SOA
Exemple d'implémentation avec REST et les services web Restful

Rappel des Principes fondamentaux de l'architecture SOA

Les principes à respecter :

- Discussion entre métier et IT
- Utilisation des use case métier

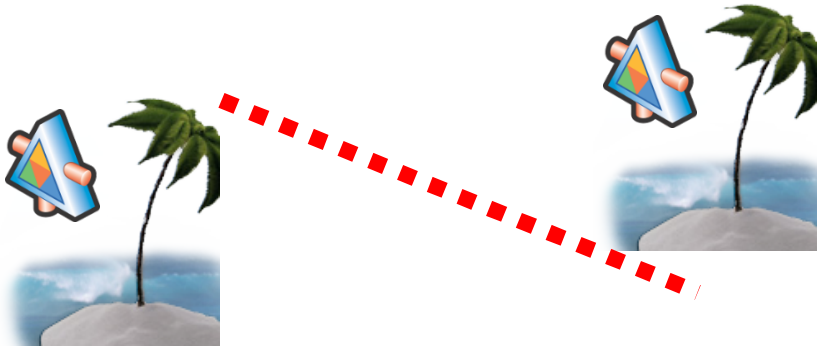
- Utilisation de standards
- Pas de remise en cause de l'existant lors d'évolutions technologiques

- Découplage entre fournisseur et consommateur de services
- Indépendance des ressources vis à vis de ceux qui les utilisent

4 propriétés du service à retenir

- **Un Service est <u>Autonome</u> et sans <u>état</u>**

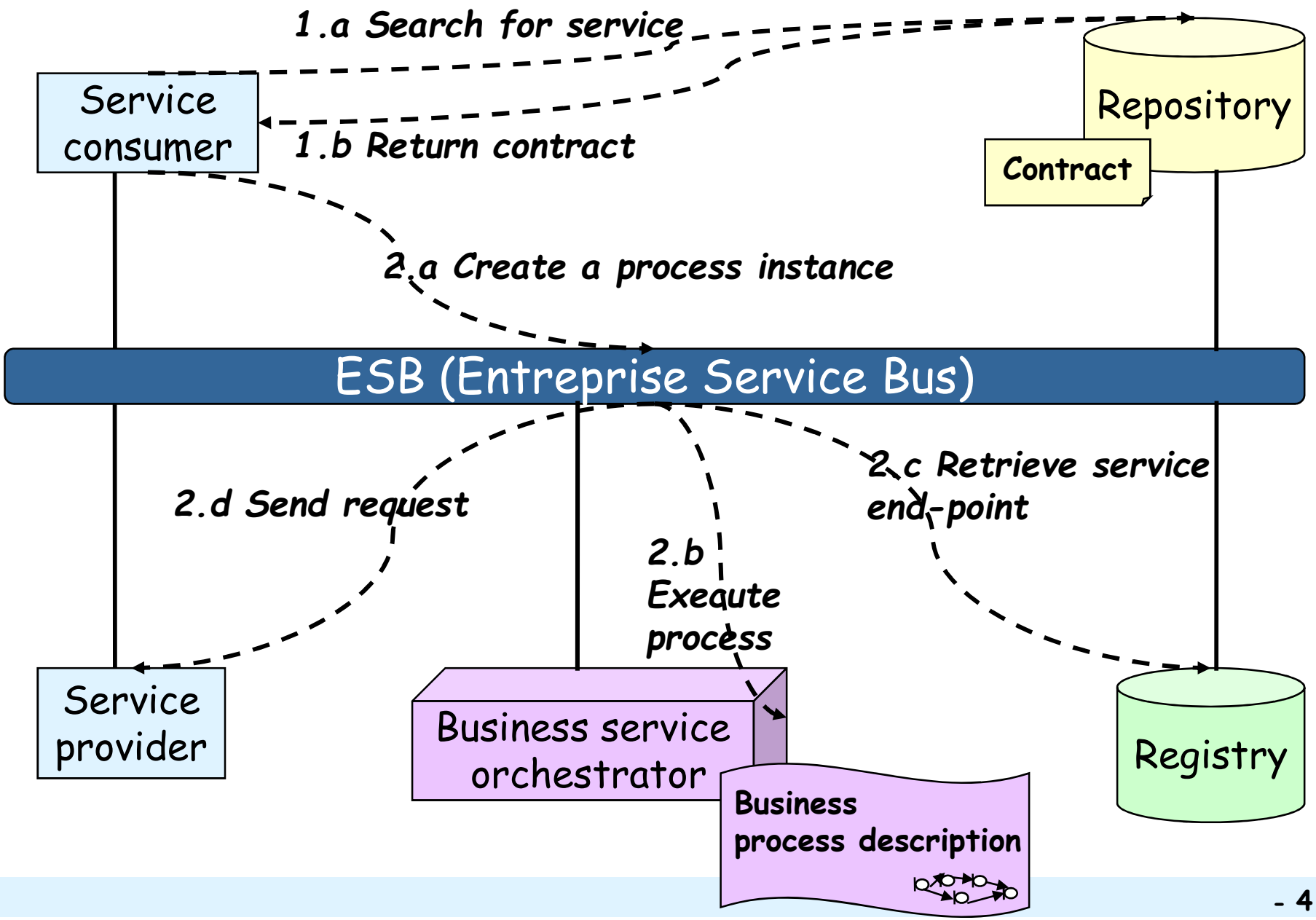
- **Les Frontières entre services sont <u>Explicites</u>**



