Chapitre 4 :
Les Services Web REST

hamdi.aloulou@isima.u-monastir.tn

Introduction

Web Service REST

Définition

- Acronyme de **RE**presentational **S**tate **T**ransfert
- REST est 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

REST ➔ Caractéristiques

› 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)

› Orienté ***Ressources,*** uniquement ***identifiées par des URI(s)***

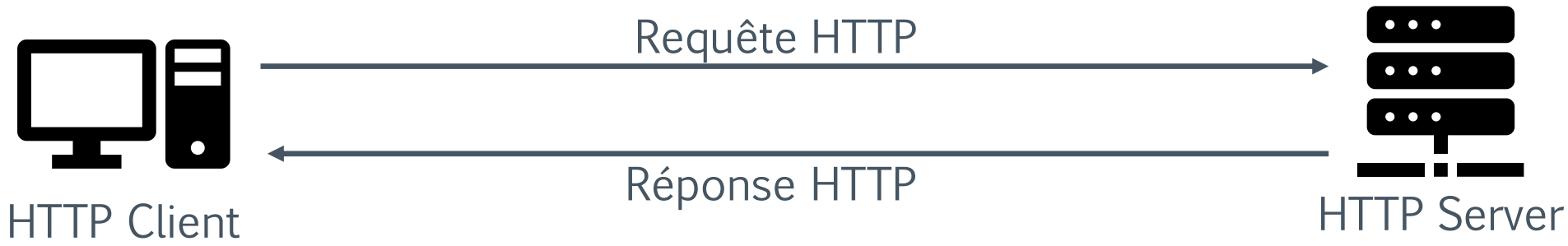
Rappel Protocole HTTP

Protocole HTTP

› HyperText Transfer Protocol

› Protocole d'échanges d'information sur le web

› Basé sur TCP/IP



Enchainement Client - Serveur

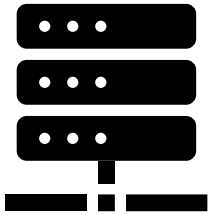
(1) L'utilisateur lance une URI à travers le navigateur http://host:port/path/file

Client (Browser)

(2) Le navigateur envoi un message request

GET URI HTTP/1.1
URI:
http://host:port/path/file
……..

(3) Le serveur mappe l'URL vers un fichier ou le répertoire du document



Server (@host:port)

(4) Le serveur envoi un message response

HTTP/1.1 200 OK
……..
file
……..

(5) Le navigateur formatte la réponse et l'affiche

HTTP (over TCP/IP)

URI

- Unique Resource Identificator
- 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)

URI :

Protocole://Host:Port/Path

Méthodes HTTP

- HTTP définit un ensemble de méthode permettant de caractériser les requêtes
  - GET : Récupérer des ressources d'un serveur
  - POST : Envoyer des données à un serveur
  - PUT : Modifier des données
  - DELETE : Suppression de données
- Autres : HEAD, TRACE, CONNECT

Requêtes HTTP

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

Requêtes HTTP

- Request Line

```
GET http://localhost:8080/bibliotheque/index.html HTTP/1.1
```

- Request Headers

```
Host: localhost:8080
Content-Length: 176
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,
likeGecko)
Content-Type: application/x-www-form-urlencoded
```

- Request Body

```
nom: licence
description: LCE
```

Réponse HTTP

- Réponse du serveur au client
- Format d'une réponse HTTP
  - **Response Header**
    - Response Line
    - Response Headers
  - **Response Body** [Optional]

Réponses HTTP

- Response Line

```
HTTP/1.1 200 OK
```

- Response Headers

```
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
```

- Response Body

```
<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>
```

Invocation de Services Web REST

REST et HTTP

- Des **ressources** Identifiée par des URIs (https://jsonplaceholder.typicode.com/users)
- **Des actions sur les ressources** correspondant aux types de requêtes HTTP : GET, POST, PUT, DELETE
- Ces actions permettent de **manipuler les ressources**
- **Format d'échanges** entre le client et le serveur (XML, JSON, text/plain,…)

Ressources

- Une ressource est identifiée par une URI : Une URI identifie de façon unique une ressource sur le Web

  https://jsonplaceholder.typicode.com/users/2

  Clef primaire de la ressource dans la BD

- Une ressource est un objet identifiable sur le Web

  ➔ Post, Comment, Client, User

- Une ressource n'est pas forcément une entité physique, elle peut être virtuelle (Comment, Post ...)

