Systèmes distribués

Filière INDIA (3ème année)

Pr. Abderrahim EL QADI

Département Mathématique appliquée et Génie Informatique

ENSAM-Université Mohammed V de Rabat

A.U. 2023/2024

Plan

Partie 1:

1. Fondements et concepts des systèmes distribues
2. Organisation des applications distribuées
3. Web services - Architecture RestFul

Partie 2:

1. Hibernate & JPA / MAVEN
2. Spring Boot
3. Microservice

Partie 3: DevOps

1. DevOps (Développement et Opérations)
2. Docker
3. Git/GitHub
4. Containers

Partie 1 :

1. Fondements et concepts des systèmes distribues
2. Organisation des applications distribuées
3. Web services - Architecture RestFul

Références

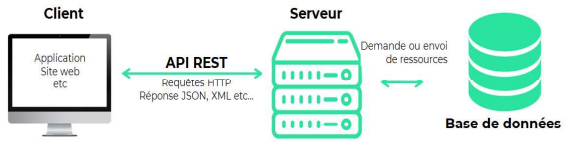
1. Services Web avec J2EE et .NET : Conception et implémentations ; Libero M. , Christian B., Xavier L.; Edition Eyrolles.
2. http://apiacoa.org/teaching/webservices/index.fr.html
3. www.labri.fr/perso/xblanc/data/OldCourses/XB4-WebServices.ppt
4. https://www.ibisc.univ-evry.fr/~tmelliti/cours/CPAR/cours6.pdf
5. Introduction aux systèmes répartis. Frank Singhoff. Université de Brest, France.
6. RESTful Java with JAX-RS 2.0. SECOND EDITION, Bill Burke. Edition: Oreilly
7. SOA – Services web REST, Mickaël BARON – 2011 (Rév. Janvier 2019).
8. Comprendre architecture des Web Services Rest. Amosse Edouard.

1. Fondements et concepts des systèmes distribués

1.1. Systèmes distribués (réparties)

- Ensemble composé d'éléments (composants) :
  - collaborent à une ou plusieurs tâches communes ;
  - communiquent par l'intermédiaire d'un réseau (protocole)
- Les éléments ont des fonctions de traitement (processeurs), de stockage (mémoire), de relation avec le monde extérieur (capteurs, actionneurs).

Exemple : Une API assure la transmission de données entre deux applications, appelées **"le client" et "le serveur".**



Avantages des systèmes distribués

- Performance : calculs parallèles en des temps plus courts.
- Transparence à la localisation : L'utilisateur ignore la situation géographique des ressources.
- Transparence d'accès : L'utilisateur accède à une ressource locale ou distante d'une façon identique.
- Transparence à l'hétérogénéité (**interopérabilité**): L'utilisateur n'a pas à se soucier des différences matérielles ou logicielles des ressources qu'il utilise.
- Transparence aux pannes (réseaux, machines, logiciels): Les pannes sont cachées à l'utilisateur.
- Confidentialité : les données brutes ne sont pas disponibles partout au même moment, seules certaines vues sont exportées.

1.2. Services et interfaces

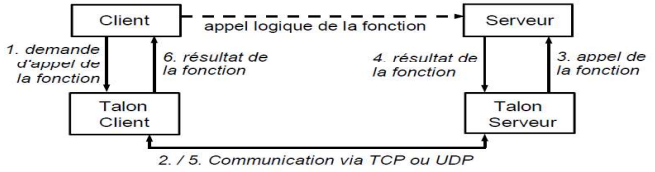
- Une application distribuée permet de répondre à un problème spécifique, fourniture de **services** à ses utilisateurs (qui peuvent être d'autres applications).
- Service : comportement défini par contrat, qui peut être implémenté et fourni par un composant pour être utilisé par un autre composant, sur la base exclusive du contrat.
- Un service est accessible via une ou plusieurs interfaces.

- Une interface décrit l'interaction entre client et fournisseur du service.
  - •Point de vue opérationnel : définition des opérations et structures de données qui concourent à la réalisation du service
  - •Point de vue contractuel : définition du contrat entre client et fournisseur.

- Où se trouvent ces services ?
  - Dans la couche transport (UNIX + TCP/IP).
  - Dans le langage de programmation (Ada 95, Java RMI).
  - Dans le "Middleware"/interlogiciel (CORBA, RPC, .NET). Logiciel se situant entre le système d'exploitation et les applications.

