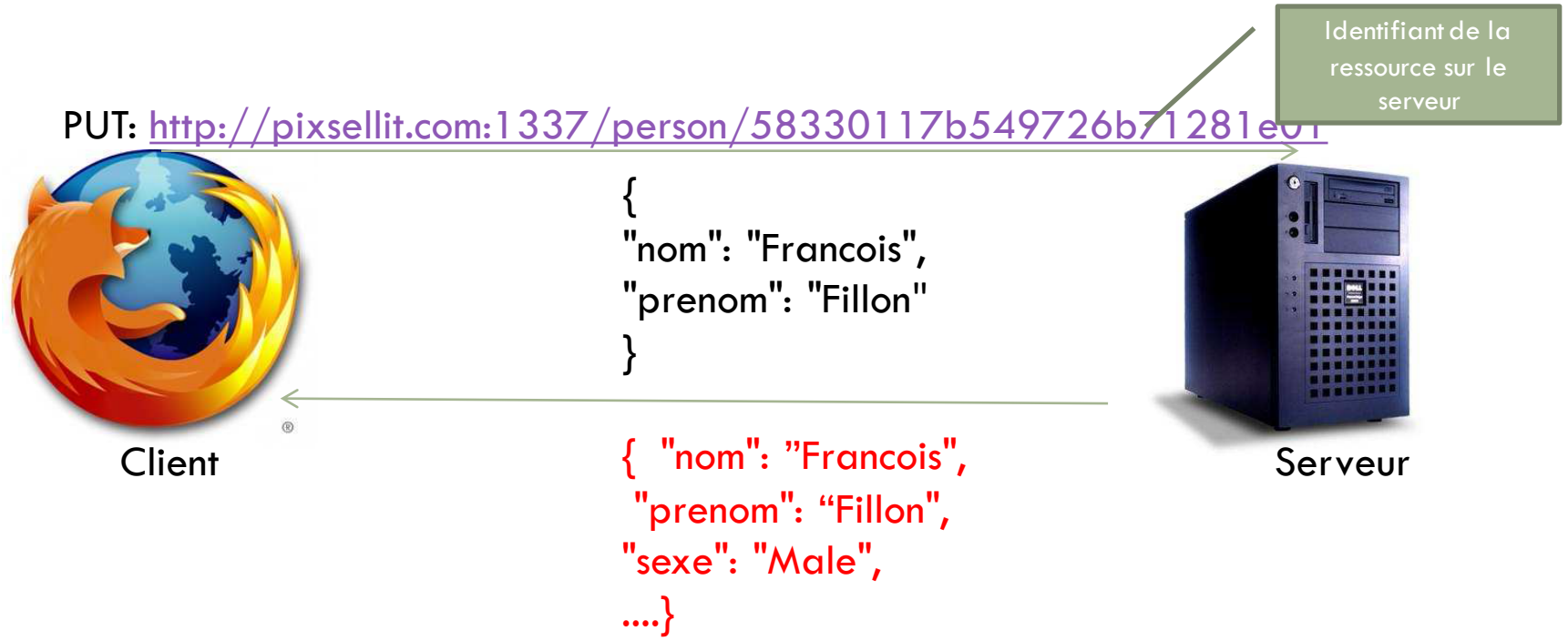
Méthode GET

- Demande une représentation de la ressource tel qu'elle est sur le système (pas de modification)



