# Les services web

***Pr. Youssef Saadi***

Master Informatique Décisionnelle

Faculté Des Sciences Et Techniques
Université Sultan Moulay Slimane  Béni-Mellal

AU: 2019/2020

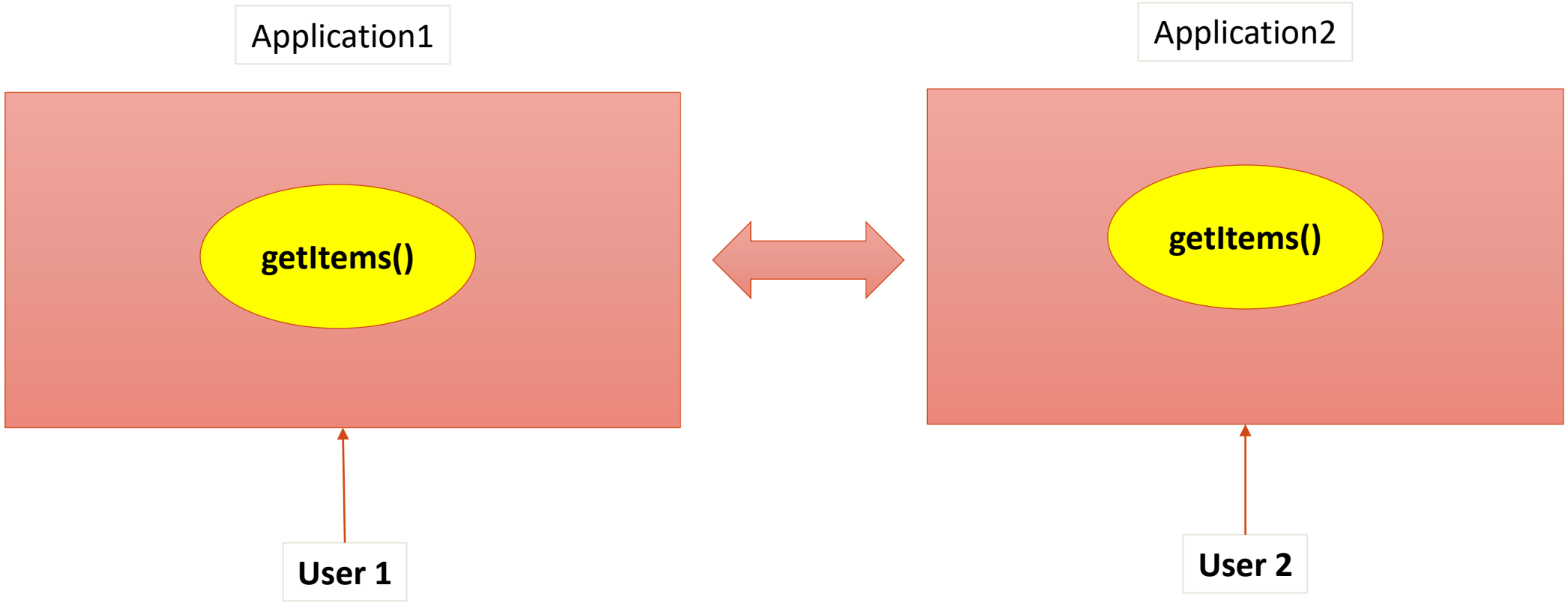
# Objectifs

- SOAP
  - Initiation au protocole et les Web services SOAP
  - Comprendre les enveloppes SOAP
  - Créer et tester des services SOAP
- REST
  - Comprendre l'architecture des applications compatibles REST
  - Exposer des services REST
  - Consommer des services REST dans de applications Web

# SOAP et REST API

| SOAP | REST |
| --- | --- |
| Jax-ws (Java API for XML – Web Service) | Jax-rs (Java API – RESTful Web Service) |
| Simple Object Access Protocol | Representational State Transfer |
| Older | Newer |