Exemple: RPC (Remote Procedure Call)

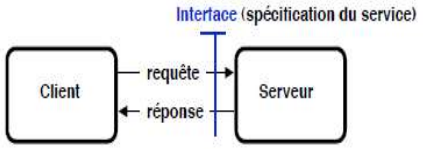
- Idée : remplacer les appels réseaux par des appels de sous-programmes
- est un protocole réseau permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'applications.
- Le client envoie un message de requête à un serveur distant connu pour exécuter une procédure spécifique avec des paramètres spécifiques.
- Le serveur distant envoie une réponse au client puis l'application continue son déroulement.



2. Organisation des applications distribuées

a. Architecture Client-serveur (Architecture 2 – tiers) :

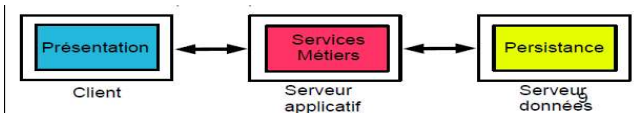
- Client : se charge de la présentation, interface utilisateur
- Serveur : se charge de la partie persistance, gestion physique des données
- Le client demande l'exécution d'un service
- Le serveur réalise le service
- Client et serveur sont (en général, pas nécessairement) localisés sur deux machines distinctes
- Indépendance interface-réalisation

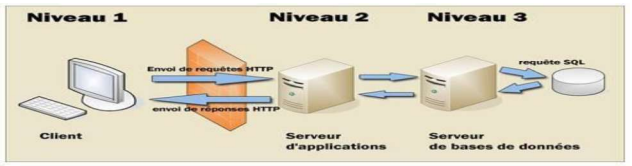


Exemples:

- Consultation des pages Web (HTTP)
- Envoi & réception des e-mails (SMTP, POP, IMAP)

b. Architecture 3-tiers

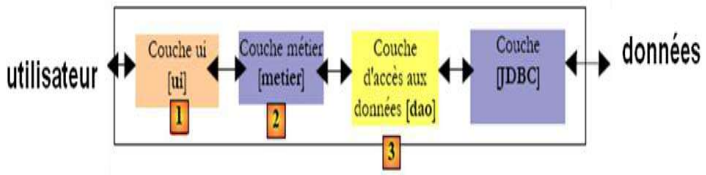




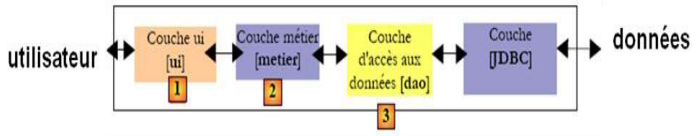
– Un client léger qui n'est autre qu'un navigateur web permettant à son utilisateur d'accéder à l'application via internet.

– Un middle tiers : le serveur d'application qui héberge toutes les couches de l'application.

– Un tiers de données qui n'est autre que le serveur de base de données.



- **Interface de communication** :
  - Chaque niveau ne communique qu'avec ses voisins immédiats au travers une interface de communication bien définie
  - Pas de communications directes entre la couche présentation et données.



- La couche [1], appelée ici [ui] (**User Interface**) est la couche qui dialogue avec l'utilisateur, via une interface graphique, console ou une interface web. Elle a pour rôle de fournir des données provenant de l'utilisateur à la couche [2] ou bien de présenter à l'utilisateur des données fournies par la couche [2].
- La couche [2], appelée ici [**metier**] est la couche qui applique les règles dites métier, c.à.d. la logique spécifique de l'application, sans se préoccuper de savoir d'où viennent les données qu'on lui donne, ni où vont les résultats qu'elle produit.
- La couche [3], appelée ici [**dao**] (Data Access Object) est la couche qui fournit à la couche [2] des données pré-enregistrées (fichiers, bases de données, ...) et qui enregistre certains des résultats fournis par la couche [2].