Action sur les ressources

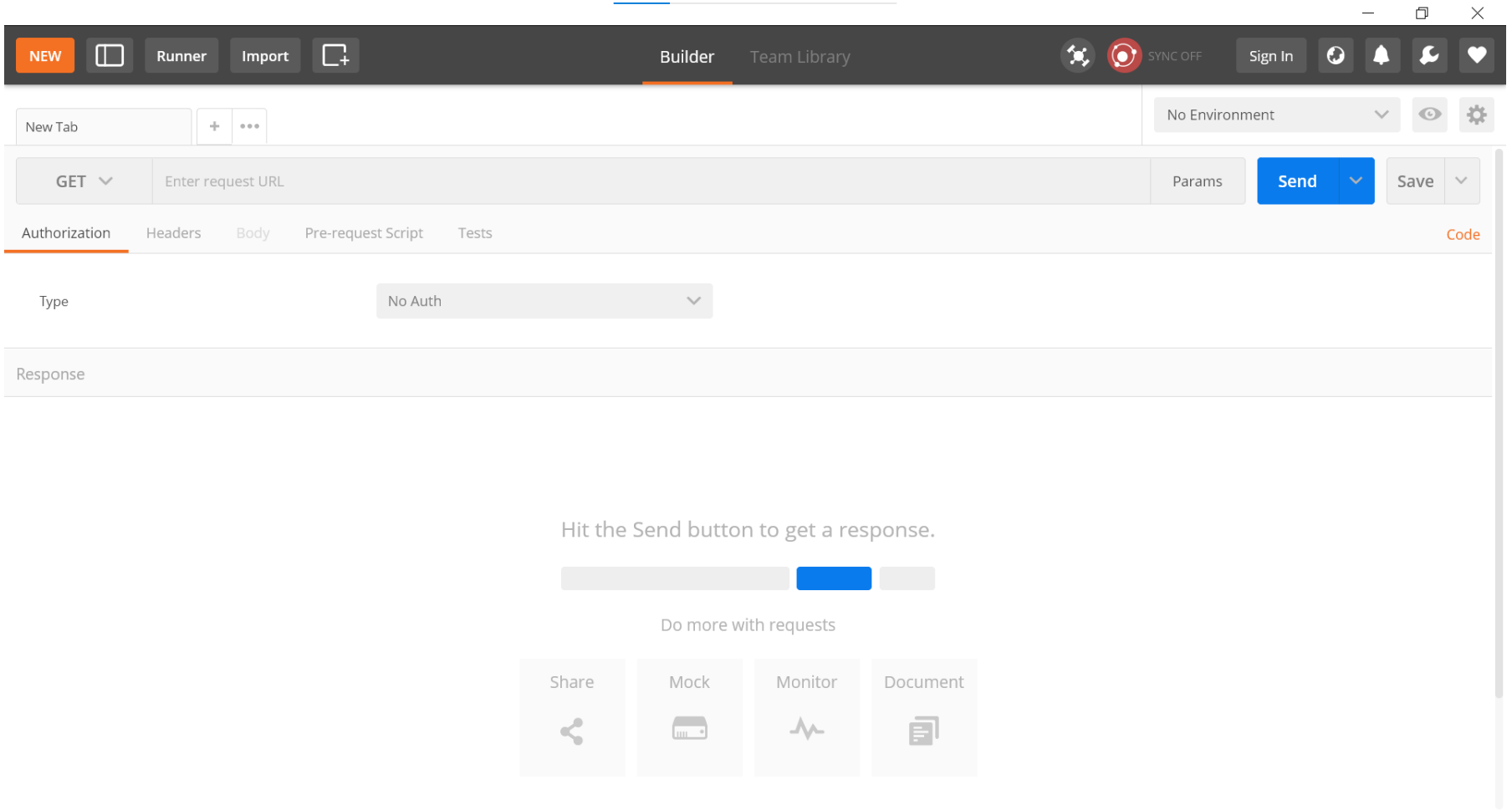
- Une ressource peut subir quatre opérations de bases CRUD correspondant aux quatre principaux types de requêtes HTTP (GET, PUT, POST, DELETE)
- REST s'appuie sur le protocole HTTP pour effectuer ces opérations sur les objets
  - POST ➔ Créer une nouvelle ressource (Ajout de données)
  - GET ➔ Récupérer une ressource sans la modifier
  - PUT ➔ Mettre à jour une ressource identifié par l'URI
  - DELETE ➔ Supprimer la ressource identifié par l'URI