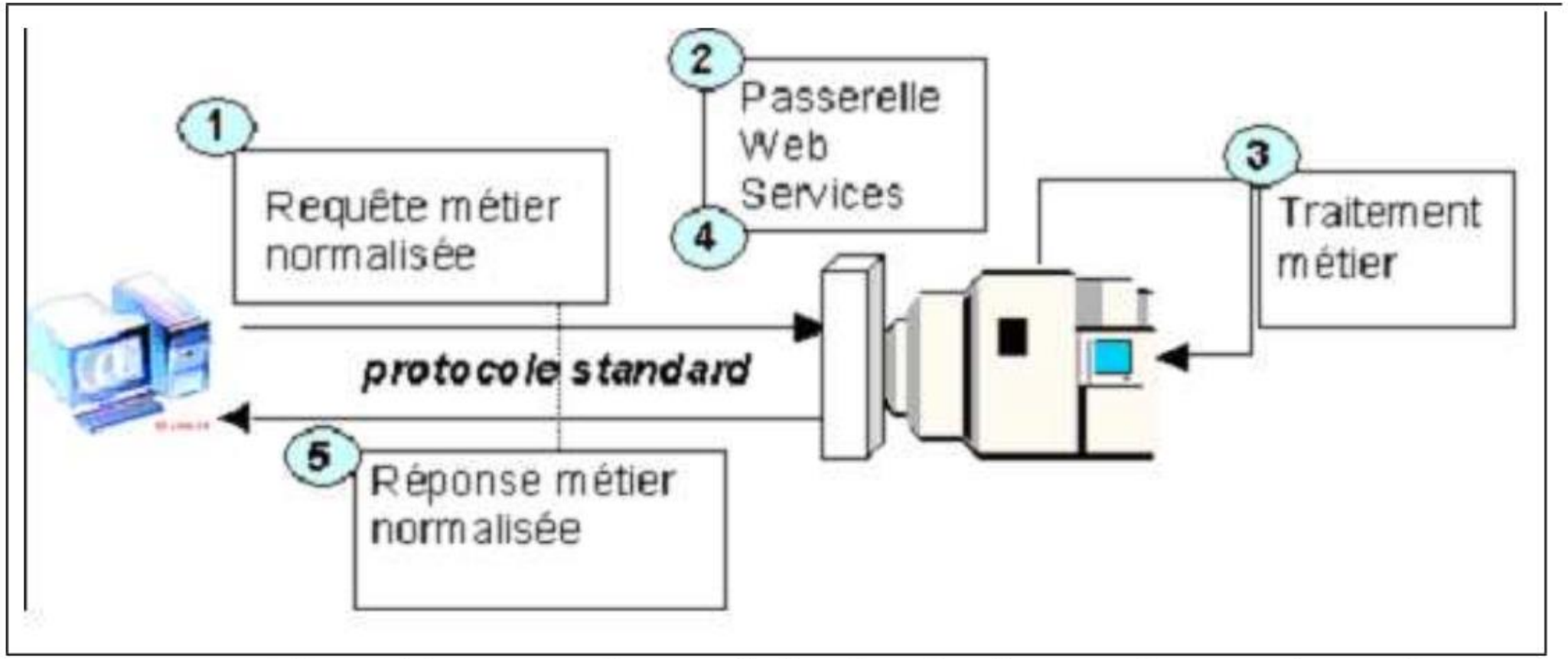
# Introduction



# Notion de Service Web

- Selon la définition du W3C (World Wide Web Consortium), un Web service (ou service Web) est une **application** appelable via Internet - par une autre application d'un autre site Internet permettant l'échange de données (de manière textuelle) afin que l'application appelante puisse intégrer le résultat de l'échange à ses propres analyses. Les requêtes et les réponses sont soumises à des standards et normalisées à chacun de leurs échanges.
- Un Web service est un mécanisme qui tend à donner plus d'interactions pour permettre à deux entités **hétérogènes** (entreprises, clients, applications, etc. ...) de dialoguer au travers du réseau Internet.