- **Un Service expose un <u>Contrat</u>**

- **Les services communiquent par messages**

Points clés de l'architecture



Standards de l'architecture

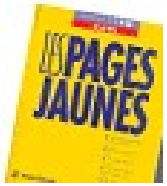
Les standards sont un élément clé d'une SOA, ils assurent l'interopérabilité

SOAP
W3C
Simple Object Access Protocol

Transporte

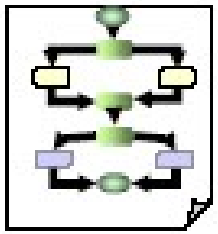
WSDL
W3C
Web Services Description Language

Décrit le contrat



UDDI
Microsoft, IBM, HP
Universal Description Discovery and Integration

Spec pour Repository/Registry



BPEL
Oasis
Business Process Execution Language

Décrit les processus métier

Les trois piliers des Services Web

REST
Style d'architecture

SOA et web services

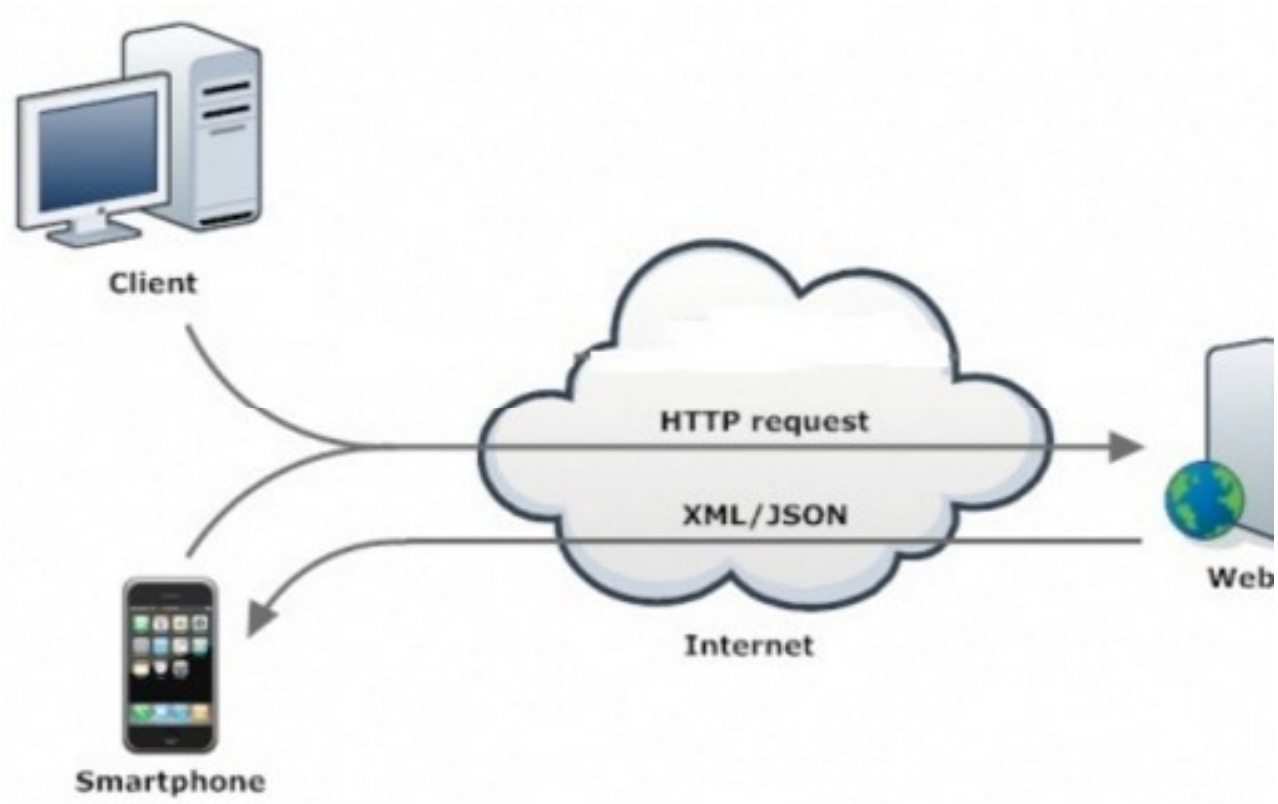
- Attention à ne pas confondre les 2 !
  - SOA est un ensemble de concepts :
    Une SOA peut se mettre en œuvre sans Web Services
  - Les WS sont de l'ordre de la technologie :
    On peut utiliser les Web Services sans faire de SOA
- Les WS constituent la meilleure solution standardisée disponible
  - Un service métier = un webservice

