

Technologies des services du Web

Jérôme DEGROOTE

Société générale



Organisation

- Répartition :
 - 3h00 CM
 - 5 * 1h30 à distance
 - 4h30 pour la séance d'évaluation
- Evaluation
 - 1 note de projet (application web à réaliser)

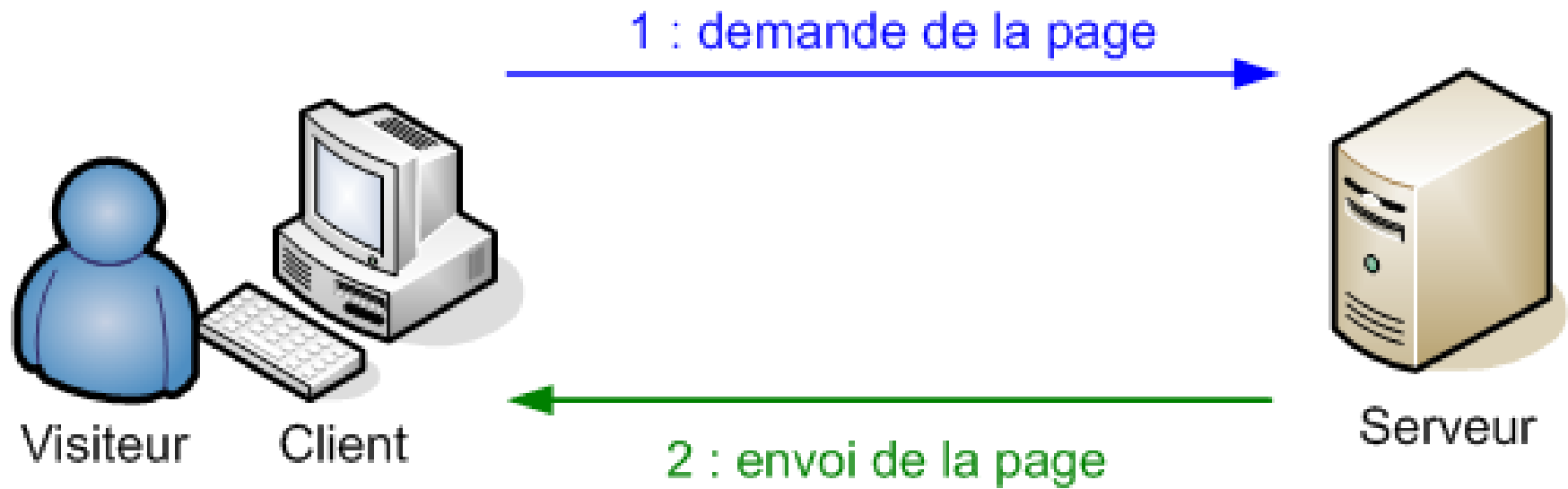
Plan du cours

- Web services
- Persistance
- Front

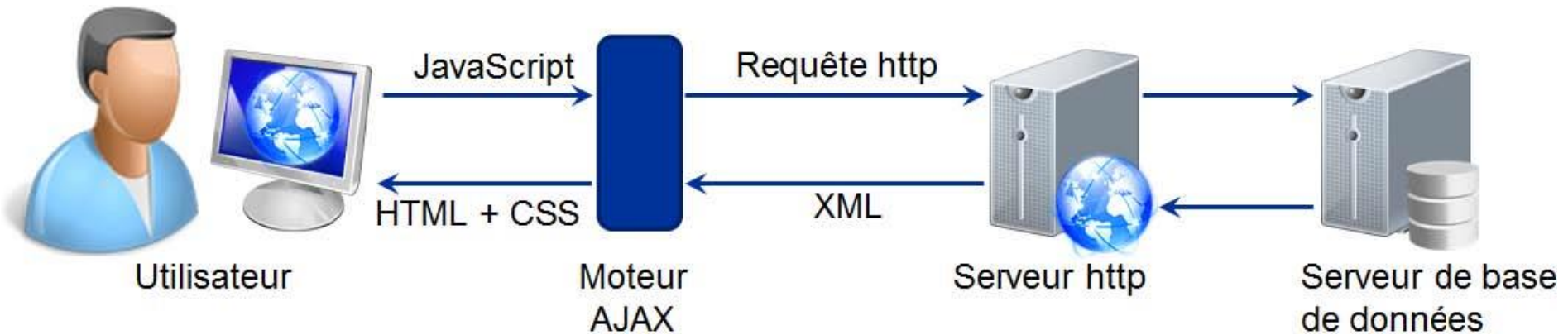
Web services

A decorative graphic consisting of a vertical green line on the left and a horizontal green line extending to the right, intersecting at the level of the text.