Exemple d'invocation d'un service web REST

- JSONPlaceholder est une API REST gratuite en ligne pour tester des requêtes d'invocation de services web REST.
- Disponible à l'adresse : https://jsonplaceholder.typicode.com/
- JSONPlaceholder fourni un ensemble de 6 ressources :
  - /posts (100 posts)
  - /comments (500 comments)
  - /albums (100 albums)
  - /photos (5000 photos)
  - /todos (200 todos)
  - /users (10 users)

Exemple d'invocation d'un service web REST

› L'outil PostMan permet de tester des services REST



Les étapes d'échange (1/3)

1. **Client – Génération d'un message**
   - Définition de l'action et donc de la méthode HTTP à utiliser (GET / POST / PUT / DELETE)
   - Définition de la ressource demandée

GET https://jsonplaceholder.typicode.com/users/2

Les étapes d'échange (2/3)

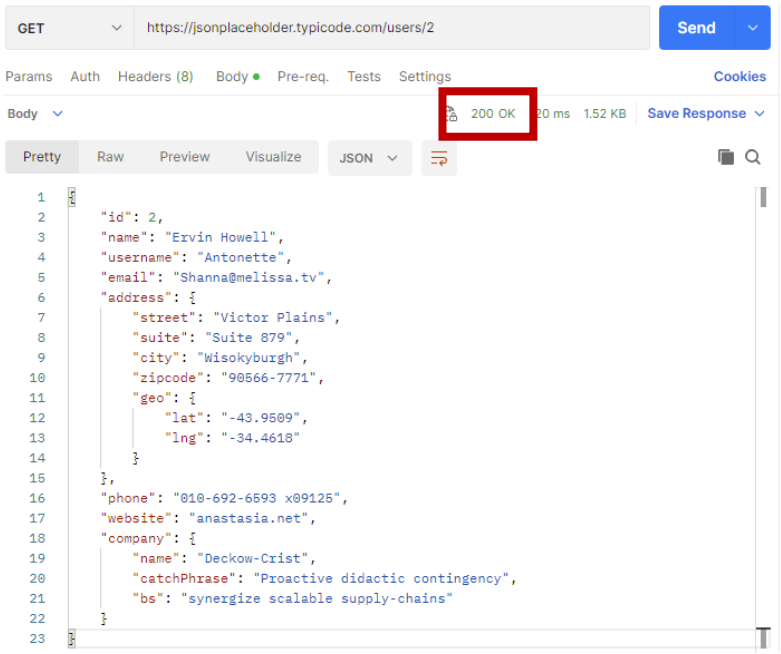
2. Serveur – Réception et Réponse

- Identification de la méthode HTTP utilisée
- Identification de la ressource demandée
- Résolution de l'action
- Génération de la réponse (Représentation de la ressource, Code d'état)

Les étapes d'échange (3/3)