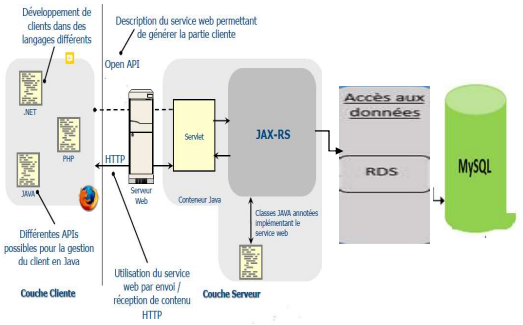
- Avantages
  - Séparation de la couche présentation et métier, trop souvent imbriquées dans les architectures client/serveur classiques
  - Meilleure maintenance : isolement des fonctionnalités (e.g. changer facilement l'IHM)
  - Allègement du poste de travail client (par rapport aux clients/serveur de données)

c. Architecture n-Tiers

L'architecture N-tier, ou aussi appelée multi-tier, est une architecture client-serveur dans la quelle une application est exécutée par plusieurs composants logiciels distincts (intègre un grand nombre de technologies).

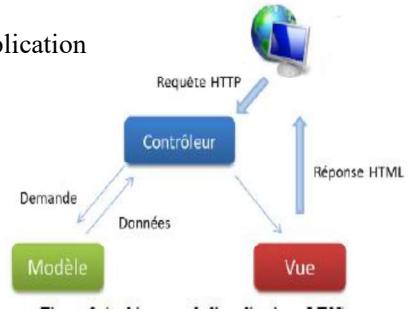
- Rajoute des étages / couches en plus
- La couche applicative peut s'appuyer et interagir avec d'autres services
- Service métier utilise d'autres services métiers
- Chaque service correspond à une couche

D'où le terme de N-tiers



d. Architecture MVC

– Cette architecture, se sépare sur trois couches : entités de données, interface et traitement de données.
  - Le modèle représente les données de l'application stockées dans une base de données.
- La vue correspond à l'IHM.
- Le contrôleur assure les échanges entre la vue et le modèle.



Le rôle du contrôleur est d'orchestrer la procédure entre une vue et un modèle. Il analyse la requête envoyée par un client, effectue les contrôles nécessaires afin d'appeler le modèle convenable.

e. Middleware : couche de logiciel (réparti) :

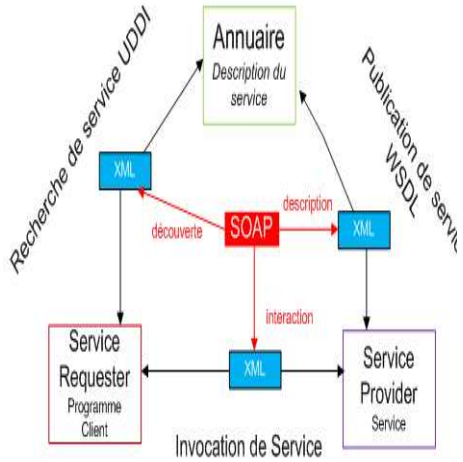
- masque l'hétérogénéité des machines et systèmes
- masque la répartition des traitements et données
- fournit une interface commode aux applications (modèle de programmation + API)

- Les Classes de Middleware :
  - Orientés objets distribués : Java RMI, CORBA, ...
  - Orientés composants distribués : EJB, CCM, ...
  - Orientés transactionnels : JTS de Sun, MTS de Microsoft
  - Orientés messages : JMS de Sun, MSMQ de Microsoft, ...
  - Orientés Web : **Web Services** (XML-RPC, SOAP)
  - Orientés SGBD / SQL : ODBC, JDBC de Sun.

3. Web services - Architecture RestFul

– Les web services sont des programmes :
  – écrits en XML
  – identifiés par un URI (Uniform Resource Identifier)
– Ils sont des « librairies » fournissant des données et des services à d'autres applications.
– Ils permettent la distribution de l'information entre clients, fournisseurs, partenaires commerciaux et leurs différentes plates-formes.

Exemple d'architecture :



Trois acteurs :
- **le fournisseur de service** (Service Provider) :
  - définit le service
  - publie sa description dans l'annuaire
  - réalise les opérations
- **l'annuaire** (discovery agency) :
  - reçoit et enregistre les descriptions de services publiées par les fournisseurs
  - reçoit et répond aux recherches de services lancées par les clients.
- **le client** (Service Requestor) :
  - obtient la description du service grâce à l'annuaire
  - utilise le service.

– Les web services implémentent la logique métier rendue consommable par l'utilisation de standards (TCP/IP, URI//URL, HTTP/SMTP/, SOAP..., pour le transport, puis XML pour le contenu), ce qui permet à n'importe quelle technologie utilisant ces standards de pouvoir l'exploiter, facilitant ainsi l'interopérabilité des applications.
– Ils utilisent le port 80, qui est toujours ouvert (employé par le protocole HTTP utilisé par les navigateurs Web).