Requêtes HTTP



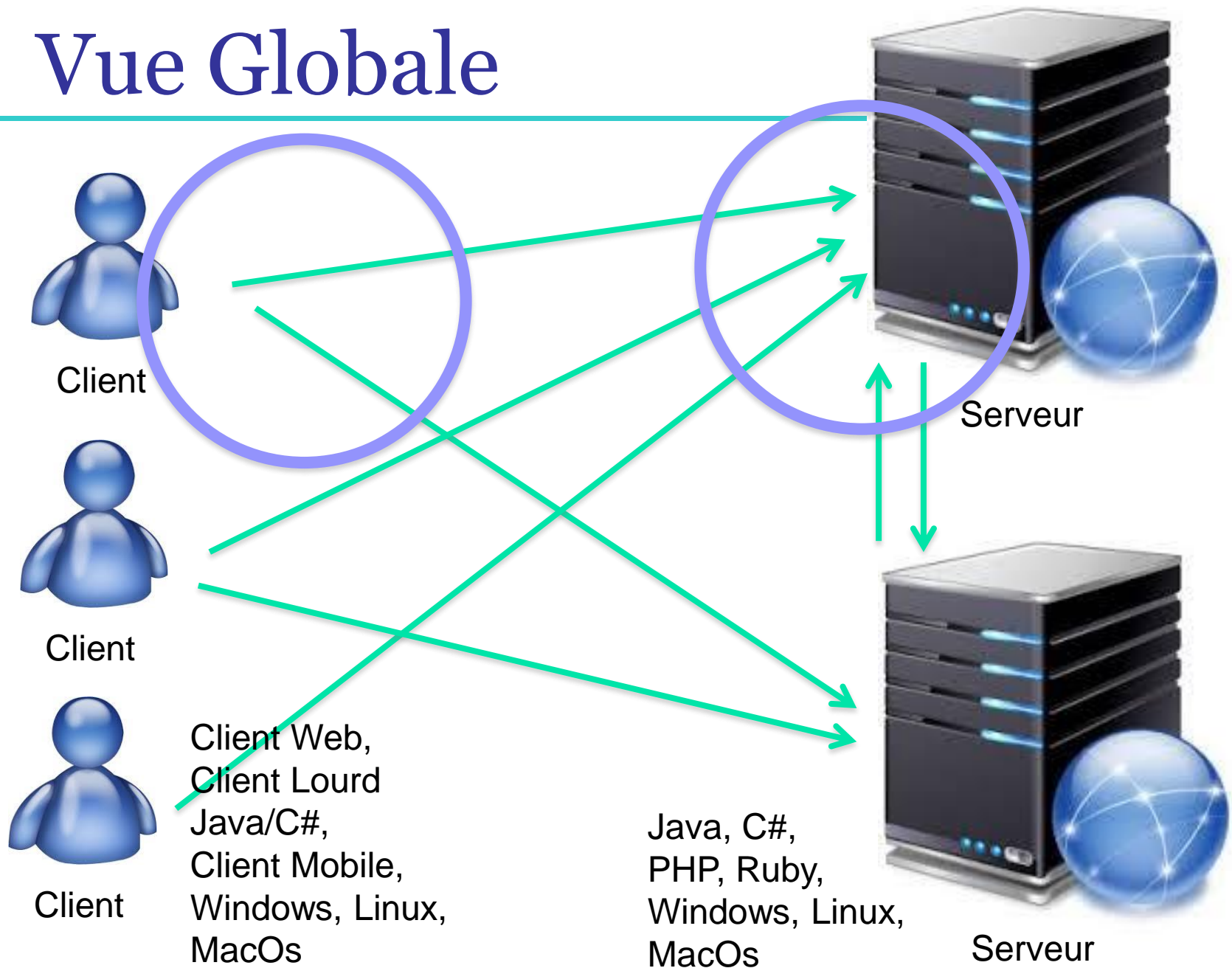
Requêtes HTTP