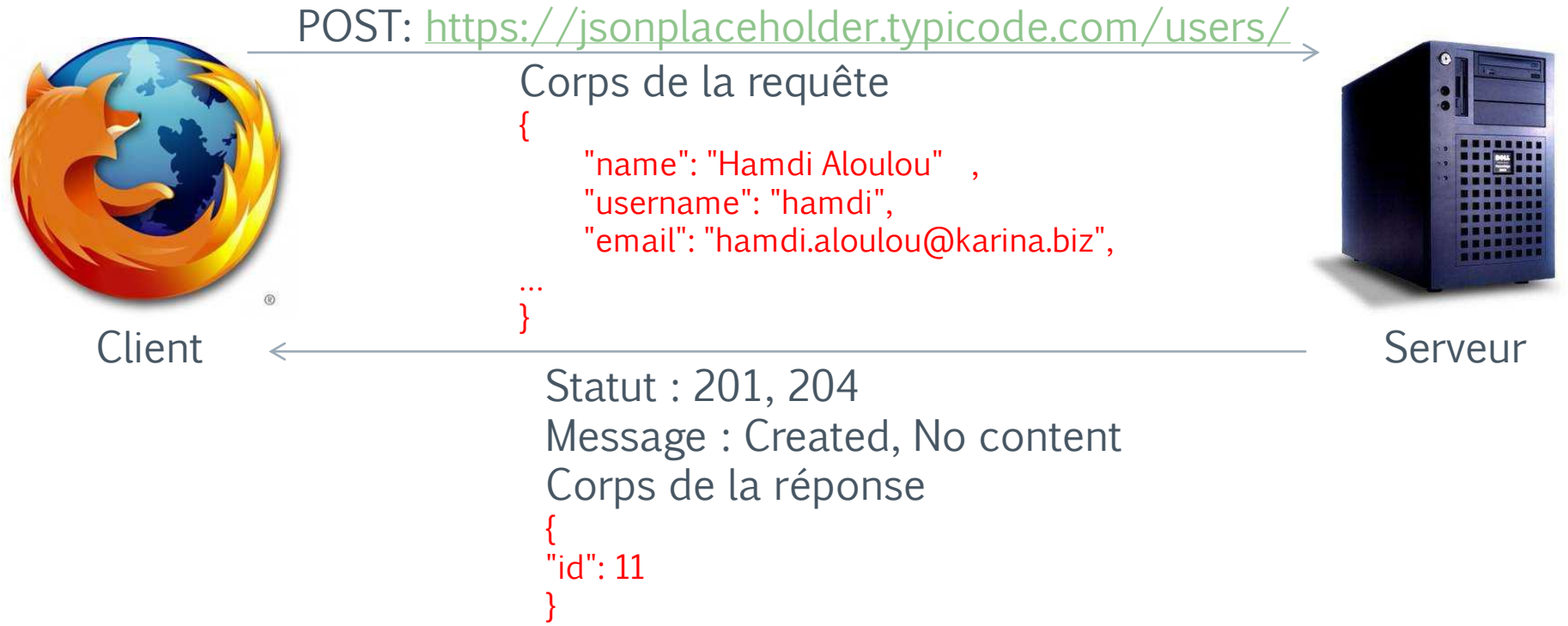
3. Client – Lecture de la réponse

- Décodage du code d'état (code résultat, code erreur)
- Si positif (200/201) : validation (création, modification, suppression, consultation)
- Si négatif (4XX/5XX) : Information sur la raison de l'échec



Méthode POST

› La méthode POST crée une nouvelle ressource sur le système



Méthode GET

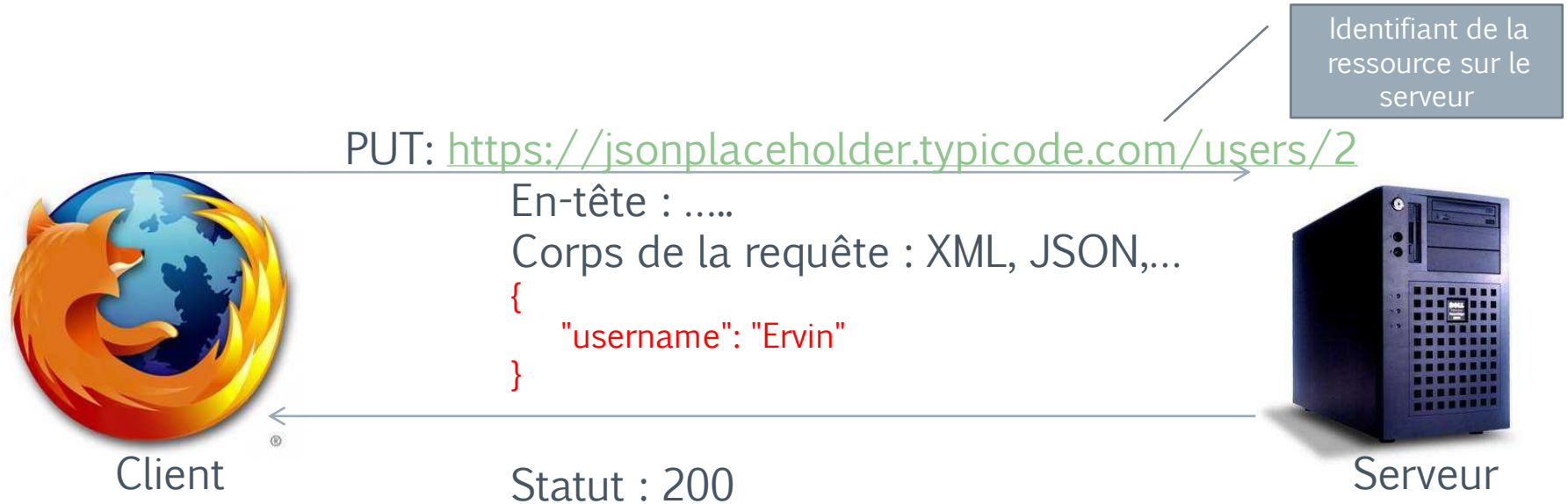
› La méthode GET renvoie une représentation de la ressource tel qu'elle est sur le système