Avec cet avantage, les services web représentent une sorte de tunneling (encapsulation de données d'un protocole réseau dans un autre).

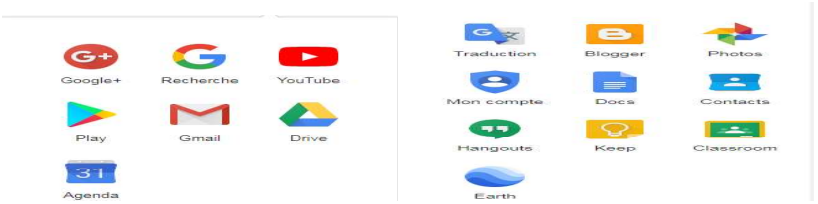
– **Les Web services assurent** :
  – Interopérabilité :
    •Différentes Applications (Client et/ou Serveur)
    •Différents Systèmes d'Exploitation
    •Différents Hardwares
  – Le couplage faible des applications (évolution indépendante) et leur coopération via des interfaces de haut niveau d'abstraction (services globaux).
  – L'utilisation des technologies standards du Web telles HTTP et XML par les services Web facilite le développement d'applications réparties sur Internet, et permet d'avoir des applications très faiblement couplées.

– La plus importante innovation des Services Web est l'utilisation de XML **(eXtensible Markup Language)** comme EDI.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<adresse>
   <lieu>
        <rue> rue de badr</rue>
        <numero>8</numero>
   </lieu>
   <ville>Rabat></ville>
   <code>10000</code>
</adresse>
```


Exemples :

– Services Web de Google:



- Google Docs : Solution en ligne directement concurrente de Microsoft Office
- Google Agenda : Calendrier et agenda en ligne interconnecté avec Gmail.
- Google Meet : Service de communication en vidéo
- Google Classroom : Service d'enseignement à distance

3.1. Architecture d'un service Web

– Les technologies utilisées par les services Web sont HTTP, WSDL, REST, XML-RPC, SOAP et UDDI :

– **XML-RPC** ***(Remote procedure call):*** est un protocole simple utilisant XML pour effectuer des messages RPC. Les requêtes sont écrites en XML et envoyées via HTTP POST. Les requêtes sont intégrées dans le corps de la réponse HTTP.

– **REST** : (***Representational State Transfer***) est une architecture permet de construire une application pour les systèmes distribués comme le World Wide Web.

– **SOAP** : (***Simple Object Access Protocol***) est un protocole standard de communication.

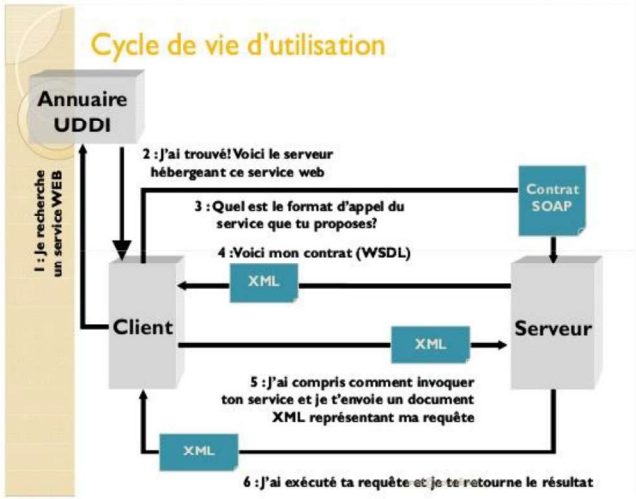
- C'est l'épine dorsale du système d'interopérabilité.
- Il circule sur le protocole HTTP et permet d'effectuer des appels de méthodes à distance.

– **WSDL** : (***Web Services Description Language***) est un langage de description standard.

- C'est l'interface présentée aux utilisateurs.