REST et les Services Web Restful

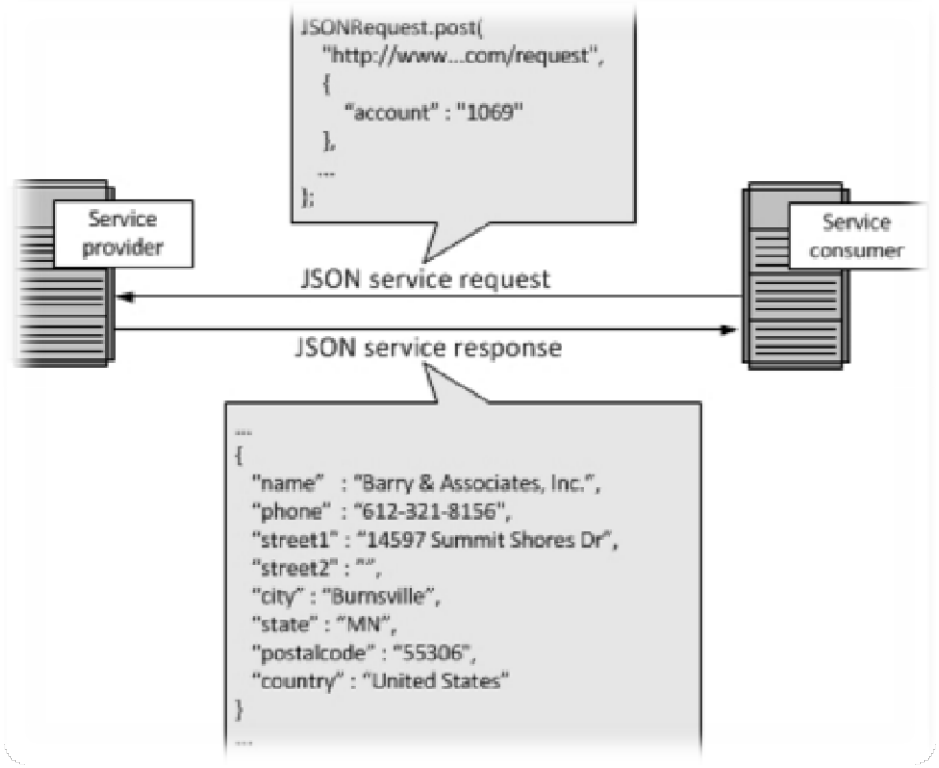
C'est quoi Rest?