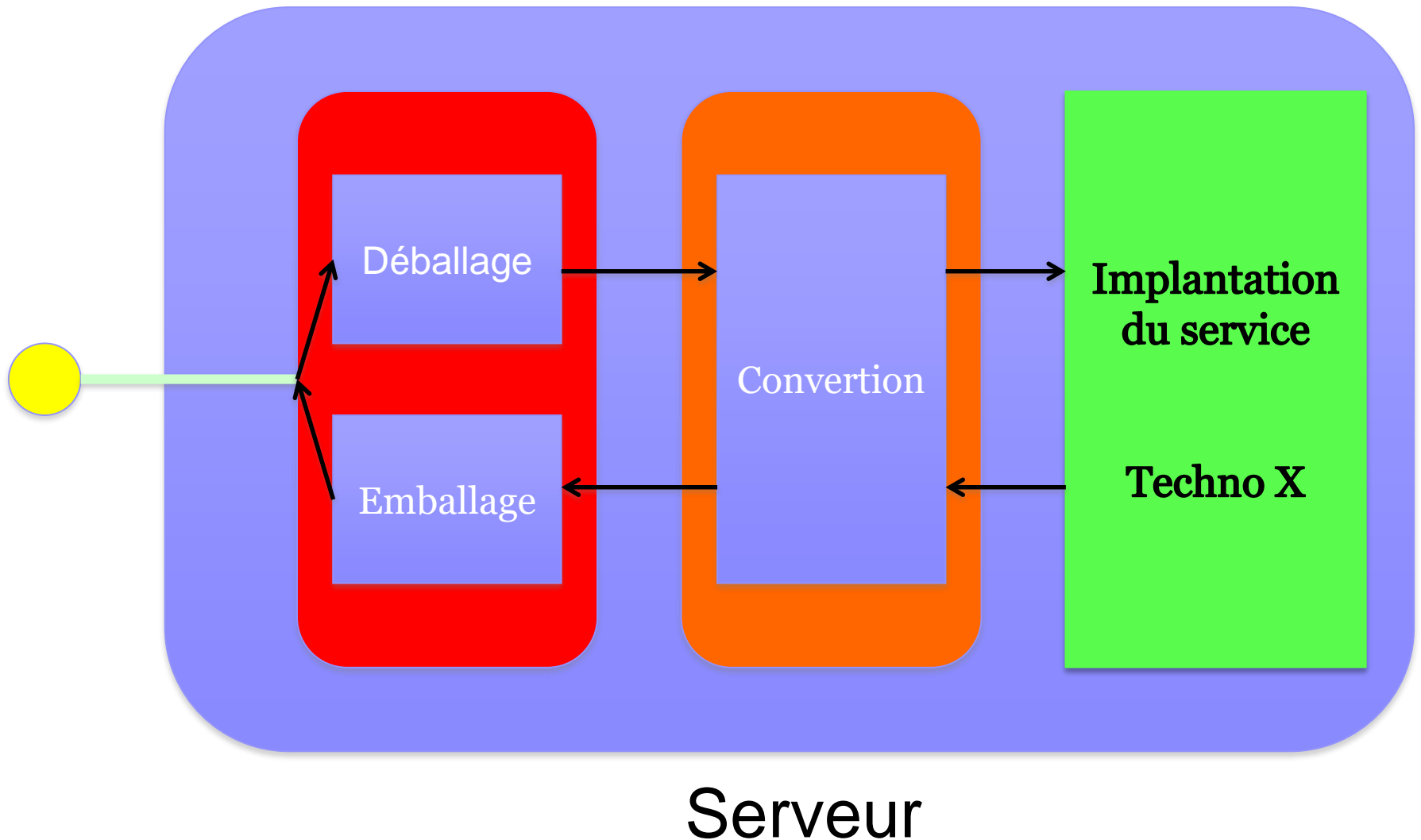
Qu'est-ce qu'un service web ?