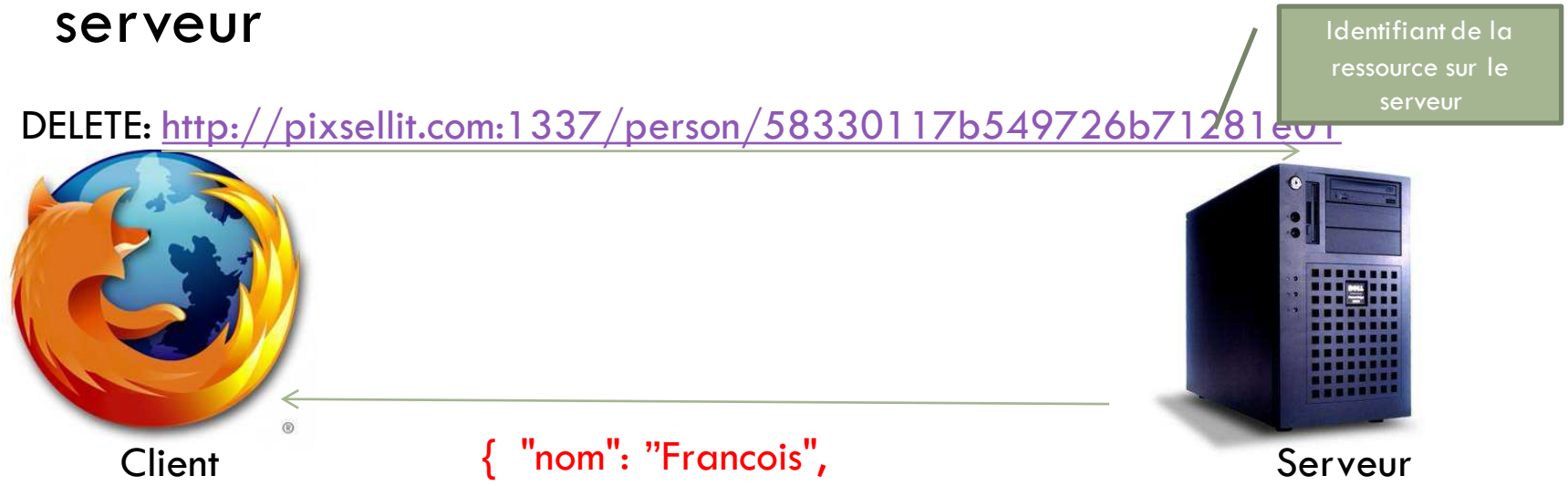
Méthode PUT

- Mise à jour de la ressource sur le système



Méthode DELETE

- Supprime la ressource identifiée par l'URI sur le serveur



REST, Méthode HTTP et Ressources

- L'opération à effectuer sur une ressource est déterminée par le type de la requête HTTP
- Plusieurs actions sont possibles pour une même URI
  ➔ Tout dépend du type de la requête

REST, Méthode HTTP et Ressources

- Crée une nouvelle personne → ***POST/http://exemple.com/rest/person***
- Liste des personnes → ***GET/http://exemple.com/rest/person***
- Récupérer une personne → ***GET/http://exemple.com/rest/person/{id}***
- Modifier une personne → ***PUT/http://exemple.com/rest/person/{id}***
- Supprimer une personne → ***DELETE/http://exemple.com/rest/person/{id}***

Les services RestFul

- Que se passe t-il
  - si on fait de la lecture avec un POST ?
  - Si on fait une mise à jour avec un DELETE ?
  - Si on fait une suppression avec un PUT ?

- ➔ REST ne l'interdit pas
- ➔ Mais si vous le faites, votre application ne respecte pas les exigences REST et donc n'est pas RESTFul

Représentation

Une représentation désigne les données échangées entre le client et le serveur pour une ressource:

- HTTP GET ➔ Le serveur renvoie au client l'état de la ressource
- PUT, POST ➔ Le client envoie l'état d'une ressource au serveur

Peut être sous différent format :

- JSON
- XML
- XHTML
- CSV
- Text/plain
- …..

Rappel → JSON

JSON

JSON « **J**ava**S**cript **O**bect **N**otation » est un format d'échange de données, facile à lire par un humain et interpréter par une machine.

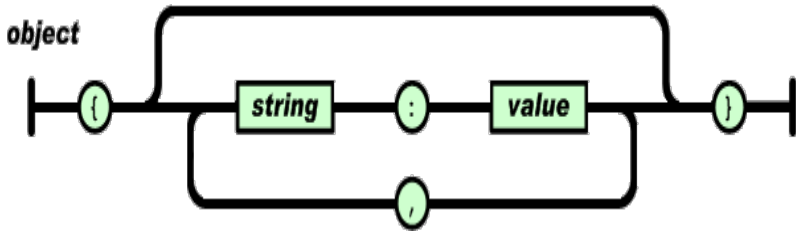
Basé sur JavaScript, il est complètement indépendant des langages de programmation mais utilise des conventions qui sont communes à toutes les langages de programmation (C, C++, Perl, Python, Java, C#, VB, JavaScript,….)