Méthode GET

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

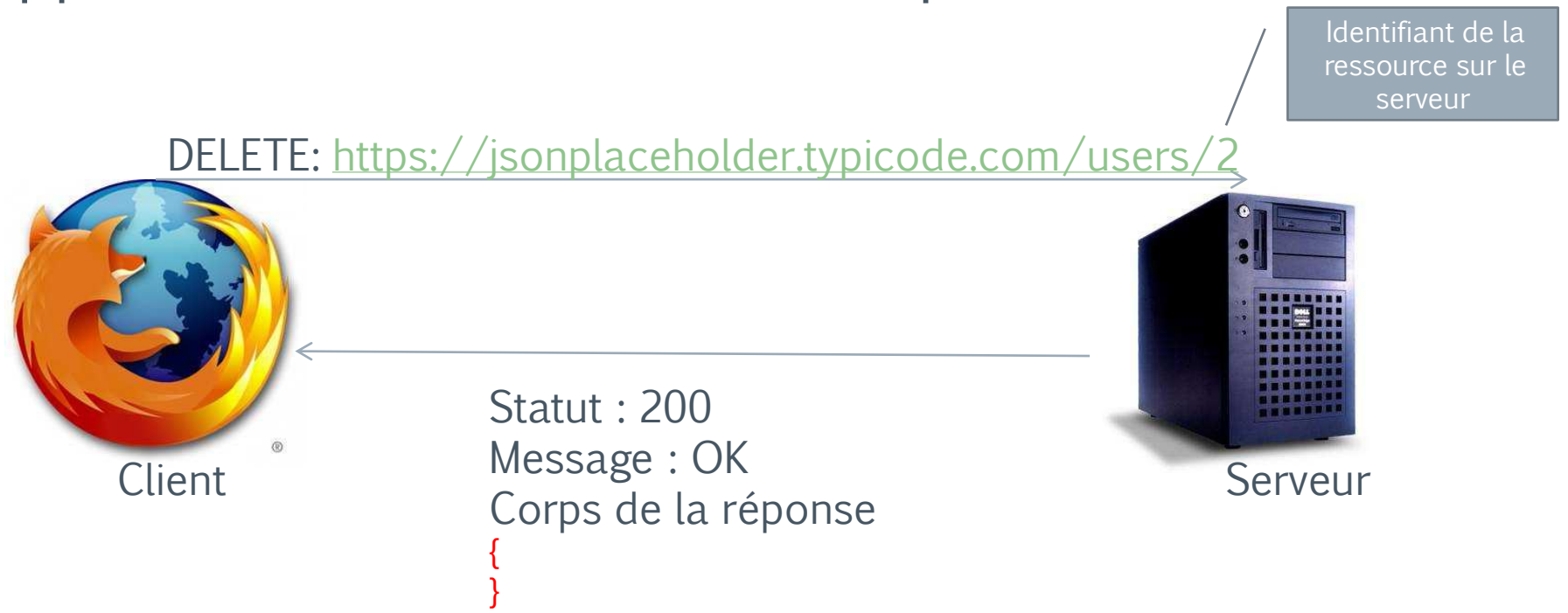
Méthode PUT

› Mise à jour de la ressource sur le système



Méthode DELETE

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



REST, Méthodes 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 méthode HTTP utilisée**

Exemple :

GET
PUT
DELETE
→ https://jsonplaceholder.typicode.com/users/2

GET
POST
→ https://jsonplaceholder.typicode.com/users

REST, Méthodes HTTP et Ressources

› Crée une nouvelle personne ➔

POST http://exemple.com/rest/person

› Liste des personnes à ➔

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

Format des données

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

Les données échangées entre le client et le serveur pour réaliser ces actions peuvent être sous différent format selon l'API REST développée / utilisée :

- XML
- JSON
- Text/plain
- ...