- REST c'est un style d'architecture (introduit par Roy Fielding en 2000) ;
- REST utilise HTTP dans le protocole de transport ;
- REST se base sur des ressources où chaque composant est une ressource et une ressource est accessible par une interface commune en utilisant des méthodes HTTP standards (GET, POST, PUT et DELETE) ;
- Le serveur REST fournit l'accès aux ressources et le client REST accède et présente les ressources ;
- Une ressource est identifiée par une URI (Uniform Resource Identifier) ;
- REST utilise diverses représentations pour représenter une ressource comme du texte, JSON et XML (**JSON est le format le plus populaire utilisé dans les services Web**)

L'architecture REST



L'architecture REST



Méthodes HTTP

- **GET :** Fournit un accès en lecture seule à une ressource.
- **PUT :** Utilisé pour créer une nouvelle ressource.
- **DELETE :** Utilisé pour supprimer une ressource.
- **POST :** Utilisé pour mettre à jour une ressource existante ou créer une nouvelle ressource.
- **OPTIONS :** Utilisé pour obtenir les opérations supportées par une ressource.

Les services web Restful

- Les services Web basés sur l'architecture REST sont connus comme les services Web RESTful.

- Ces services Web utilisent des méthodes HTTP pour mettre en œuvre le concept de l'architecture REST.

- L'API JAX-RS 2.0 permet la création des services web Restful.

JAX-RS 2.0: Les implémentations

- ✓ Jersey (L'implémentation de référence de SUN) ;
  - https://jersey.java.net ;
  - Dernière version : 2.23.1 (Juin 2016).

- ✓ CXF (Framework développé par Apache) :
  - http://cxf.apache.org ;
  - Dernière version : 3.1.6 (Mars 2016).

- ✓ RESTEasy (Framework de Jboss) :
  - http://resteasy.jboss.org ;
  - Dernière version : 3.0.17 (Juin 2016).

- ✓ Restlet :
  - https://restlet.com/ ;
  - Dernière version : 2.3.7 (Mars 2016).

Jersey: Configuration de la Servlet

```
<servlet>
      <servlet-name>Jersey RESTful Application</servlet-name>
      <servlet-class>org.glassfish.jersey.servlet.ServletContainer<
      <init-param>
          <param-name>jersey.config.server.provider.packages</param
          <param-value>rest.service</param-value>
      </init-param>
  </servlet>
  <servlet-mapping>
      <servlet-name>Jersey RESTful Application</servlet-name>
```

Jersey: Méthode GET (1/5)