- Il est basé sur XML et permet de décrire de façon précise les détails concernant le service Web tels que les protocoles, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie et les exceptions pouvant être envoyées.

– **UDDI**: (***Universal Description, Discovery and Integration***) est un annuaire de services. Il fournit l'infrastructure de base pour la publication et la découverte des services Web. Les informations qu'il contient peuvent être séparées en trois types :

- Les pages blanches qui incluent l'adresse, le contact et les identifiants relatifs au service Web ;
- Les pages jaunes qui identifient les secteurs d'affaires relatifs au service Web ;
- Les pages vertes qui donnent les informations techniques.



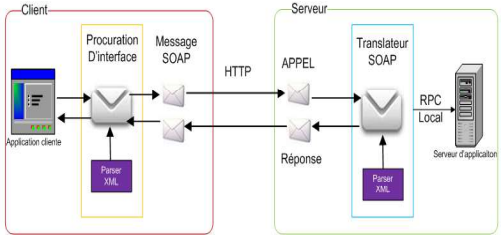
3.2. Protocole de communication SOAP

– SOAP (*Simple Object Access Protocol*) a pour principal objectif **d'assurer la communication entre machines**.

– est basé sur XML

– est un protocole permet d'appeler une méthode RPC et d'envoyer des messages aux machines distantes via HTTP. Ce protocole permet de fournir au client une grande quantité d'informations récupérées sur un réseau de serveurs tiers.

– SOAP a été dépassé par REST dans ces dernières années

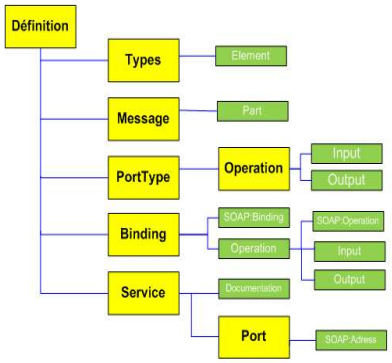


Structure de SOAP : Exemple

SOAP Request :	**SOAP Response :**
<?xml version="1.0" encoding="UTF-8"?> <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"> <S:Header/> <S:Body> <ns2:sommer xmlns:ns2="http://TP/"> <a>3 5 </ns2:sommer> </S:Body> </S:Envelope>	<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"> <S:Body> <ns2:sommerResponse xmlns:ns2="http://TP/"> <return>8</return> </ns2:sommerResponse> </S:Body> </S:Envelope>

Langage de description WSDL

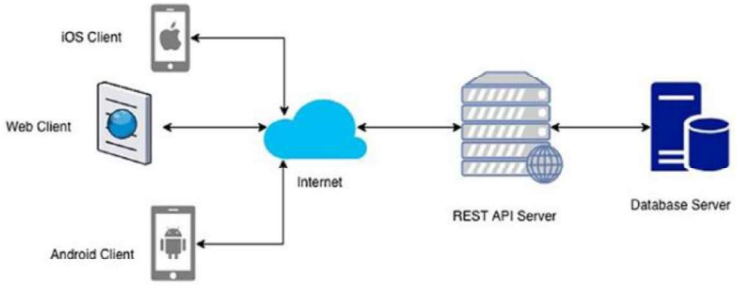
– Décrit les aspects techniques d'implantation d'un service web (quel est le protocole utilisé, quel est l'adresse du service)
Une description WSDL est un document XML qui commence par la balise **Definition** et contient les balises suivantes :



Structure d'un document WSDL

3.3. Architecture REST

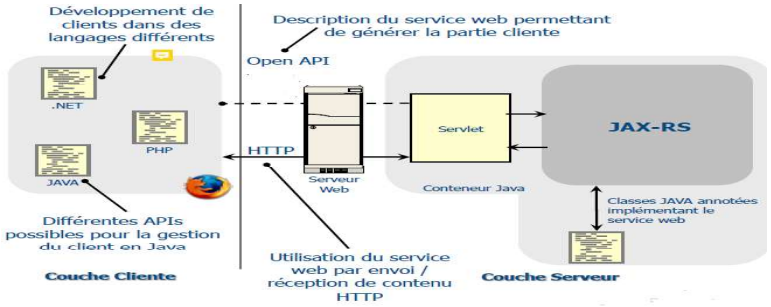
– Le premier concept sur REST (**Re**presentational **S**tate **T**ransfer) a été lancé en 2000 dans la thèse de doctorat de **Roy Thomas Fielding** (co-fondateur du protocole **HTTP**).