Le format JSON

JSON

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

Deux structures :

› Une collection de clefs/valeurs ➔ Object

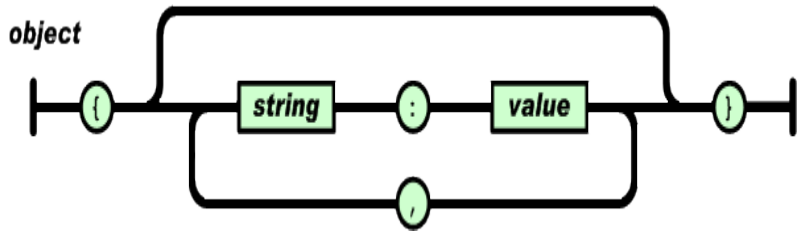
```
{"nom": "Ahmed"}
```

› Une collection ordonnée d'objets ➔ Array

```
[{"nom": "Ahmed"}, {"nom": "Sana"}, {"nom": "Walid"}]
```

JSON Object

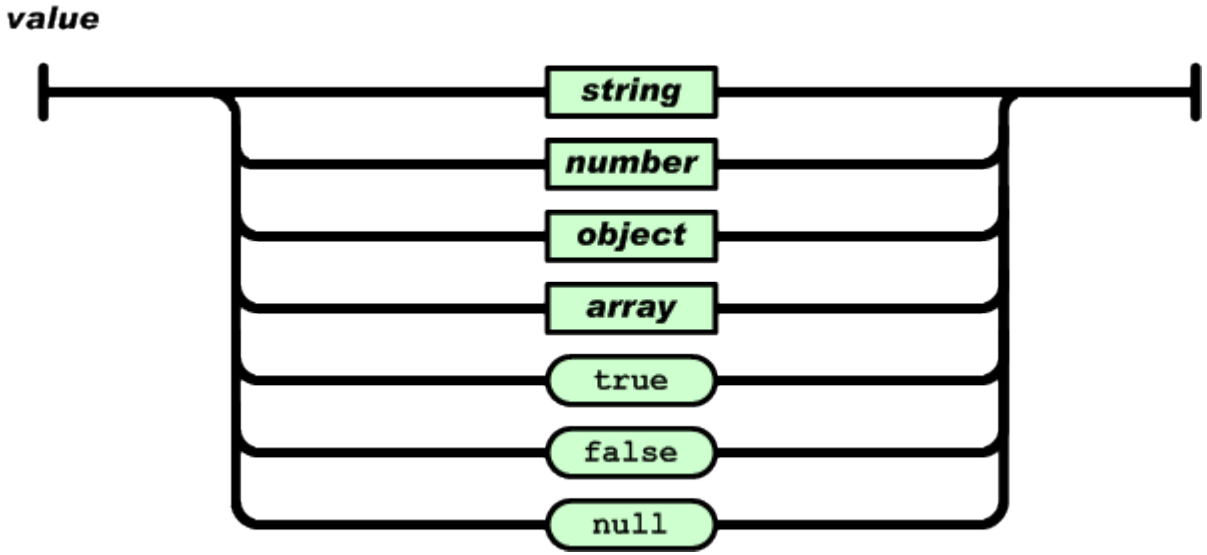
Commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paire clefs/valeurs. Une clef est suivie de « : » et les paires clef/valeur sont séparés par « , »




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

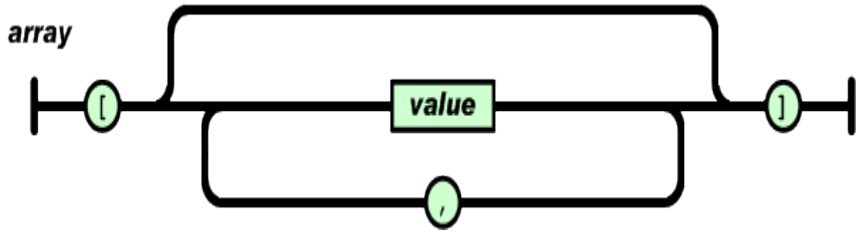
JSON Value

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



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": ""
}
]
```

Exemple de document JSON

```
{
    "titre": "The Social network",
    "résumé": "On a fall night in 2003, Harvard undergrad and programming
              genius Mark Zuckerberg sits down at his computer and
              heatedly begins working on a new idea.(...)",
    "année": 2010,
    "réalisateur": {"nom": "Fincher", "prénom": "David"},
    "acteurs": [{"prénom": "Jesse", "nom": "Eisenberg"},
              {"prénom": "Rooney", "nom": "Mara"}]
}
```

Équivalant en XML