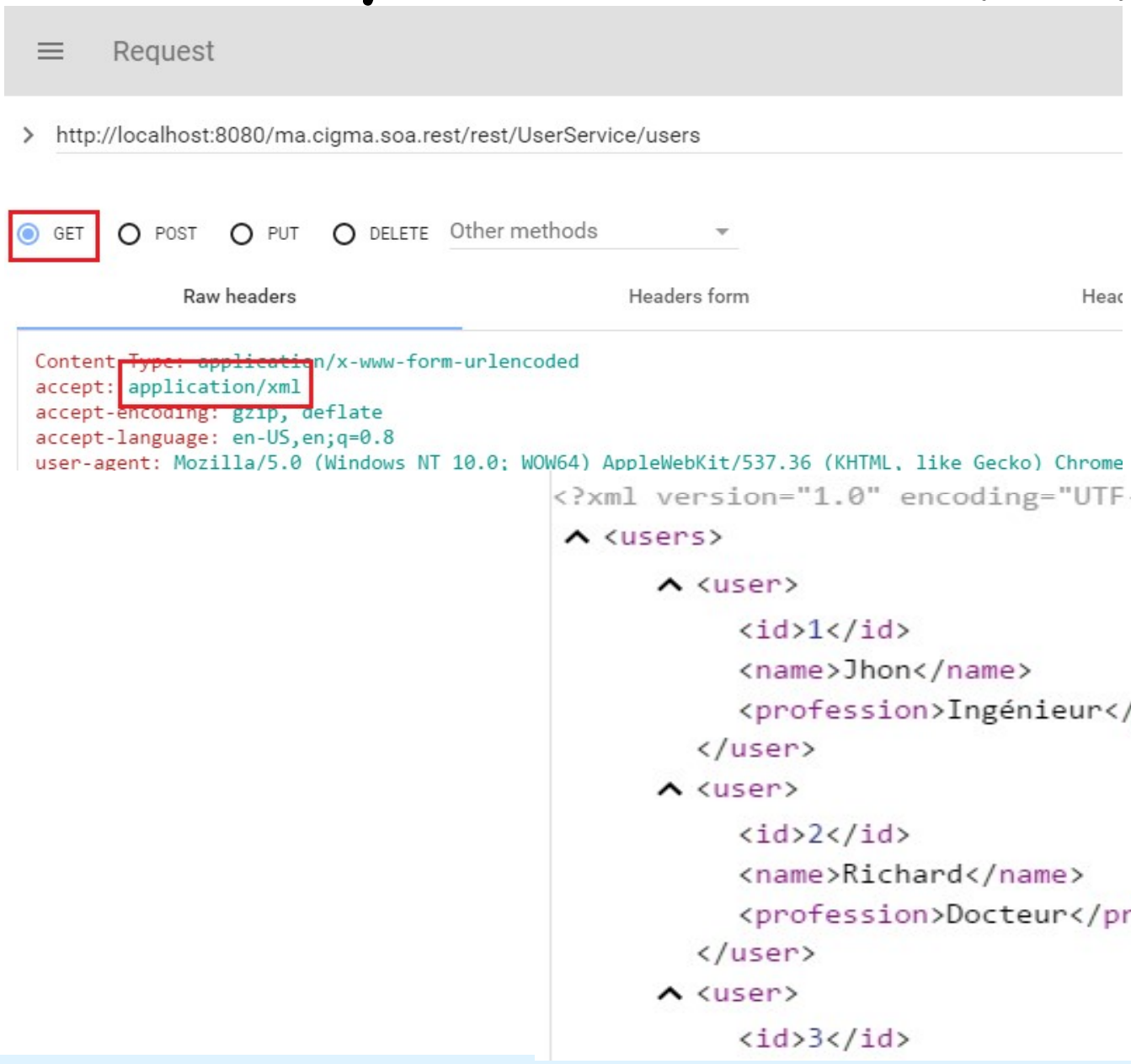
```java
@Path("/UserService")
public class UserService {

    IUserDao userDao = new UserDaoImpl();
    private static final String SUCCESS_RESULT = "<result>success</result>";
    private static final String FAILURE_RESULT = "<result>failure</result>";

    @GET
    @Path("/users")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public List<User> getUsers() {
        return userDao.getAllUsers();
    }
}
```

- L'exécution du lien :