- Technologie permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans un environnement distribué
- Un service web fournit une fonctionnalité exposée sur internet (ou intranet)

Vue Globale



Zoom côté serveur



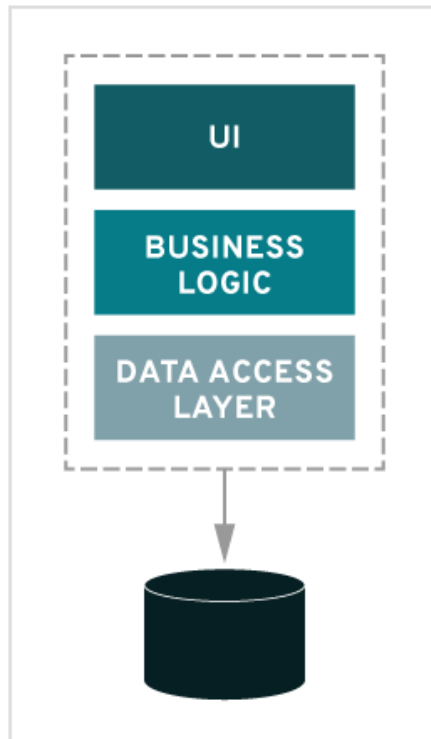
Zoom côté client

- Même principe que côté serveur
 - Protocole/langage pivot à traiter de la même façon côté client et serveur
- Technologies différentes
- Côté serveur
 - Serveur d'application, Conteneur logiciel, Injection au déploiement, Conception des web services assistées, etc.
- Côté client
 - Outils de générations pour client lourd ou léger
 - Eventuellement gestion "à la main"

Architectures possibles

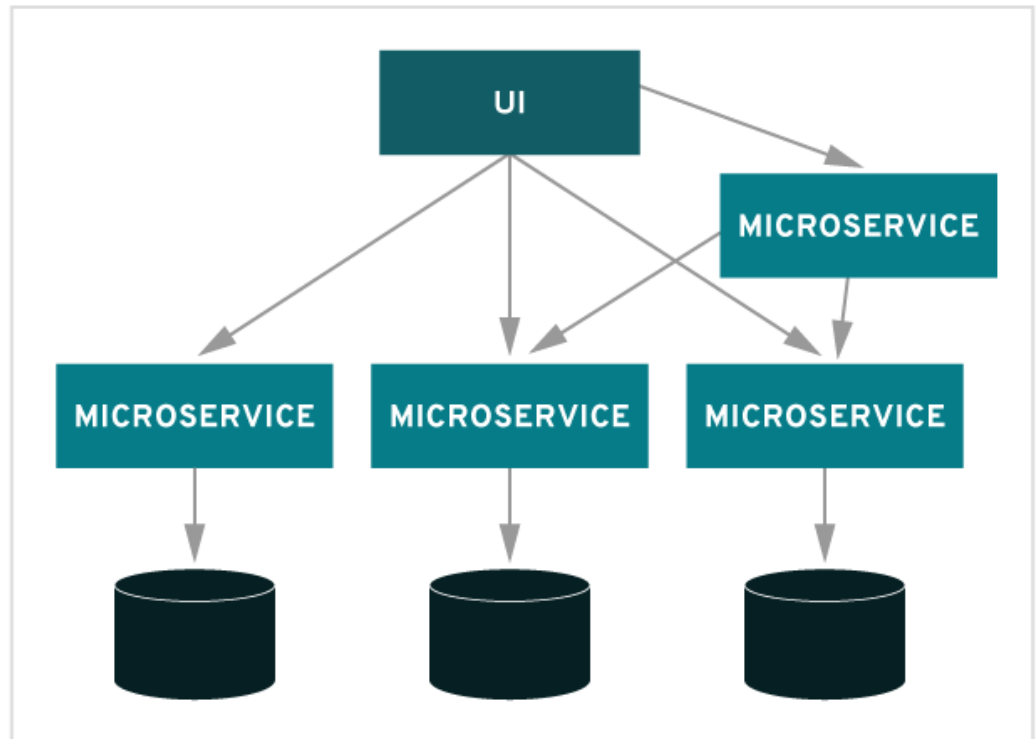
Architectures possibles

MONOLITHIC



VS.