# Fonctionnement d'un échange de données grâce aux services Web

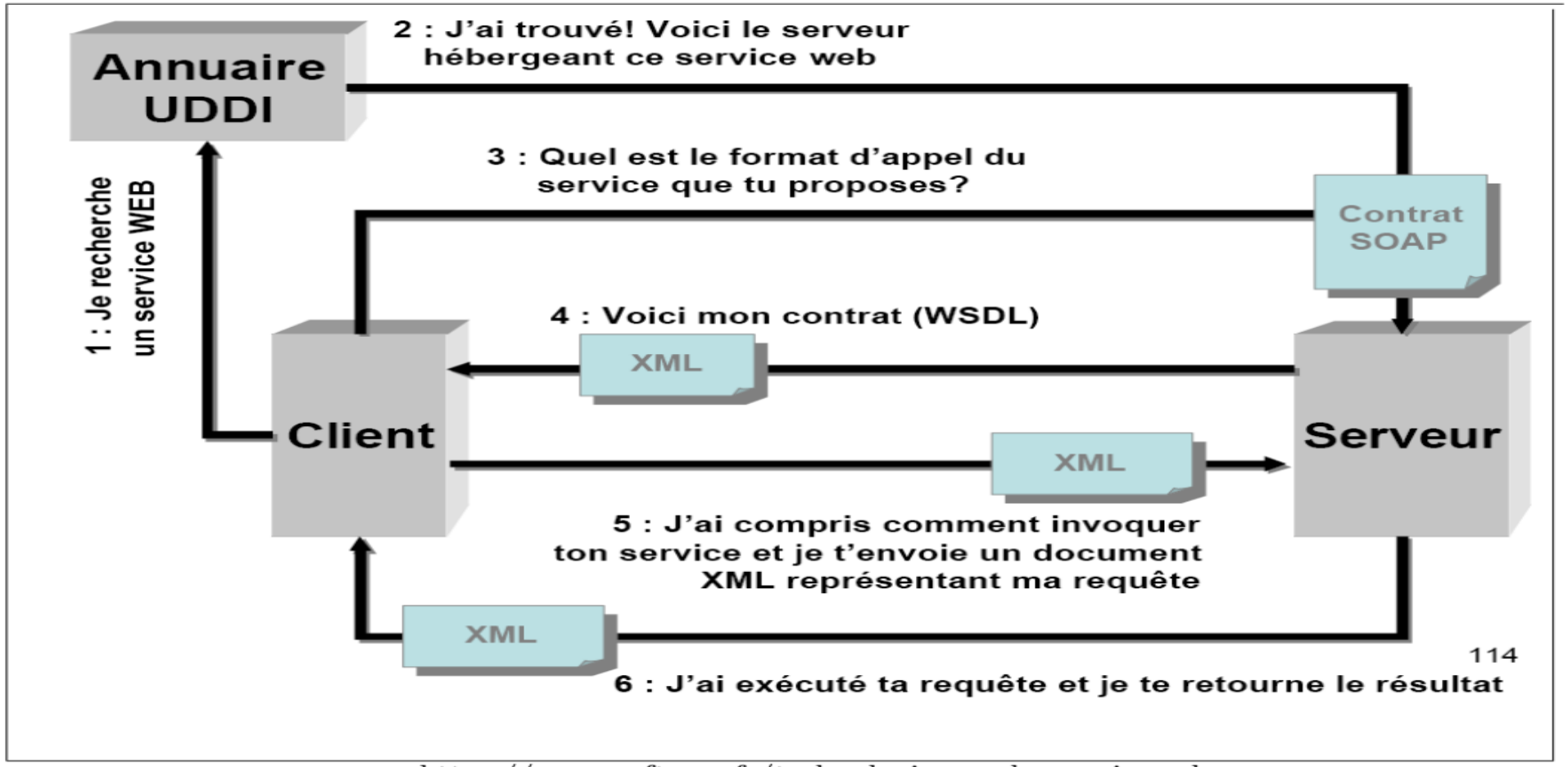



source : http ://www.softeam.fr/technologies_web_services.php

# Simple Object Access Protocol

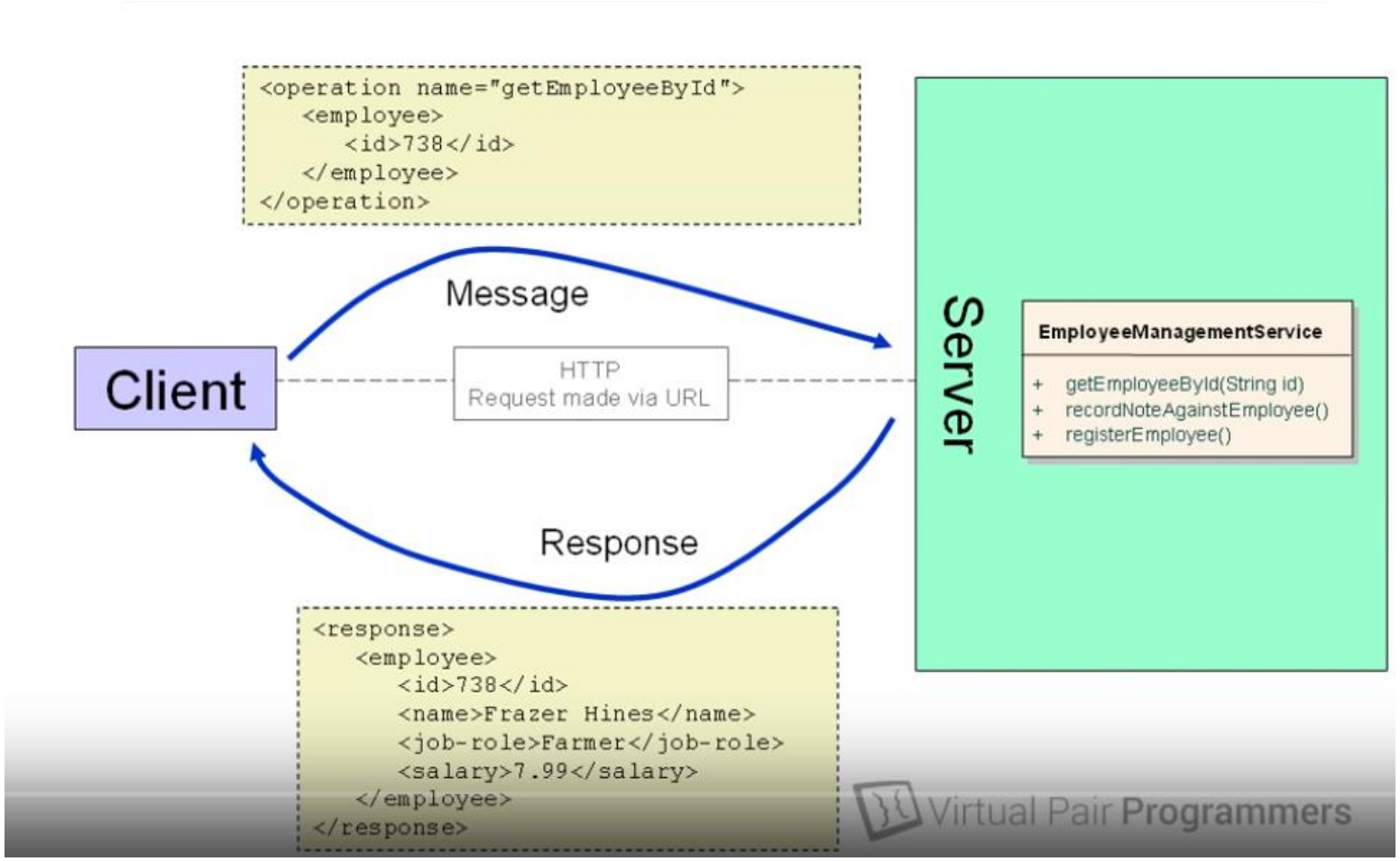
- Protocole d'échanges d'informations dans un environnement distribué basé sur XML
  - Interopérabilité entre applications d'une même entreprise (Intranet)
  - Interopérabilité inter entreprises entre applications et services web
- Similaire au protocole RCP
- SOAP peut être utilisé de concert avec plusieurs autres protocoles : HTTP, SMTP, POP
  - HTTP est le plus utilisé

# Architecture des services web




source : http ://www.softeam.fr/technologies web services.php

# Architecture SOAP



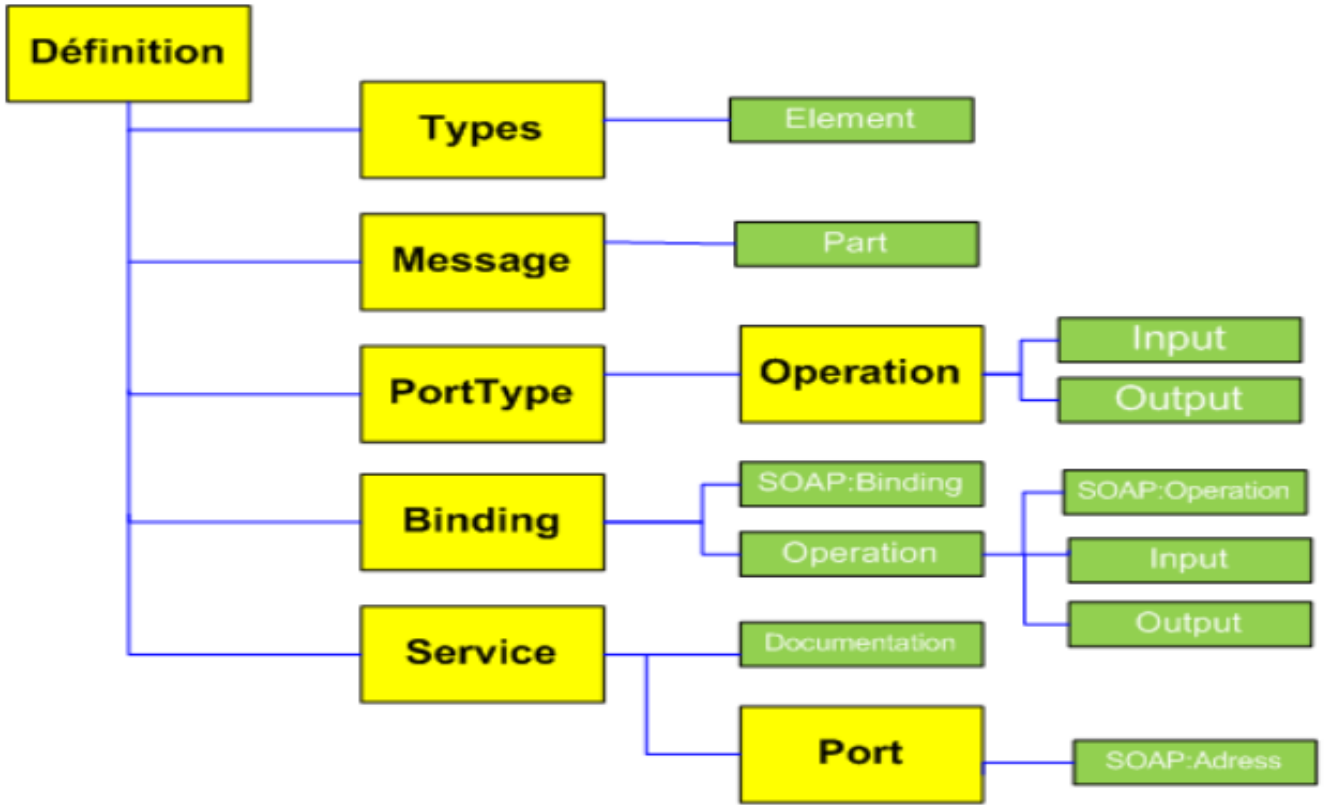
# Exemple: Le web service

```
@WebService(name = "perws")
public class PersonWS {
    @WebMethod(operationName = "create")
    public Person createPerson(@WebParam(name ="age" )int a, @WebParam(name ="name" )String n) {
        return new Person(a,n);
    }

    @WebMethod(operationName = "persons")
    public List<Person> getPersons(){
        List<Person> lst = new ArrayList<>();

        lst.add(createPerson((int)Math.random()*100, "A"));
        lst.add(createPerson((int)Math.random()*100, "B"));
        lst.add(createPerson((int)Math.random()*100, "C"));

        return lst;

    }

}
```