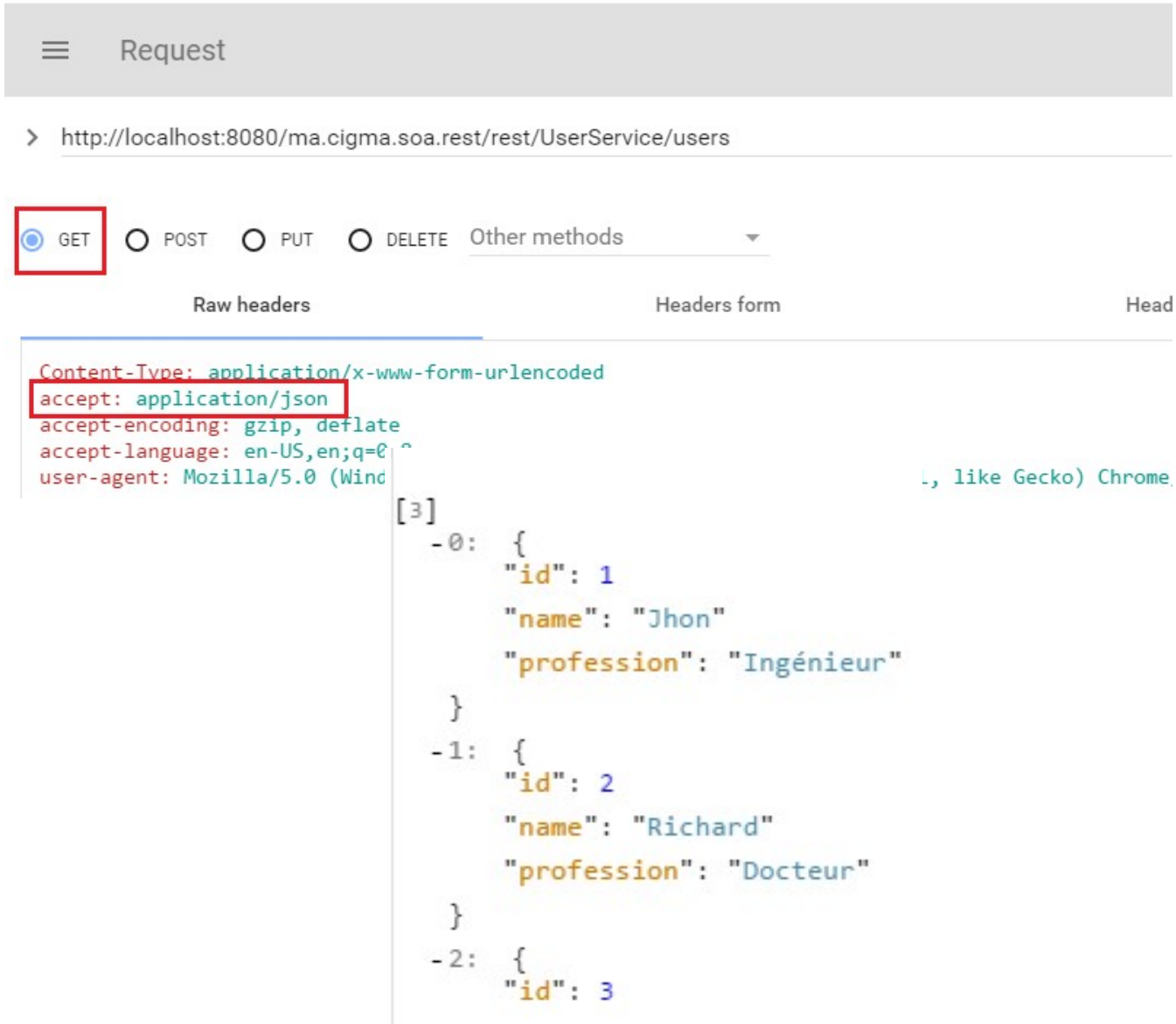
  http://localhost:8080/ma.cigma.soa.rest/rest/UserService/users

donnera comme résultat la liste des utilisateurs en format XML ou bien en format Json selon la requête.

Jersey: Méthode GET (2/5)



Jersey: Méthode GET (3/5)



Jersey: Méthode GET (4/5)