REST Architecture

– REST n'est pas un protocole ou un format, contrairement à SOAP (Simple Object Access Protocol), HTTP ou RCP, mais un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP.
– Utiliser dans le développement des applications orientés ressources (ROA) ou orientées données (DOA)
– Les applications respectant l'architecture REST sont dites **RESTful**
– Les RESTful Web Services sont légers, faciles à étendre et à entretenir, que les Web Services basés sur SOAP et WSDL.

– **Exemple d'architecture : RESTful avec JAX-RS**.



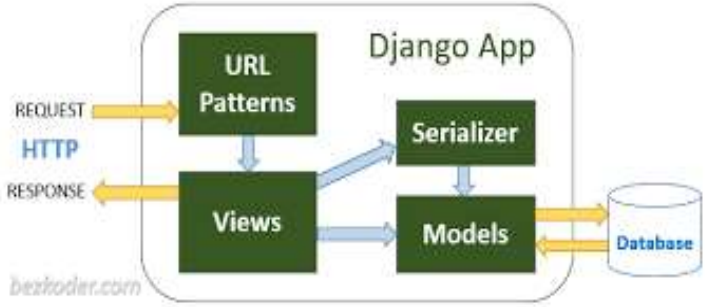
JAX-RS Fonctionnement

– **JAX-RS** est l'acronyme Java API for RESTful Web Services.
– Seule la configuration de la Servlet « JAX-RS » est requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées

– Différentes implémentations de la spécification JAX-RS sont disponibles :

- 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.

– **Exemple 2 : Architecture REST avec Django Python**



3.3.1. C'est quoi REST ?

– **REST** définit les règles de l'architecture des **Web Services**, en mettant l'accent sur les ressources du système par le biais de méthodes **HTTP**.

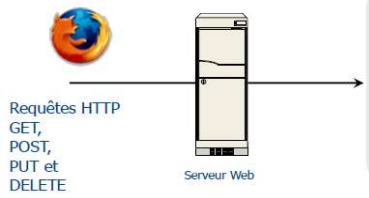
▪ **Ressource :**

- est un objet identifiable sur le système (ex. Livre, Client)
- identifié par une URI (http://localhost:8082/restful3-livre/rest/livres)
- est annotée en java par : **@Path, @PathParam, @QueryParam, @FormParam, @HeaderParam**
- En python est annotée par : **@app.route('/')**

- Peut subir quatre opérations de bases **CRUD** correspondant aux quatre principaux types de requêtes HTTP (**GET, PUT, POST, DELETE**):

Opération	**Requête**
CREATE	POST
RETRIEVE	GET
UPDATE	PUT
DELETE	DELETE

– Les opérations CRUD sur des ressources sont réalisées au travers des méthodes HTTP



- **Méthodes** HTTP pour manipuler l'identifiant :
  - **POST** : pour créer une ressource sur le serveur,
  - **GET** : pour accéder à une ressource, demander les données, insérer, éditer ou supprimer les données sur le Serveur.
  - **PUT** : pour modifier l'état d'une ressource ou pour la mettre à jour
  - **DELETE** : pour annuler ou supprimer une ressource.

Exemple 1: Service web REST « hello » avec Java

Définition d'un chemin de ressource pour associer une ressource hello à une URI