Deux structures :

- Une collection de clefs/valeurs → Object
- Une collection ordonnée d'objets → Array

JSON Object

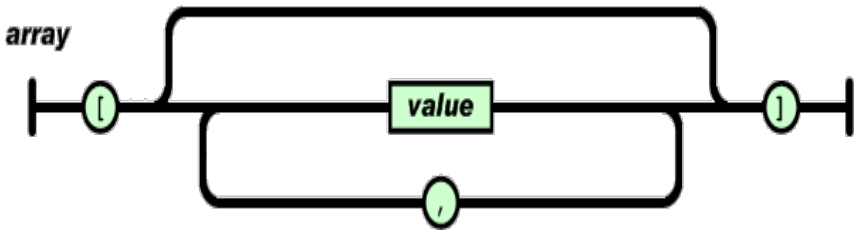
Commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paire clefs/valeurs. Une clef est suivie de « : » et les paires clef/valeur sont séparés par « , »




```
{ "id": 51,
"nom": "Mathematiques 1", "resume":
"Resume of math ", "isbn": "123654",
"categorie":
     {
     "id": 2, "nom": "Mathematiques",
     "description": "Description of
     mathematiques "
     },
"quantite": 42,
"photo": ""
}
```

JSON Array

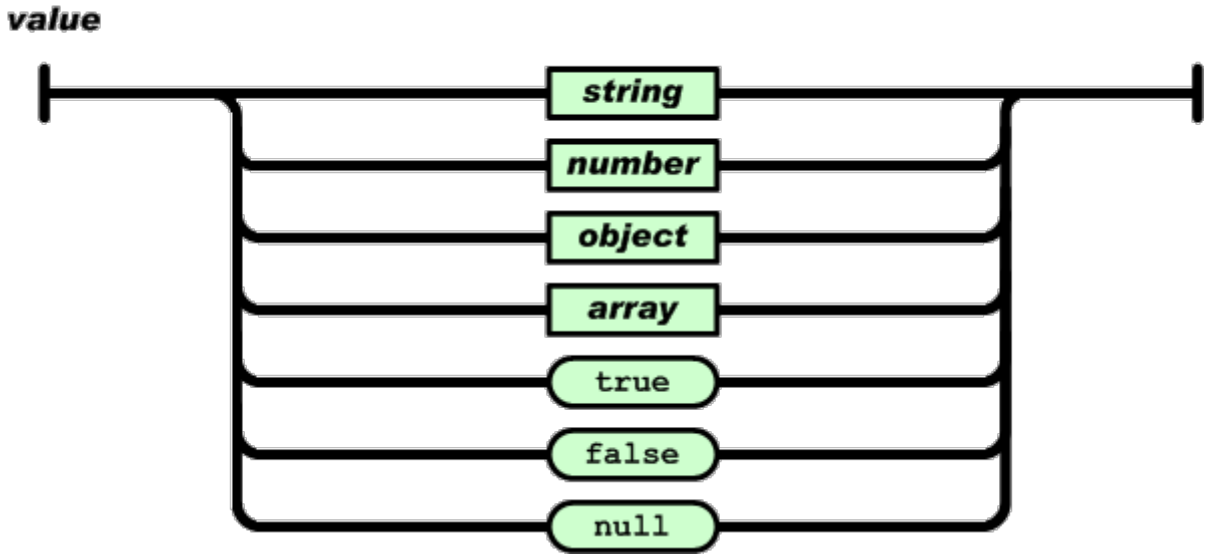
Liste ordonnée d'objets commençant par « [« et se terminant par «] », les objets sont séparés l'un de l'autre par « , ».




```
[
{ "id": 51,
"nom": "Mathematiques 1",
"resume": "Resume of math ",
"isbn": "123654",
"quantite": 42,
"photo": ""
},
{ "id": 102,
"nom": "Mathematiques 1",
"resume": "Resume of math ",
"isbn": "12365444455",
"quantite": 42,
"photo": ""
}
]
```

JSON Value

Un objet peut être soit un string entre « ""» ou un nombre (entier, décimal) ou un boolean (true, false) ou null ou un objet.