```
<film>
    <titre>The Social network</titre>
    <résumé>On a fall night in 2003, Harvard undergrad and programming genius Mark
     Zuckerberg sits down at his computer and heatedly begins working on a new idea.</résumé>
    <année>2010</année>
    <réalisateur>
        <nom>Fincher</nom>
        <prénom>David</prénom>
    </réalisateur>
    <acteur>
        <prénom>Jesse</prénom>
        <nom>Eisenberg</nom>
    </acteur>
    <acteur>
        <prénom>Rooney</prénom>
        <nom>Mara</nom>
    </acteur>
</film>
```

Pour s'exercer

- Comment savoir qu'un document JSON est bien formé (c'est-à-dire syntaxiquement correct)? Il existe des validateurs en ligne, bien utiles pour détecter les fautes.
- Essayez par exemple http://jsonlint.com/: copiez-collez les documents JSON donnés précédemment dans le validateur et vérifiez qu'ils sont correct (ou pas...).
- Le document suivant contient (beaucoup) d'erreurs, à vous de les corriger. Cherchez-les visuellement, puis aidez-vous du validateur.

Pour s’exercer

```
{  "titre": "Taxi driver",
"année": 1976,
"genre": "drama",
"résumé": 'Vétéran de la Guerre du Vietnam,
Travis Bickle est chauffeur de taxi dans la ville
de New York.  La violence quotidienne l'affecte
peu à peu.’,
"pays": "USA",
"réalisateur": {
"nom": "Scorcese",
prénom: "Martin",
"date_naissance": "1962"
},
"acteurs": [
{nom: "Jodie",
"prénom": "Foster",
"date_naissance": null,
"rôle": "1962" }
{nom: "Robert",
"prénom": "De Niro",
"date_naissance": "1943",
"rôle": "Travis Bickle ", }
}
```

Exercice

- Installer le plugin PostMan (si vous ne l'avez pas déjà)
- Effectuer les opérations de base (CRUD) sur les ressources identifiées par :
  - https://jsonplaceholder.typicode.com/users
  - https://jsonplaceholder.typicode.com/posts
  - https://jsonplaceholder.typicode.com/comments
  - https://jsonplaceholder.typicode.com/albums
  - https://jsonplaceholder.typicode.com/photos
  - https://jsonplaceholder.typicode.com/todos

Développer des Web Services REST avec JAVA

JAX-RS

- Acronyme de Java API for RestFul Web Services
- Il fait partie intégrante de la spécification Java EE
- 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

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 dynamique
- Approche Bottom/Up : Développer et annoter les classes

Annotation JAX-RS

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 de Classes

Annotation @PATH

› 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")
```

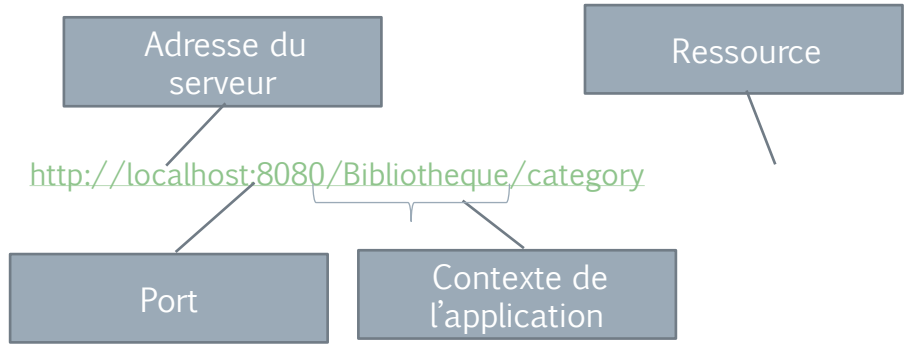
› Expose la classe comme ressource dans le WS

Les annotations de Classes

Annotation @PATH

- @PATH définit la racine des ressources (Root Racine Ressources)
- L'annotation permet de rendre une classe accessible par une requête HTTP
- La valeur donnée correspond à l'uri relative de la ressource

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



Exemple

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

http://localhost:8080/MyApplication/hello

Les annotations des méthodes

Annotation @PATH

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

Exemple

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

http://localhost:8080/MyApplication/hello/greetings

Annotations Dynamiques

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

Annotations Dynamiques Possibles

- @PathParam ➔ Récupérer dans l'URL (Path)
- @QueryParam ➔ Récupérer les paramètres de l'URL (Query)