```
@Path("/hello")

public class HelloWorldResource {
    @GET
    @Produces("text/plain")
    public String getHelloWorld() {
            return "Rest API avec Java";
    }
}
```

Lecture de la ressource HelloWorld via une requête HTTP de type GET

Le type MIME de la réponse est de type text/plain

Exécution de la ressource : http://localhost:8080/projects/hello

Résultat : Rest API avec Java.

Exemple 2: Service web REST « hello » avec Flask et Python

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Rest API avec Flask/Python!'
if __name__ == '__main__':
  app.run(port=5002)
```

Exécution de la ressource : http://127.0.0.1:5002/

Résultat : Rest API avec Flask/Python!.

3.3.2. Annotations avec JAX-RS / Flask API

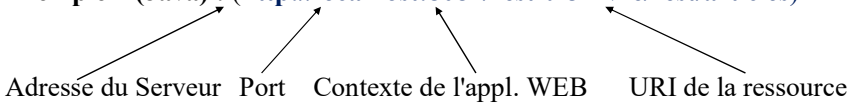
3.3.2.1. @Path

- Une classe Java doit être annotée par @Path pour qu'elle puisse être traitée par des requêtes HTTP
- L'annotation @Path sur une classe définit des ressources appelées racines (Root Resource Class)

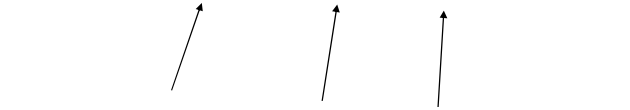
  Exemple : @Path ("/articles")
- La valeur donnée à @Path correspond à une expression URI relative au contexte de l'application web

  Avec Flask Python : @app.route('/articles')

Exemple 1 (Java) : (http://localhost:8082/restful3-livre/rest/articles)



Exemple 2 (Flask Python) : **http://127.0.0.1:5005/person/getAll**



3.3.2.2. @Produces et @Consumes

- Ces deux annotations permettent de déclarer les types MIME supportés pour la requête (@Consumes) et pour la réponse (@Produces). Ces déclarations sont facultatives.
- Parfois, le Client peut également spécifier les types de données de retour qu'il souhaite (JSON ou XML, ...).

MIME-Type	Content-Type
TEXT	text/plain
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

JSON « JavaScript Obect Notation » est un format d'échange de données, facile à lire par un humain et interpréter par une machine.

3.3.2.3. Annotations des méthodes CRUD

- Elles correspondent aux cinq méthodes HTTP qui portent le même nom.
- Une telle méthode doit être public. Lorsqu'une telle requête arrive, la bonne méthode est invoquée, avec les paramètres annotés correctement positionnés.
- Une méthode annotée peut aussi porter un paramètre non annoté, que l'on appelle le paramètre d'entité. Ce paramètre portera l'intégralité de la requête.

Exemple:

```
@POST @Path("/postlivre")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public Livre addLivreAsJson(Livre livre){      .....      }
```

```
@app.route('/person/getAll', methods=['GET'])                # Python
def get_all():
…..
```


3.3.2.4. Annotations des paramètres

- La valeur définie dans @Path ne se limite pas seulement aux expressions constantes
- Possibilité de définir des expressions plus complexes appelées : "Template Parameters"
- Pour distinguer une expression complexe dans la valeur du @Path, son contenu est délimité par { … }

Exemple 1:

```
@Path("/getid/{id}")
public Article getArticle(@PathParam("id") int id){
        return listeArticles.getArticle(id);
}
```

Exemple 2:

```
@GET
@Produces({MediaType.APPLICATION_JSON})
@Path("lister/{nom}/{prenom}")

public String hello(@PathParam("nom") String nom,
            @PathParam("prenom") String prenom)
{
      return "Hello " + nom + " " + prenom;
}
```

```
@app.route('/person/getId', methods=['GET'])
def get_records_id():
   if 'id' in request.args:
      id = int(request.args['id'])
   else:
      return "Erreur: id n'existe pas."
```


@QueryParam

– est un type de paramètre à extraire pour l'utiliser dans la classe de ressources.
– Les paramètres sont extraits à partir des paramètres de requête URI.

Exemple:

```
Path("/restwb")
public class LivreService {
      @GET
      @Path("/data")
      @Produces("application/json")
      public Livre getLivreDetails(@QueryParam("code") int code,
                  @QueryParam("titre") String titre) {
            .....
      }
}
```

URI Pattern : **http://localhost:8080/rest/restwb/data?code=1&titre=JAX-RS**

@FormParam

Les paramètres sont extraits à partir des formulaires (HTML, JSP, ...)
Exemple:

```
@POST
@Path("/savedata")
@Consumes("application/x-www-form-urlencoded")
public Response saveLivre(@FormParam("code") int code, @FormParam("titre")
      String  titre, @FormParam("auteur") String auteur, @FormParam("prix")
      double prix) {
            Livre livre=new Livre();
            livre.setCode(code);   livre.setTitre(titre);
            livre.setAuteur(auteur);  livre.setPrix(prix);
            return  Response.ok(livre).build();
}
```


@HeaderParam

– permet d'associer une valeur d'un champ HTTP à un champ ou un paramètre d'une méthode de la classe de ressources.

Exemple:

```
@Path("/header")
public class EmployeeService {
    @GET
    @Path("/getemp")
    @Produces("application/json")
    @Formatted
    public Response getEmp(@HeaderParam("User1") String user1) {
        Map<String,String> map = new HashMap<String,String>();
        map.put("User1", user1);
        return Response.ok(map).build();
    }
}
```