WADL

- Web Application Description Language
- Standard du W3C
- Permet de décrire les éléments des services
  - Resource, Méthode, Paramètre, Réponse
- Permet d'interagir de manière dynamique avec les applications REST

→ Moins exploité que le WSDL pour les Services SOAP

WADL

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<application xmlns="http://wadl.dev.java.net/2009/02">
   <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.0 2013-05-03 14:50:15"/>
   <grammars/>
  ▼<resources base="http://localhost:8080/Bibliotheque/webresources/">
    ▼<resource path="category">
      ▼<method id="test" name="GET">
        ▼<response>
            <representation mediaType="application/xml"/>
            <representation mediaType="application/json"/>
          </response>
        </method>
      ▼<method id="apply" name="OPTIONS">
        ▼<request>
            <representation mediaType="*/*"/>
          </request>
        ▼<response>
            <representation mediaType="application/vnd.sun.wadl+xml"/>
          </response>
        </method>
      ▼<method id="apply" name="OPTIONS">
        ▼<request>
            <representation mediaType="*/*"/>
          </request>
        ▼<response>
            <representation mediaType="text/plain"/>
          </response>
        </method>
      ▼<method id="apply" name="OPTIONS">
        ▼<request>
            <representation mediaType="*/*"/>
          </request>
        ▼<response>
            <representation mediaType="*/*"/>
          </response>
        </method>
```

```
▼<resource path="application.wadl">
  ▼<method id="getWadl" name="GET">
    ▼<response>
        <representation mediaType="application/vnd.sun.wadl+xml"/>
        <representation mediaType="application/xml"/>
      </response>
    </method>
  ▼<method id="apply" name="OPTIONS">
    ▼<request>
        <representation mediaType="*/*"/>
      </request>
    ▼<response>
        <representation mediaType="text/plain"/>
      </response>
    </method>
  ▼<method id="apply" name="OPTIONS">
    ▼<request>
        <representation mediaType="*/*"/>
      </request>
    ▼<response>
        <representation mediaType="*/*"/>
      </response>
    </method>
```

REST VS SOAP

Carte Postale Vs. Courier

REST VS SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body> <ns2:hello
xmlns:ns2="http://services.bibliotheque.ntdp.miage.unice.fr/">
<name>Miage NTDP</name>
</ns2:hello>
</S:Body>
</S:Envelope>
```

SOAP

Client

Serveur

http://localhost:8080/Bibliotheque/webresources/category/Miage%20NTDP

REST

Client

Serveur

Services Web étendus VS REST

SOAP

- Avantages
  - Standardisé
  - Interopérabilité
  - Sécurité (WS-Security)
- Inconvénients
  - Performances (enveloppe SOAP supplémentaire)
  - Complexité, lourdeur
  - Cible l'appel de service

Services Web étendus VS REST

REST

- Avantages
  - Simplicité de mise en œuvre
  - Lisibilité par un humain
  - Evolutivité
  - Repose sur les principes du web
  - Représentations multiples (XML, JSON,…)
- Inconvénients
  - Sécurité restreinte par l'emploi des méthodes HTTP
  - Cible l'appel de ressources

WADL

- Web Application Definition Language est un langage de description des services REST au format XML. Il est une spécification di W3C initié par SUN (www.w.org/Submission/wadl)
- Il décrit les éléments à partir de leur type (Ressources, Verbes, Paramètre, type de requête, Réponse)
- Il fournit les informations descriptives d'un service permettant de construire des applications clientes exploitant les services REST.

Exercice

- Premier pas avec les services REST
- Installer le plugin PostMan (si vous n'e l'avez pas déjà)
- Effectuer les opérations de base (CRUD) sur les ressources identifiées par (http://pixsellit.com:1337/person)

Partie 2

Développer des Web Services REST avec JAVA

Mais avant….

1. Lancer Netbeans
2. Créer un projet de type Web Application
3. Créer un package (com.example)
4. Ajouter une classe dans le package
5. Lancer le projet
6. C’est parti ☺

JAX-RS

- Acronyme de Java API for RestFul Web Services
- Version courante 2.0 décrite par JSR 339
- Depuis la version 1.1, il fait partie intégrante de la spécification Java EE 6
- Décrit la mise en œuvre des services REST web coté serveur
- Son architecture se repose sur l'utilisation des classes et des annotations pour développer les services web

JAX-RS ➔ Implémentation

- JAX-RS est une spécification et autour de cette spécification sont développés plusieurs implémentations
  - JERSEY : implémentation de référence fournie par Oracle (http://jersey.java.net)
  - CXF : Fournie par Apache (http://cfx.apache.org)
  - RESTEasy : fournie par JBOSS
  - RESTLET : L'un des premiers framework implémentant REST pour Java