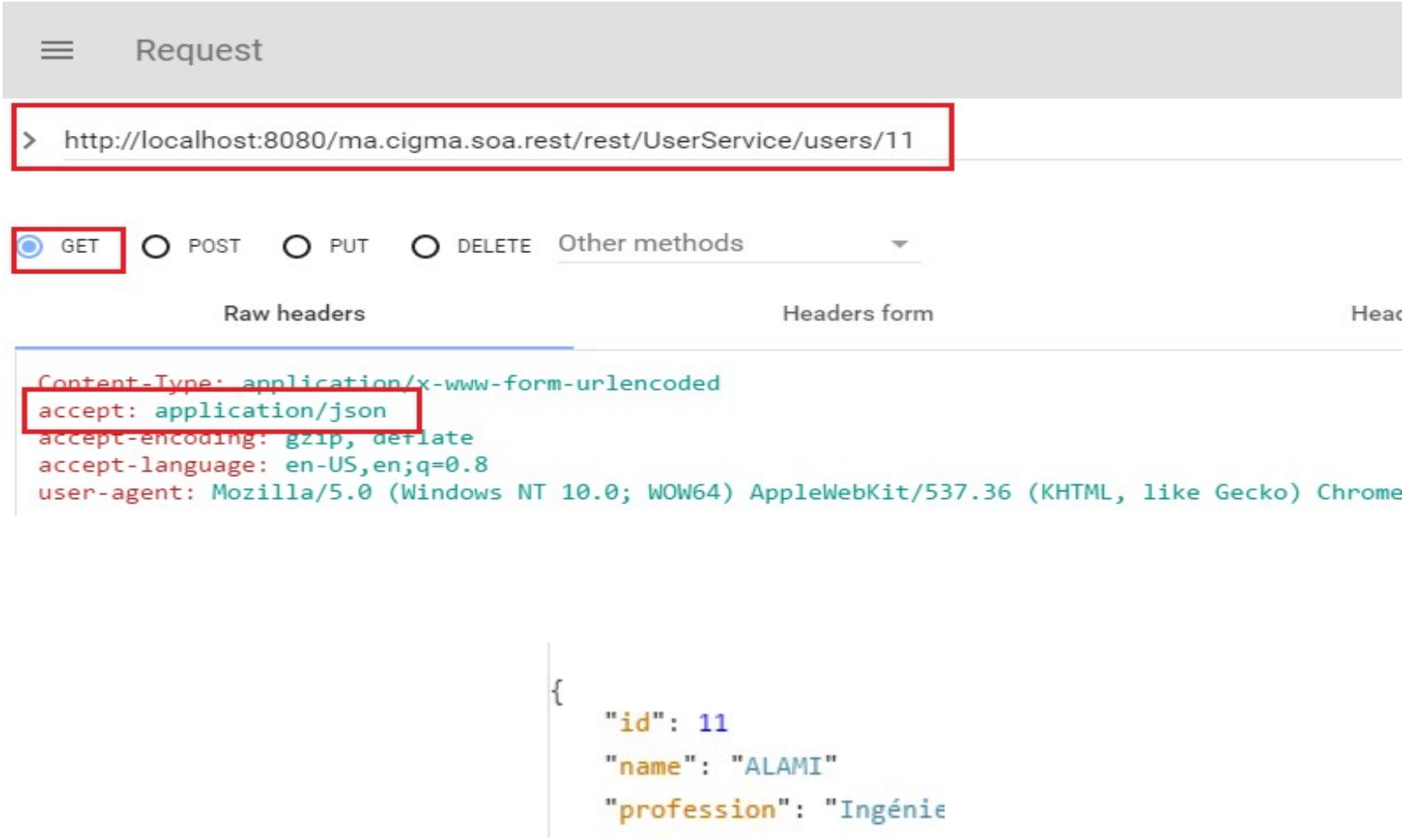
```
@GET
@Path("/users/{userid}")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATI
public User getUser(@PathParam("userid") int userid) {
    return userDao.getUser(userid);
```

- L'exécution du lien :

http://localhost:8080/ma.cigma.soa.rest/rest/UserService/users/1

donnera comme résultat l'utilisateur ayant l'id 1 en format Json ou bien en format XML selon la demande du client.