MICROSERVICES



Principes des microservices

- Indépendance de chaque microservice
 - Evolutivité et maintenabilité
 - Time-to-market réduit
 - On accepte la duplication de code
 - Chaque microservice peut être codé dans son propre langage
- Application sans état (stateless)
 - Redémarrage/Ajout/Suppression d'une instance sans impact

Principes des microservices

- Focus sur le déploiement continu et le devops
- Application autonome (pas de serveur d'application)
- Versionning des services pour les changements non rétrocompatibles
- Base de données non partagées

Réaliser un service

- Ex : Convertir des valeurs
- Ex : Faire des calculs
- Ex : Obtenir le(s) numéro(s) de téléphone(s) d'une personne
- Retour vide, un élément ou plusieurs

Messages

A decorative graphic consisting of a vertical green line on the left and a horizontal green line extending to the right, intersecting at the start of the word 'Messages'.

Le XML

```
<root>
  <person id='1'>
    <name>Alan</name>
    <url>http://www.google.com</url>
  </person>
  <person id='2'>
    <name>Louis</name>
    <url>http://www.yahoo.com</url>
  </person>
</root>
```

JSON

```
{
  "root": {
    "person": [
      {
        "@id": "1",
        "name": "Alan",
        "url": "http://www.google.com"
      },
      {
        "@id": "2",
        "name": "Louis",
        "url": "http://www.yahoo.com"
      }
    ]
  }
}
```

Premiers pas

- Consommons les APIs suivantes :
 - <https://jsonplaceholder.typicode.com/>
 - <https://docs.github.com/en/rest>

Architecture Rest

A decorative graphic consisting of a vertical green line on the left and a horizontal green line extending to the right, intersecting at the start of the title.

RESTful

- Representational State Transfer
- Bien adapté pour l'intégration de scénarios basiques
- Intégration HTTP facilitée
- JAX-RS (Java API for RESTful Web Services)
 - Jersey, Resteasy

Cible RESTful

- Service sans état
 - Supporte un redémarrage du serveur
- Peut être mis en cache
 - Pas de génération dynamique
- Le fournisseur et le consommateur ont une connaissance mutuelle du contexte et du contenu
- Bande passante limitée
- La consommation du service est simple

Créer un service REST

- Principes

- Identification des ressources par des URI
 - Uniform Resource Identifiers

- Interface uniforme

- PUT, GET, POST, DELETE, PATCH

- Messages auto-descriptifs (XML, JSON, etc.)

- Interactions avec états par des hyperliens

- URI rewriting, cookies, hidden forms...

Une ressource

- Ce sont des POJOs
- Annoté avec un `@Path`
 - Chemin d'accès à la ressource
- Ou une Request Method Designator
 - `@GET`, `@PUT`, `@POST`, `@DELETE`

Langage d'échange

- @Consumes
- @Produces
- Utilise les types MIME
 - application/json
 - application/xml
 - image.jpg
 - text/csv
 - Etc.

Example

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

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

@Path

```
@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String userName) {
        ...
    }
}
```

```
@Path("/employees/{firstname}.{lastname}@{domain}.com")
```

@QueryParam

- @Exemple d'URI

- users/query?from=100&to=200&
orderBy=age&orderBy=name

- Obtention des paramètres

```
@GET
@Path("/query")
public Response getUsers(
    @QueryParam("from") int from,
    @QueryParam("to") int to,
    @QueryParam("orderBy") List<String> orderBy) {
    ....
}
```



Exemple

Données -> Personne

```
package org.test.services;
```

```
public class Personne {
```

```
    int i;
```

```
    String nom;
```

```
    String prenom;
```

```
    int age;
```

```
    Personne(){
```

```
        i = 5;
```

```
        nom = "doe";
```

```
        prenom = "john";
```

```
        age = 2;
```

```
    }
```

```
    // Getters & Setters
```

```
}
```

Service REST

```
package org.test.services;

import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.GenericEntity;

@Path("/test")
public class TestResource {
```

Sortie JSON

@GET

@Produces("application/json")

@Path("/personnejson")

public **Personne** jsonPersonne() {

 Personne p = **new** **Personne**();

return p;

}

Affichage

```
{"age":2,"i":5,"nom":"doe","prenom":"john"}
```

Sortie XML

@GET

@Produces("application/xml")

@Path("/personnxml")

```
public Personne xmlPersonne() {  
    Personne p = new Personne();  
    return p;  
}
```