JERSEY

- Version actuelle 2.24.1 implémentant les spécifications de JAX-RS 2.0
- Intégré dans Glassfish et l'implémentation Java EE (6,7)
- Supportés dans Netbeans

JAX-RS : Développement

- Basé sur POJO (Plain Old Java Object) en utilisant des annotations spécifiques JAX-RS
- Pas de modifications dans les fichiers de configuration
- Le service est déployé dans une application web
- Pas de possibilité de développer le service à partir d'un WADL contrairement à SOAP
- Approche Bottom/Up
  - Développer et annoter les classes
  - Le WSDL est automatiquement généré par l'API

Annotation JAX-RS

La spécification JAX-RS dispose d'un ensemble d'annotation permettant d'exposer une classe et ses méthodes dans un services web :

- @Path
- @GET, @POST, @PUT, @DELETE
- @Produces, @Consumes
- @PathParam

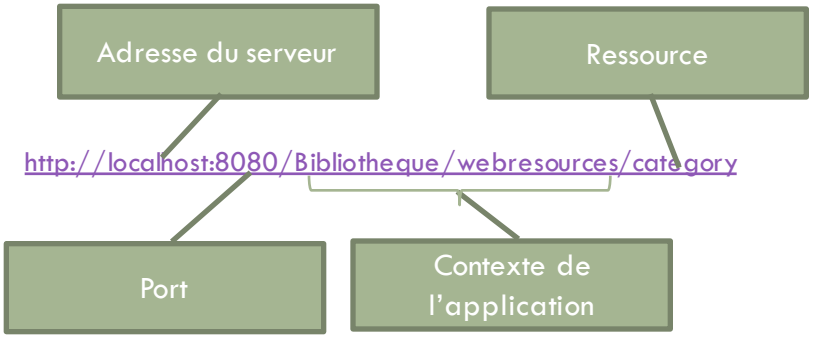
Les Annotations de Classes

- L'annotation @Path permet de définir l'URI des ressources modélisés par une classe
- A positionner au-dessus de la déclaration d'une classe
- Syntaxe :
  ```
  @Path("nom_de_la_ressource")
  ```
- Expose la classe comme ressource dans le WS

Annotations de classes

- @Path : Définit la racine des ressources (Root Racine Ressources)
- Sa valeur correspond à l'URI relative de la ressource

```
@Path("category")
public class CategoryService  {
……
}
```



Exemple

```
@Path("hello")
public class HelloWorld{
@GET
@Produces(MediaType.TEXT_PLAIN)
public String sayHello()
   {
     return "Hello World!";
   }
}
```

http://localhost:8080/rest/webresources/hello/test

URIs de Méthodes

- @Path peut être utilisée pour annoter des méthodes d'une classe
- Permet de définir des URIs pour les actions sur une une ressource
- L'URI résultante est la concaténation entre le valeur de @Path de la classe et celle de la méthode

http://localhost:8080/Bibliotheque/webresources/category/test

Exemple

```
 @Path( "hello")
 public class CategoryFacade {
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("greetings")
  public String hello()
   {
    return "Hello World!";
   }
}
```

http://localhost:8080/rest/webresources/hello/greetings

Annotation @Produces

- @Produces placé sur les méthodes
- Spécifie le format de données renvoyées par la méthode
- Correspond à la clef Content-Type de l'en tête des requêtes HTTP
-

Annotation @Consumes

- @Consumes spécifie le format des données d'entrées de la méthode
- Placé au dessus de la déclaration des méthodes
- Correspond à la clef Accept de l'en tete des requetés HTTP

Valeurs @Consumes/@Produces

- Plusieurs Mime-Type
  - text/plain
  - application/json
  - application/xml
  - application/x-www-form-urlencoded
  - Application/csv
  - …

Annotation Dynamiques

- La valeur définie dans l'annotation @Path n'est forcément un constante, elle peut être variable.
- Possibilité de définir des expressions plus complexes, appelées Template Parameters
- Les contenus complexes sont délimités par « {} »
- Possibilité de mixer dans la valeur @Path des expressions régulières

http://localhost:8080/Bibliotheque/webresources/category/hello/Miage

Paramètres Dynamiques

- @PathParam → Récupérer dans l'URL (Path)
- @FormParam → Récupérer dans le formulaire (Form)
- @QueryParam → Récupérer dans les paramètres de l'URL (Query)
- @HeaderParam → Récupérer dans l'entête de la requête (Header)
- @CookieParam → Récupérer les informations de cookies