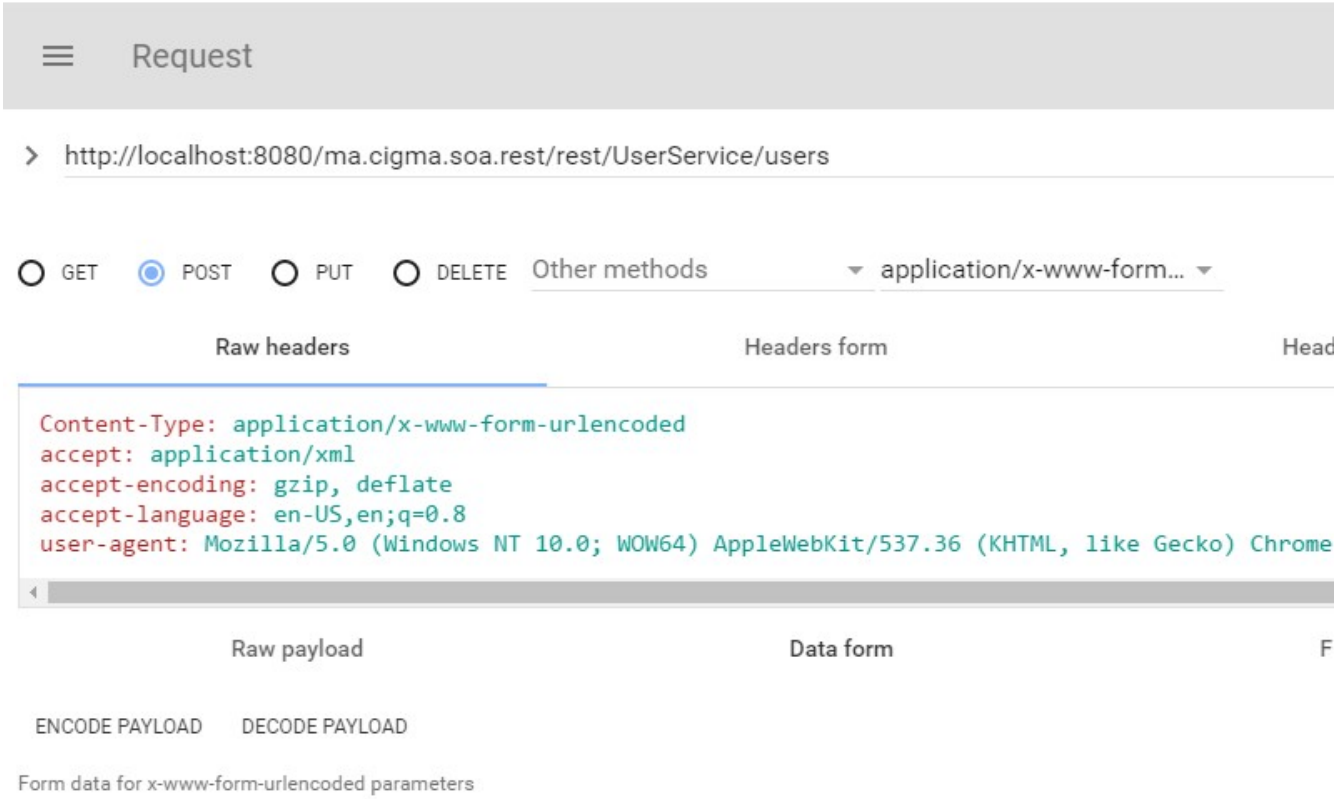
Jersey: Méthode GET (5/5)



Jersey : Méthode POST (1/2)

```
@POST
@Path("/users")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON,MediaType.TEXT_P
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public String updateUser(@FormParam("id") int id, @FormParam("name") String name,
        @FormParam("profession") String profession, @Context HttpServletResponse s
                throws IOException {
    User user = new User(id, name, profession);
    int result = userDao.updateUser(user);
    if (result == 1) {
        return SUCCESS_RESULT;
    }
```

Jersey : Méthode POST (2/2)



<result>success</resu

Jersey : L'api Client (Méthode GET)

```java
package test;

import java.util.List;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.core.GenericType;
import javax.ws.rs.core.MediaType;
import rest.modele.User;
public class Test2 {
    private static String REST_SERVICE_URL = "http://localhost:8080/ma.cigma.soa.rest/rest/U
    private static final String PASS = "pass";
    private static final String FAIL = "fail";
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        GenericType<List<User>> list = new GenericType<List<User>>() {};
        List<User> users = client.target(REST_SERVICE_URL).request(MediaType.APPLICATION_XML
        String result = PASS;
        if (users.isEmpty()) {
            result = FAIL;
```

Jersey : L'api Client (Méthode DELETE)

```
package test;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.core.MediaType;
public class Test2 {
    private static String REST_SERVICE_URL = "http://localhost:8080/ma.cigma.soa.rest/rest
    private static final String PASS = "pass";
    private static final String FAIL = "fail";
    private static final String SUCCESS_RESULT = "<result>success</result>";
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        String callResult = client.target(REST_SERVICE_URL).path("/{userid}").resolveTempl
                .request(MediaType.APPLICATION_XML).delete(String.class);
        String result = PASS;
        if (!SUCCESS_RESULT.equals(callResult)) {
            result = FAIL;
```

Jersey : Exemple complet

Réaliser le TP n° 1 :

➔ Développer une application web permettant la gestion des CRUD (Create, Read, Update et Delete) en utilisant Jeresy et en respectant les principes SOA.