# Exemple: Le contrat WSDL

```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://soap.webservice.youssadi.com/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="PersonWSService" targetNamespace="http://soap.webservice.youssadi.com/">
  ▼<wsdl:types>
    ▶<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://soap.webservice.youssadi.com/" elementFormDefault="unqualified"
     targetNamespace="http://soap.webservice.youssadi.com/" version="1.0">...</xs:schema>
   </wsdl:types>
  ▶<wsdl:message name="create">...</wsdl:message>
  ▶<wsdl:message name="createResponse">...</wsdl:message>
  ▶<wsdl:message name="persons">...</wsdl:message>
  ▶<wsdl:message name="personsResponse">...</wsdl:message>
  ▶<wsdl:portType name="perws">...</wsdl:portType>
  ▼<wsdl:binding name="PersonWSServiceSoapBinding" type="tns:perws">
     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    ▶<wsdl:operation name="create">...</wsdl:operation>
    ▶<wsdl:operation name="persons">...</wsdl:operation>
   </wsdl:binding>
  ▶<wsdl:service name="PersonWSService">...</wsdl:service>
 </wsdl:definitions>
```

# Le format WSDL




Structure d'un document WSDL

# Requête/Réponse SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schem
  <soapenv:Header/>
  <soapenv:Body>
    <soap:create>
      <age>8</age>
      <!--Optional:-->
      <name>saadi</name>
    </soap:create>
  </soapenv:Body>
</soapenv:Envelope>
```

XML Raw

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soa
  <soap:Body>
    <ns2:createResponse xmlns:ns2="http://soap.webservice.
      <return>
        <age>8</age>
        <name>saadi</name>
      </return>
    </ns2:createResponse>
  </soap:Body>
</soap:Envelope>
```

# Requête/Réponse SOAP

http://localhost:8080/tuto-servlet/PersonWS

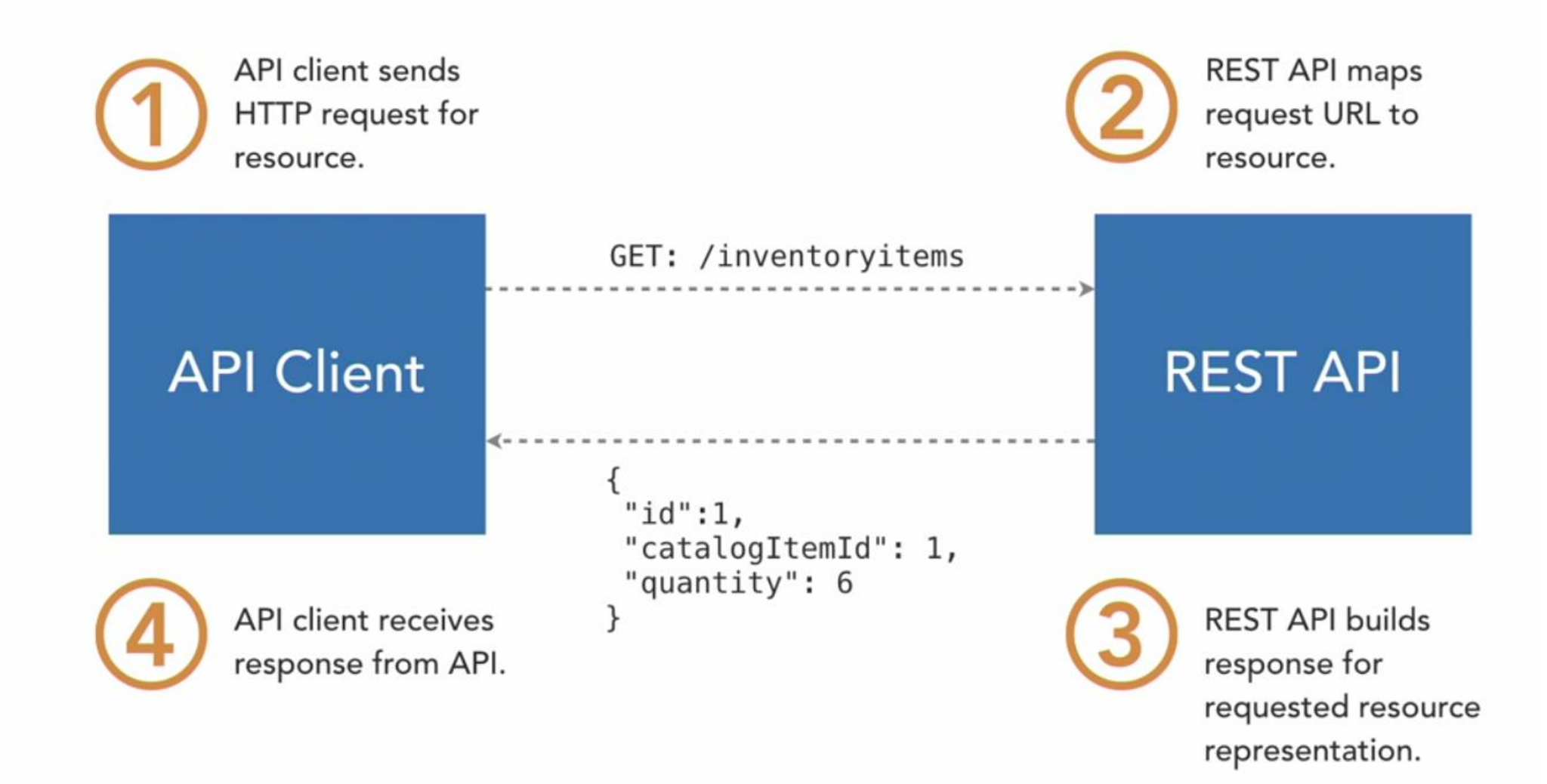
```
<soapenv:Envelope xmlns:soapenv="http://schem
  <soapenv:Header/>
  <soapenv:Body>
    <soap:persons/>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soa
  <soap:Body>
    <ns2:personsResponse xmlns:ns2="http://soap.webservic
      <return>
        <age>0</age>
        <name>A</name>
      </return>
      <return>
        <age>0</age>
        <name>B</name>
      </return>
      <return>
        <age>0</age>
        <name>C</name>
      </return>
    </ns2:personsResponse>
  </soap:Body>
</soap:Envelope>
```

# REST: **RE**presentational **S**tate **T**ransfert

- Le principe de REST est d'utiliser HTTP pour l'implémentation d'un Web Service, non plus comme simple protocole de transport, mais également pour définir l'API de chaque service c'est à dire la définition même des messages entre clients et serveur.
- La représentation des ressources est libre, utilisant différents formats de représentation XML, HTML, JSON.
- **REST** est un style d'architecture réseau pour Web Services qui met l'accent sur la définition de ressources identifiées par des URI, et utilise les messages du protocole HTTP pour définir la sémantique de la communication client/serveur.