@PathParam – Exemple

Dire bonjour à la personne passée en paramètre

http://localhost:8080/rest/webresources/hello/greetings/Miage

```
 @Path( "hello")
 public class HelloWorld{
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("calcul")
  public String calculPost(@FormParam( »nb1") int
nom){
      return "Hello " + nom;
    }
}
```

@PathParam – Exemple 2

Calcule la somme de deux nombres passés en paramètre

http://localhost:8080/rest/webresources/hello/calcul/12+20

```
 @Path( "hello")
 public class HelloWorld{
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path( "calcul/{nb1}+{nb2}")
  public double calcul(@PathParam( "nb1") nb1,
(@PathParam( "nb2") int nb2){
      return nb1 + nb2;
    }
}
```

@QueryParam– Exemple 2

Calcule la somme de deux nombres passés en paramètre

http://localhost:8080/rest/webresources/hello/calcul/?nb1=12&nb2=20

```
 @Path( "hello")
 public class HelloWorld{
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path( "calcul/ ")
  public double calcul(@QueryParam( "nb1") nb1, (@
QueryParam( "nb2") int nb2){
     return nb1 + nb2;
    }
}
```

Exercice

- Créer une méthode qui effectue une opération quelconque sur deux nombres
  - Les nombres doivent être passés dans le path de l'URL et sont de types entiers
  - Le type d'opération dans l'en-tête
  - Les opérations supportés sont addition, soustraction, multiplication et division
  - Le type de retour doit être une double
  - Attention aux erreurs comme une division par zero

Type d'actions des méthodes

- Quatre types principaux:
  - @GET → Opération de lecture (Read)
  - @POST → Opération d'écriture (Create)
  - @PUT → Opération d'écriture (Create/Update)
  - @DELETE → Opération de suppression (Delete)

- Rappel → Correspondent aux types de requête HTTP

Annotation @GET

- @GET Placé sur les méthodes à exposer
- Pour des opérations de lecture
- Répond uniquement aux requêtes HTTP de type GET
- Accepte des données dans l'URL de la requête
- Une annotation @Consumes serait obsolète
- @Produces format des données de retour (Obligatoire)

Annotation @POST

- @POST pour annoter les méthodes
- Précise que la méthode n'est accessible que par des requêtes HTTP de type POST
- Opération d'écriture (Create)
- Doit recevoir logiquement des paramètres dans le corps de la requête
- Peut accepter des paramètres dans l'URL
- @Consumes pour définir le format des données d'entrée
- @Produces format des données de retour (optionnel)

Annotation @PUT

- @POST Précise que la méthode n'est accessible que par des requêtes HTTP de type POST
- Opération d'écriture (Create, Update)
- Doit recevoir logiquement des données dans le corps de la requête
- Peut accepter des paramètres dans l'URL
- @Consumes pour définir le format des données d'entrée
- @Produces format des données de retour (optionnel)

Annotation @DELETE

- @DELETE Précise que la méthode n'est accessible que par des requêtes HTTP de type DELETE
- Opération de suppression (Delete)
- Peut accepter des paramètres dans l'URL
- @Consumes serait obsolète
- @Produces format des données de retour (pas obligatoire)

@FormParam

- Permet de recevoir des données d'un formulaire
- Associé aux données d'entrée de type application/x-www-form-urlencoded

```
 @Path( "hello")
 public class HelloWorld{
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Consumes(MediaType. APPLICATION_FORM_URLENCODED)
  @Path( "calcul/ ")
  public double calcul(@FormParam( "nb1") nb1, (@
FormParam( "nb2") int nb2){
      return nb1 + nb2;
    }
}
```