Exemple 1 (@PathParam)

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

http://localhost:8080/MyApplication/hello/greetings/Mohamed

Exemple 2 (@QueryParam)

```
@PATH("/greetings")
@GET
@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@Produces ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
public String sayHello (@QueryParam("nom") String nom){
return "Hello " + nom;
}
```

http://localhost:8080/ MyApplication/hello/greetings/?nom=Mohamed

Les annotations des méthodes

Annotations @GET, @POST, @PUT, @DELETE

- Permettent de mapper une méthode à un type de requête HTTP
- Ne sont utilisables que sur des méthodes
- Le nom de la méthode n'a pas d'importance, JAX détermine la méthode à exécuter en fonction de la requête

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

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

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

GET http://localhost:8080/Bibliotheque/category/IRS2

Les annotations des méthodes

Annotations @GET, @POST, @PUT, @DELETE

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

Les annotations des méthodes

Annotation @Consumes

- Placé sur les méthodes
- spécifie le format des données d'entrées de la méthode
- Correspond à la clef Content-Type de l'en tête des requêtes HTTP

Les annotations des méthodes

Annotation @Produces

› Placé sur les méthodes

› Spécifie le format de données renvoyées par la méthode

› Correspond à la clef Accept de l'en tête des requêtes HTTP

Les annotations des méthodes

Valeurs @Consumes / @Produces

- Plusieurs Mime-Type
  - text/plain (`MediaType.TEXT_PLAIN`)
  - application/json (`MediaType.APPLICATION_JSON`)
  - application/xml (`MediaType.APPLICATION_JSON`)
  - ...

Exemple

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

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

Annotation @GET

- Opération de lecture
- Peut accepter des paramètres dans l'URL de la requête
- Pas de données dans le corps de la requête
- L'annotation @Consumes est obsolète
- Renvoi des données dans le corps de la réponse
- L'annotation @Produces est obligatoire (format de retour)

Annotation @POST

- Opération d'écriture (Create)
- Reçoit des données dans le corps de la requête
- L'annotation @Consumes est obligatoire (format des données d'entrée)
- Peut renvoyer des données dans le corps de la réponse
- L'annotation @Produces est optionnelle (format des données de retour)

Annotation @PUT

- Opération d'écriture (Update)
- Accepte des paramètres dans l'URL de la requête
- Reçoit des données dans le corps de la requête
- L'annotation @Consumes est obligatoire (format des données d'entrée)
- Peut renvoyer des données dans le corps de la réponse
- L'annotation @Produces est optionnelle (format des données de retour)

Annotation @DELETE

- Opération de suppression (Delete)
- Accepte des paramètres dans l'URL de la requête
- Pas de données dans le corps de la requête
- L'annotation @Consumes est obsolète
- Peut renvoyer des données dans le corps de la réponse
- L'annotation @Produces est optionnelle (format des données de retour)

Exemple complet

```
@Path("livre")
public class LivreFacadeREST extends AbstractFacade<Livre> {

@POST
@Consumes({"application/xml", "application/json"})
public void create(Livre entity) {
super.create(entity);
}

@PUT
@Consumes({"application/xml", "application/json"})
public void edit(Livre entity) {
super.edit(entity);
}

@DELETE @Path("{id}")
public void remove(@PathParam("id") Long id) {
super.remove(super.find(id));
}

@GET @Path("{id}")
@Produces({"application/xml", "application/json"})
public Livre find(@PathParam("id") Long id) {
return super.find(id);
}

@GET
@Produces({"application/xml", "application/json"})
public List<Livre> findAll() {
 return super.findAll();
}

@GET @Path("{from}/{to}") @Produces({"application/xml", "application/json"})
public List<Livre> findRange(@PathParam("from") Integer from, @PathParam("to") Integer to) {
return super.findRange(new int[]{from, to});
}
}
```

Outils de test

- Il existe de nombreux outils en ligne permettant de tester les services Web REST
- Sur ligne de commande : CURL (Client Url Request Library)
  -X : Verbe
  -I : Header de la réponse
  -H : Spécifie le header de la requête
  -d : Data à envoyer (`-d '{"title":"Along came a Spider"}'`)
  -v : Verbose
- Certains sont disponibles sous forme d'extension que vous pouvez installer dans les navigateurs
  - PostMan
  - RestConsole