# Principe REST



# Web service REST est sans état

- Les services REST sont sans états (Stateless)
  - Chaque requête envoyée au serveur doit contenir toutes les informations relatives à son état et est traitée indépendamment de toutes autres requêtes
  - Minimisation des ressources systèmes (pas de gestion de session, ni d'état)
- Interface uniforme basée sur les méthodes HTTP (GET, POST, PUT, DELETE):
  - **GET** pour le rapatriement d'une ressource,
  - **POST** pour une création,
  - **PUT** pour une modification/création,
  - **DELETE** pour un effacement.

# Ressource

- Une ressource est un objet identifiable sur le système
  - Livre, Catégorie, Client, Prêt
  - Prêt, Consultation, Facture…
- Une ressource est identifiée par une URI : Une URI identifie uniquement une ressource sur le système
  http://www.youssadi.com/bookstore/books/1
- Une ressource peut subir quatre opérations de bases CRUD correspondant aux quatre principaux types de requêtes HTTP (GET, PUT, POST, DELETE).

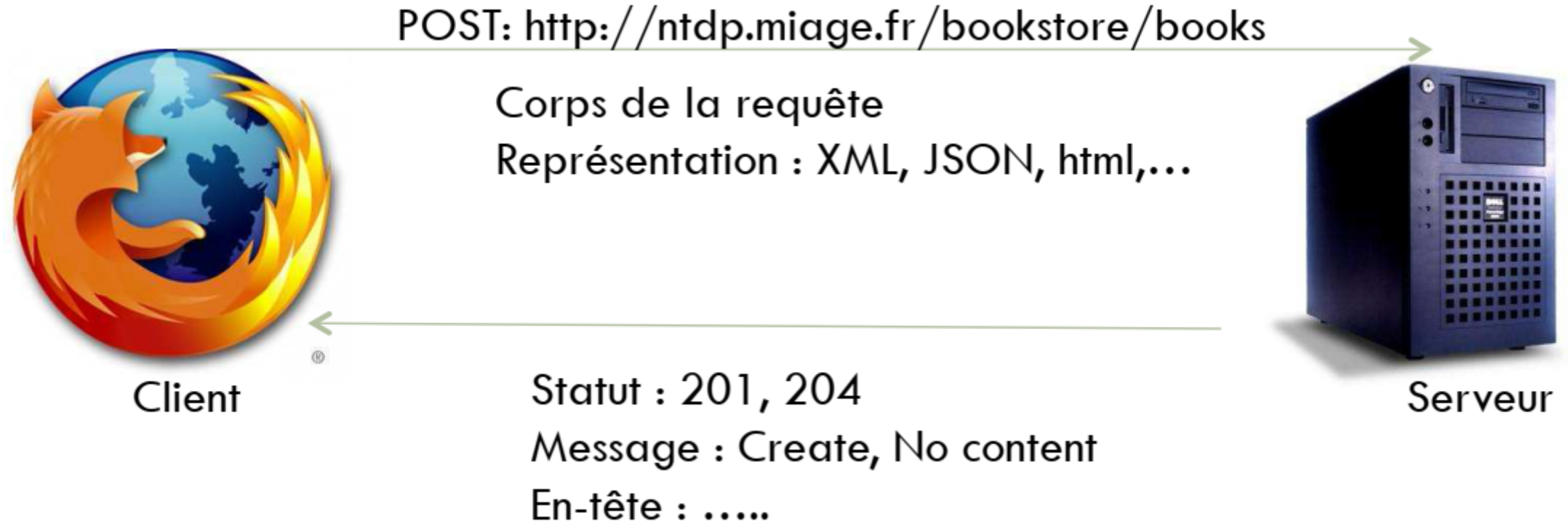
# Méthode GET

- La méthode GET renvoie une représentation de la ressource tel qu'elle est sur le système


Réf: https://slideplayer.fr/slide/1157200/

# Méthode POST

- La méthode POST crée une nouvelle ressource sur le système:



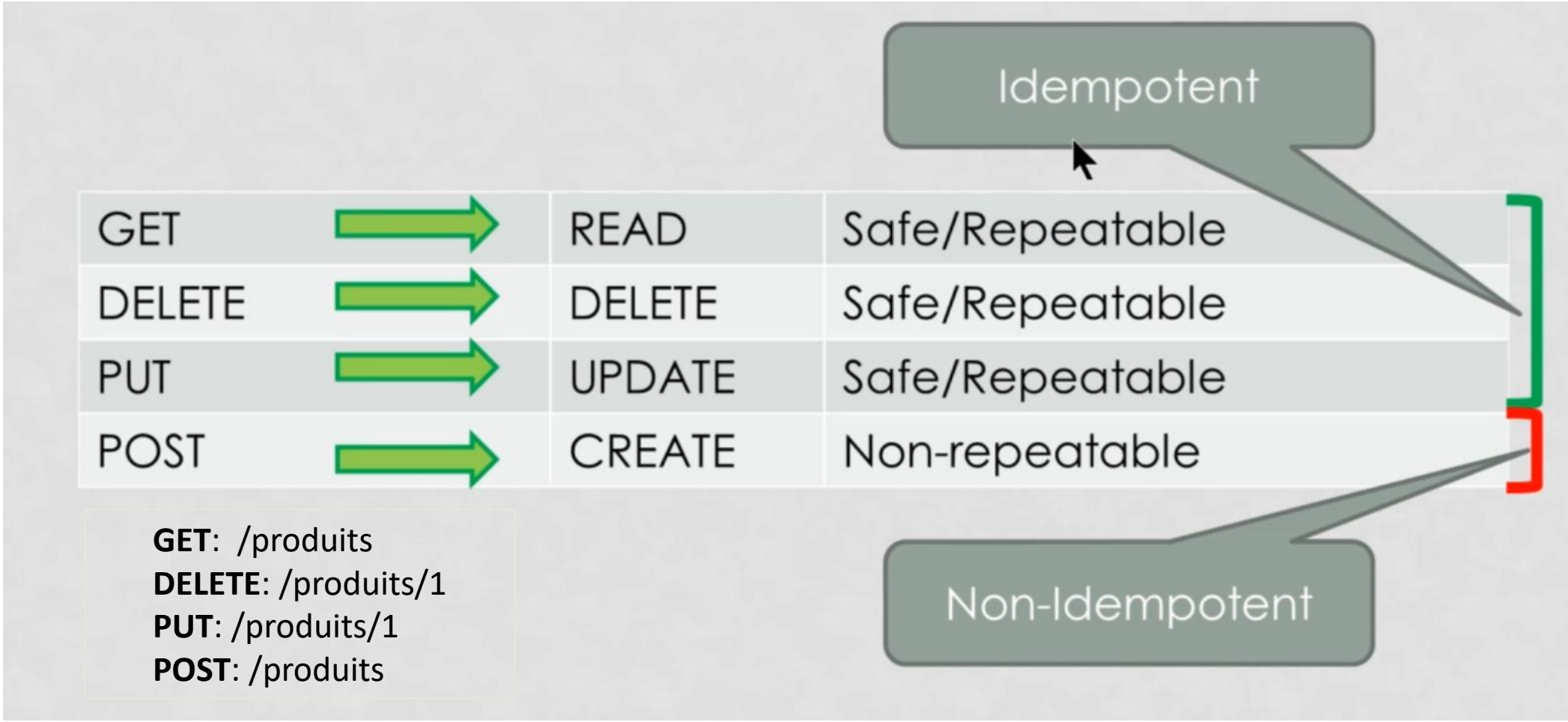
# Méthode DELETE

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

# La méthode PUT

- Mise à jour d'une ressource dans le système:

# Méthode « idempotent »



# Requête REST API

Header

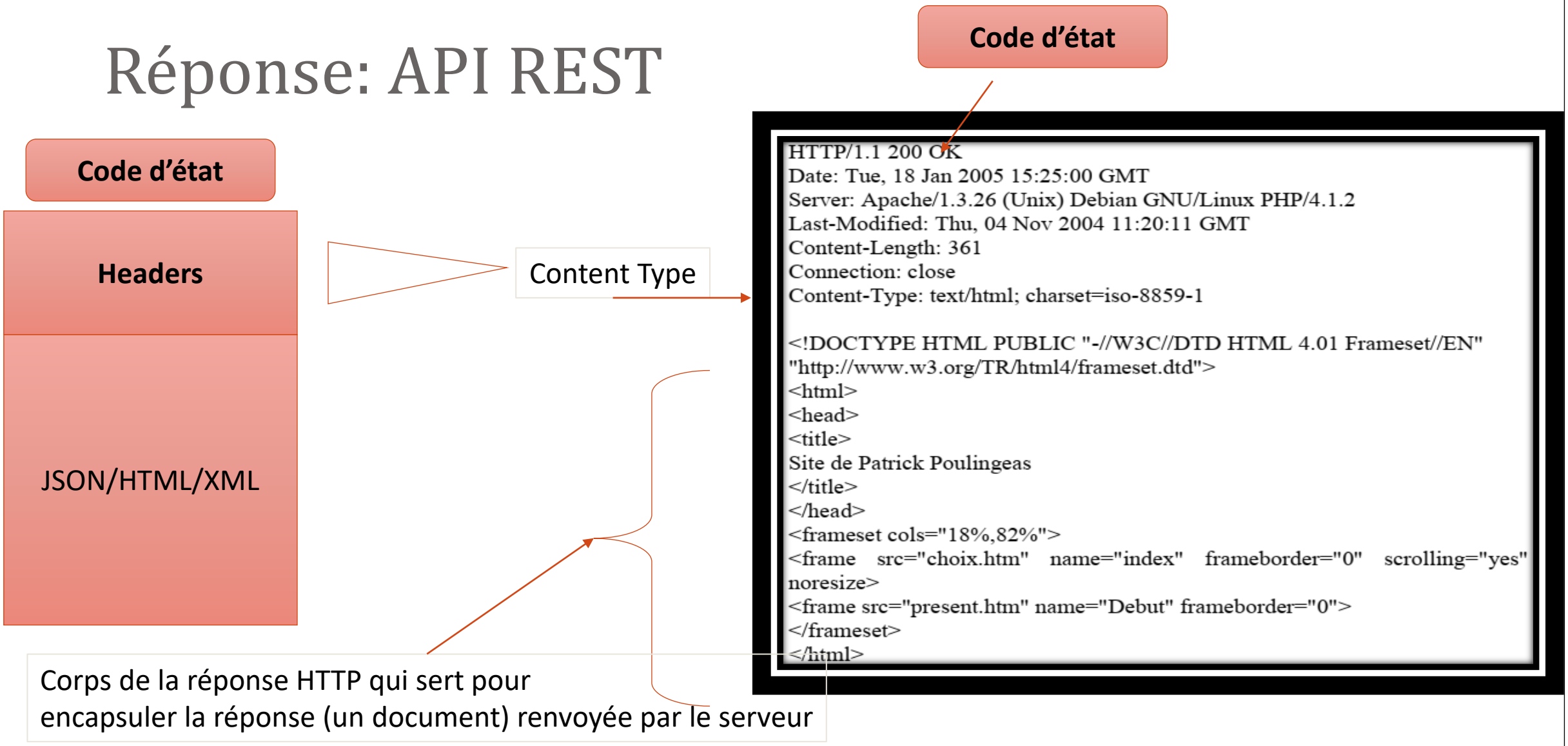
Le verbe HTTP

Le champ accept

```
GET /~poulingeas/index.htm HTTP/1.1
Accept: image/gif, image/jpeg, */*
Accept-Language: fr
Host: www.msi.unilim.fr
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

**Corps**

# Réponse: API REST



# Réponse REST

- Type de média : JSON

```
public class MessageEntity {
    private long id;
    private String message;
    private Date created;
    private String author;
...
}
```

Structure de la ressource coté serveur

```
{
  "id":"10",
  "message":"Hello world",
  "created":"2014-06-01T18:06:36.902",
  "author":"koushik"
}
```

Exemple d'objet JSON retourné dans une réponse REST

# Réponse REST

- Type de média: XML

```
public class MessageEntity {
    private long id;
    private String message;
    private Date created;
    private String author;
...
}
```