@GET, @POST, @PUT, @DELETE

- Permettent de mapper une méthode à un type de requête HTTP
- Ne sont utilisables que sur des méthodes
- Plusieurs méthodes peuvent avoir le même chemin, le mapping uri/méthode est fait automatiquement par JAX-RS en fonction du type de la requête

http://localhost:8080/Bibliotheque/webresources/category/test

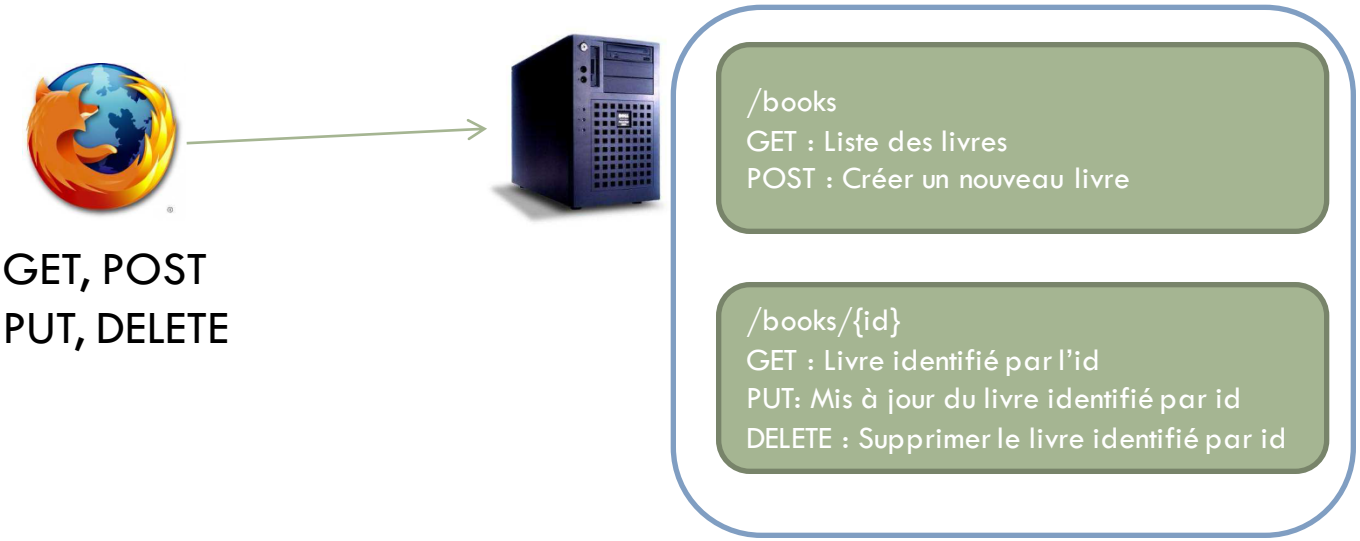
```
@GET
    @Produces({MediaType.APPLICATION_JSON,
MediaType.APPLICATION_XML})
    @Path("hello/{nom}/{prenom}")
    public String hello(@PathParam("nom") S
nom, @PathParam("prenom") String prenom) {
        return "GET " + nom + " " + prenom
    }
```

```
 @POST
    @Produces({MediaType.APPLICATION_JSON,
MediaType.APPLICATION_XML})
    @Path("hello/{nom}/{prenom}")
    public String helloPost(@PathParam("nom")
String nom, @PathParam("prenom") String prenom) {
        return "POST " + nom + " " + prenom;
    }
```

[GET/POST] http://localhost:8080/Bibliotheque/webresources/hello/Miage/NTDP

@GET, @POST, @PUT, @DELETE

- Les opérations CRUD sur les ressources sont réalisées au travers des méthodes de la requête HTTP



Contenus Personnalisés

- La spécification JAXB permet d'utiliser des types personnalisés (i.e. Class)
- JAXB permet de faire le mapping XML ←→POJO
- Manipuler les objets directement sans avoir à gérer directement du XML
- JAXB définit des annotations permettant de faire le mapping
  - @XmlRootElement, @XmlElement, @XmlType

Contenus Personnalisés

- Toute class (entité) annoté par @XmlRootElement est automatiquement mappé en XML
- Le format JSON est également supporté
- Le format adopté dépend des annotations
  - @Consumes → Désérialisation
  - @Produces → Sérialisation
- XML : text/xml, application/xml…
- JSON : application/json

Outils de test

- Il existe de nombreux outils en ligne permettant de tester les services Web REST

- Certains sont disponibles sous forme d'extansion que vous pouvez installer dans les navigateurs
  - RestConsole
  - PostMan

A vos marques !