```
<messageEntity>
  <id>10</id>
  <message>Hello world</message>
  <created>2014-06-01T18:06:36.902</created>
  <author>koushik</author>
</messageEntity>
```

# Les codes d'état

**1xx**: Information. **2xx**: action exécuté avec succès.

**3xx**: Accès impossible, redirection nécessaire.

**4xx**: Requête invalide. **5xx**: Erreur du serveur.

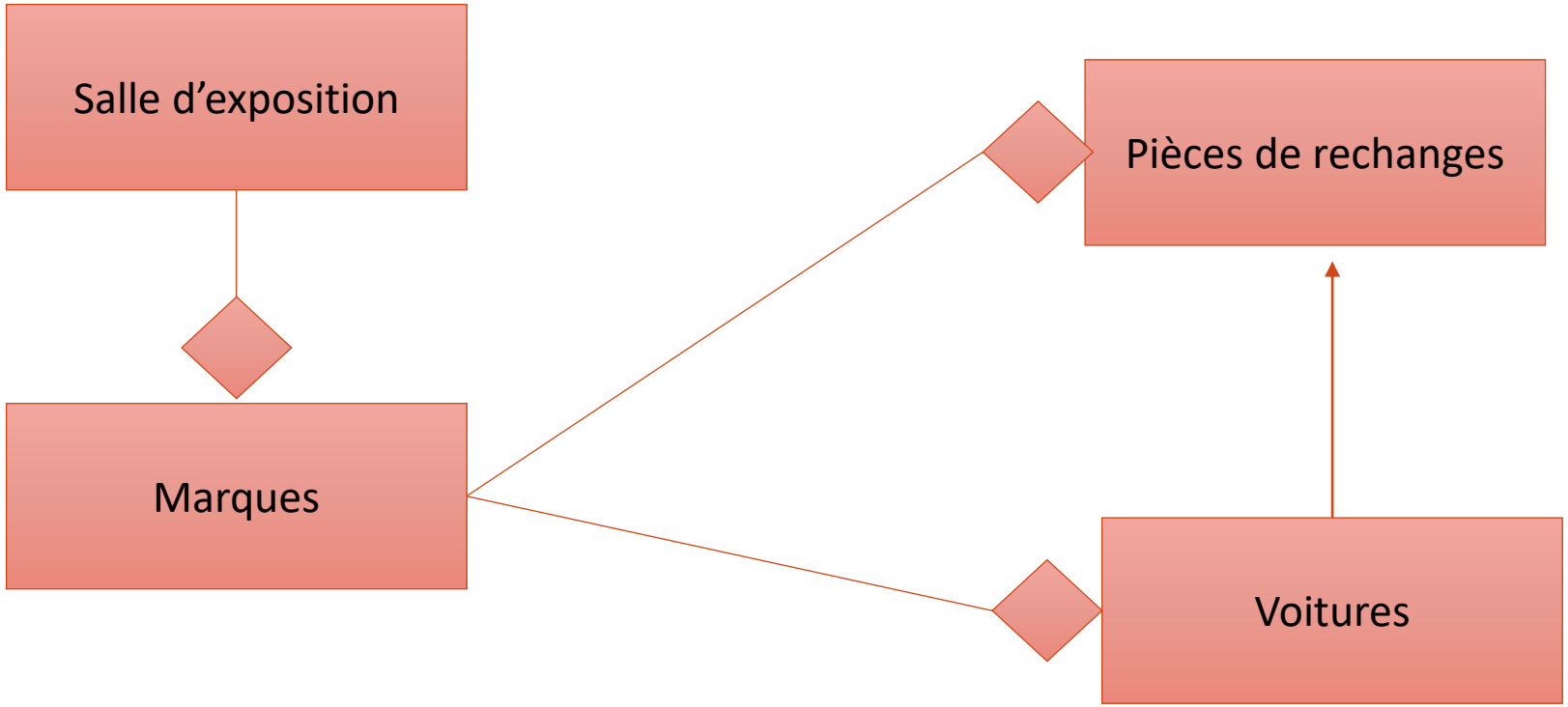
| Operation | Status | Method | Code |
|---|---|---|---|
| **READ** | Successful | **GET** | 200 OK |
| | Not found | | 404 Not Found |
| | Failure | | 500 Internal Server Error |
| **DELETE** | Successful | **DELETE** | 200 OK/204 No Content |
| | Not found | | 404 Not Found |
| atus codes | Failure | | 500 Internal Server Error |

# Les codes d'état

| Operation | Status | Method | Code |
|---|---|---|---|
| **CREATE** | Successful | **POST** | 201 Created |
| | Data error | | 400 Bad Request |
| | Failure | | 500 Internal Server Error |
| **UPDATE** | Successful | **PUT** | 200 |
| | Data error | | 400 Bad Request |
| | Not found | | 404 Not Found |
| | Failure | | 500 Internal Server Error |

tatus codes

# URIs des ressources



# URIs des ressources

- /getMarques
- /getPieces
- /getVoitures
- /getVoitures?marque=audi
- /getPieces?marque=audi
- getPiece?id=15
- /setMarques
- /setVoitures?marque=audi
- /setPiece?id=15

- /Marques
- /Pieces
- /Voitures
- /Voitures/audi
- /Pieces/audi
- /Pieces/15
- /Marques
- /Voitures/audi
- /Pieces/15

# URIs des ressources

- La conventions est d'utiliser des noms au pluriels pour accéder aux ressources.
  - Si l'on veut accéder au voitures exposés dans la salle d'exposition, on pourra utiliser l'URI suivante: /voitures
  - Et pour accéder à la voiture identifiée par l'ID 4, l'on utilise : /voitures/5
- Et d'identifier chaque ressource via un Id à exprimer comme information de chemin
- On pourra aussi exprimer des relations entre les ressources via l'URI:
  - Si l'on veut accéder à la pièce de rechange identifiée par 4 qui concerne la voiture identifiée par l'id 15, alors l'on exprimer cette relation par: /voitures/15/pieces/4.

# Les URIs comme des collections de ressources

- On peut imaginer que chaque URI désigne une ressource ou une collection de ressources:
- Par exemple : /voitures désigne l'ensemble des voitures présents dans la salle d'exposition.
- À ces collections on peut appliquer des filtres ou des contraintes:
  - La liste des voitures dont l'année de fabrication est : 2019: /voitures?modèle=2019 → l'on utilise des query parameters
  - Filtre de pagination : /voitures?offset=30&limit=10
  - Ceci exprime la liste des 10 premières voitures dont l'Id est supérieur ou égale à 10.

# HATEOAS: **Les liens hypermédia**

- Lorsque l'on obtient une ressource, ou une page sur le web, il est très important de la lier à d'autres ressources via des liens.
- En rajoutant ces liens, si vous avez un client assez intelligent, il saura s'adapter aux changements de l'API, ce qui en fait un atout car les APIs doivent évoluer (les ressources peuvent changer de nom et les actions sont amenées à évoluer).
- Référez vous à ce lien pour plus de détails:
  - https://zestedesavoir.com/tutoriels/299/la-theorie-rest-restful-et-hateoas/

# Remarques

- 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

# 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

# Les implémentations JAX-RS

- 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 )
- ESTEasy : fournie par JBOSS
- RESTLET : L'un des premiers framework implémentant REST pour Java

# Développement des webservices REST

- Basé sur POJO (Plain Old Java Object) en utilisant des annotations spécifiques JAX-RS
- Le service est déployé dans une application web
- Approche Bottom/Up
  - Développer et annoter les classes
  - Le WADL est automatiquement généré par l'API
- La spécification JAX-RS dispose d'un ensemble d'annotation permettant d'exposer une classe java dans un services web :
  - @Path
  - @GET, @POST, @PUT, @DELETE
  - @Produces, @Consumes
  - @PathParam

# Les annotations JAX-RS

| | |
|---|---|
| **@Path** | indique un morceau du chemin de l'URI |
| **@GET** | désigne une méthode pour traiter les requêtes GET |
| **@PUT** | désigne une méthode pour traiter les requêtes PUT |
| **@POST** | désigne une méthode pour traiter les requêtes POST |
| **@DELETE** | désigne une méthode pour traiter les requêtes DELETE |
| **@HEAD** | désigne une méthode pour traiter les requêtes HEAD |
| **@HeaderParam** | extrait un paramètre à partir du header http |
| **@PathParam** | extrait un paramètre à partir du chemin |
| **@QueryParam** | extrait un paramètre de l'URI |
| **@Consumes** | type MIME des données acceptées par la méthode |
| **@Produces** | type MIME des données produites par la méthode |
| **@Provider** | indique qu'une classe fournit des méthodes auxiliaires pour JAX-RS |

# Les annotations JAX-RS

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class Hello
{
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello()
  {
    return "Hello Jersey";
  }
}
```

# Les annotations JAX-RS

```
@Path("/hello")
public class Hello
{
  // This method is called if TEXT_PLAIN is request
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello() {
    return "Hello Jersey";
  }

  // This method is called if XML is request
  @GET
  @Produces(MediaType.TEXT_XML)
  public String sayXMLHello() {
    return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
  }

  // This method is called if HTML is request
  @GET
  @Produces(MediaType.TEXT_HTML)
  public String sayHtmlHello() {
    return "<html> " + "<title>" + "Hello Jersey" + "</title>"
        + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
  }
}
```

# JAXB

- JAXB - Java Architecture for XML Binding
  - Sérialiseur/parseur XML en Java
  - Entièrement généré par annotations de code
  - Conversion XSD <-> code Java annoté
  - Permet la sérialisation désérialisation JSON (MediaType explicite)

```
import javax.xml.bind.annotation.*;

@XmlRootElement(name="hello")
public class HelloObject
{
  public HelloObject(int a)
  {
    id = a;
  }
  @XmlElement(name="id")
  int id;
}
```

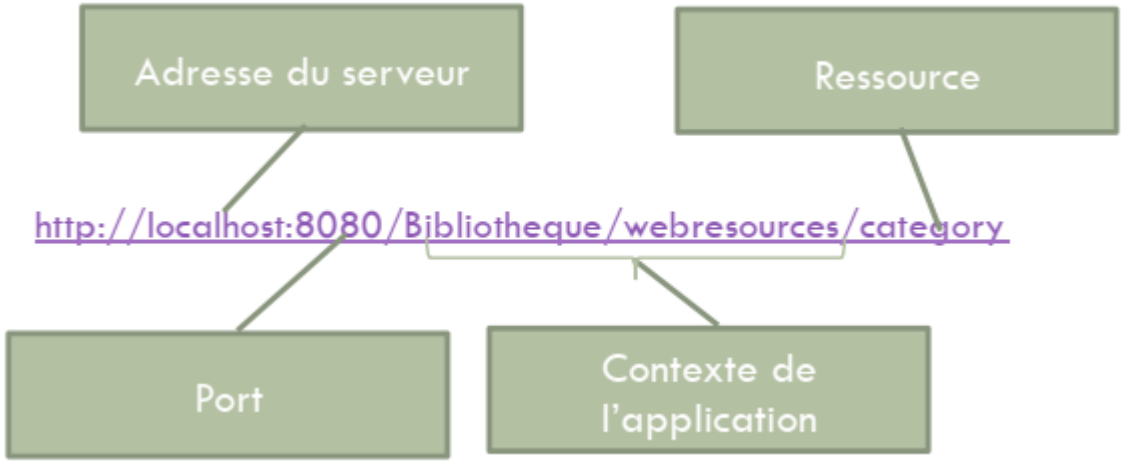
# Modéliser les URIs

- A l'utilisateur d'appliquer les conventions
- Si on peut utiliser une URI de ressource, c'est que c'est une ressource !
  - À ne pas faire : GET http://MyService/Persons?id=1
  - Préférer : GET http://MyService/Persons/1
- GET ne fait pas d'effet de bord
  - À ne pas faire : GET http://MyService/DeletePerson/1
  - Préférer : DELETE http://MyService/Persons/1
- PUT est idempotent
  - Appeler la requête plusieurs fois à le même effet qu'une seule fois
- Pas d'état dans le serveur
  - À ne pas faire : GET http://MyService/Persons/1 HTTP/1.1
    GET http://MyService/NextPerson HTTP/1.1
  - Préférer : liste de liens, **itérateur côté client**

# Modéliser les URIs

- URIs sont déterminés par l'annotation @Path
- Permet d'exposer une classe dans le WS
- Définit la racine des ressources (Root Racine Ressources)
- Sa valeur correspond à l'URI relative de la ressource

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



# Modéliser les URIs

- @Path peut être utilisée pour annoter des méthodes d'une classe
- L'URI résultante est la concaténation entre le valeur de @Path de la classe et celle de la méthode

```
@Path("category")
public class CategoryFacade {
@GET
@Produces({MediaType.APPLICATION_XML,
MediaType.APPLICATION_JSON})
@Path("test")
public String hello()
{
return "Hello World!";
}
..
}
```

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

# URIs 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.

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

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

# URIs dynamiques

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

GET http://localhost:8080/Bibliotheque/webresources/category/coucou/Miage/NTDP

```
  @GET
    @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
    @Path("{id}")
    public Categorie find (@PathParam("id") Long id) {
        return super.find(id);

    }
```

GET http://localhost:8080/Bibliotheque/webresources/category/1

# @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

# Outils de test

- Il existe de nombreux outils en ligne permettant de tester les services Web REST :
- Certains sont disponibles sous forme d'extsnsion que vous pouvez installer dans les navigateurs:
  - RestConsole
  - PostMan
  - SOAPUI