Affichage

```
<personne>  
  <age>2</age>  
  <i>5</i>  
  <nom>doe</nom>  
  <prenom>john</prenom>  
</personne>
```

Application

- Il est nécessaire d'enregistrer les services dans une classe « Application »
- Liste de toutes les ressources qui seront exposées

Application

```
package org.test;
```

```
import java.util.HashSet;
```

```
import java.util.Set;
```

```
import javax.ws.rs.ApplicationPath;
```

```
import javax.ws.rs.core.Application;
```

```
import org.test.services.TestResource;
```

```
@ApplicationPath("resources")
```

```
public class MyApplication extends Application {
```

```
    public Set<Class<?>> getClasses() {
```

```
        Set<Class<?>> s = new HashSet<Class<?>>();
```

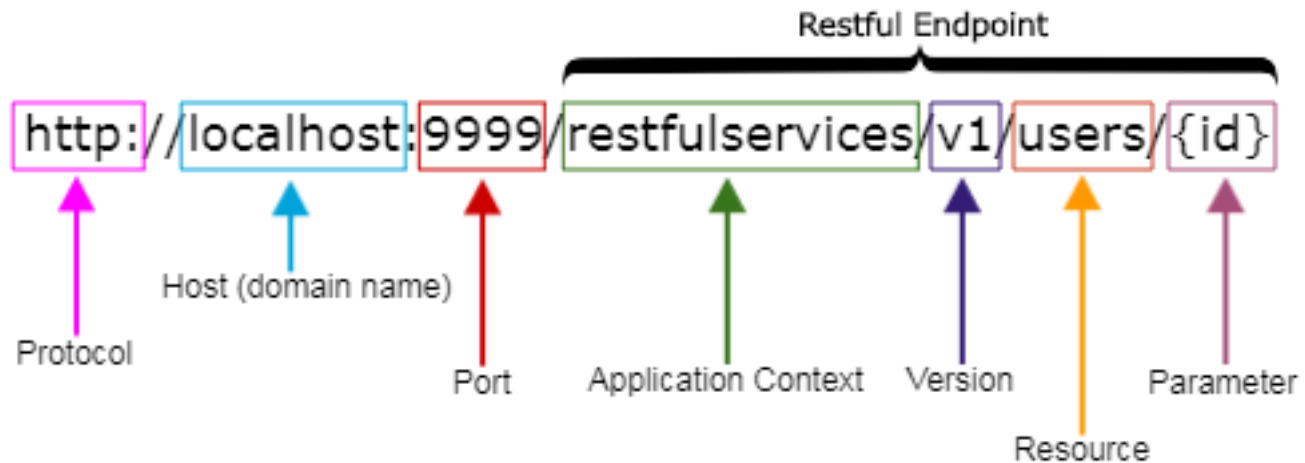
```
        s.add(TestResource.class);
```

```
        return s;
```

```
    }
```

```
}
```

URL Rest



Test des services

- Les requêtes GET sans header peuvent être testées directement depuis le navigateur
- Pour le reste, des outils existent, en voici quelques-uns :



En pratique...

...avec spring boot

Spring Boot

- Framework qui permet de créer des applications autonomes basées sur Spring :
 - <https://start.spring.io/>
 - Starters pour gérer les dépendances
 - Tomcat embarqué
 - Auto-configuration
 - Métriques accessibles via des URLs
 - Healthcheck

Environnement

- Spring boot avec jersey
- Maven



RESTful Web Services in Java.



Jersey

- Pour utiliser les capacités de Jersey dans spring boot, il faut créer une classe de configuration

```
@Component
@ApplicationPath("rest")
@Configuration
public class JerseyConfiguration extends ResourceConfig {
    public JerseyConfiguration() {
        register(MyRessource.class);
    }
}
```

Comment démarrer

- Initialiser un projet spring boot
<https://start.spring.io/>
 - Pensez à cocher la dépendance *Jersey (JAX-RS)*
 - Supprimer les fichiers générés inutiles (.mvn, mvnx et mvnw.cmd)
- Importer le projet dans votre IDE
- Créer un premier service REST permettant de récupérer une personne :
 - Classes JerseyConfiguration, Ressource et Personne

JerseyConfiguration

@Component

@ApplicationPath("rest")

@Configuration

```
public class JerseyConfiguration extends ResourceConfig
{
    public JerseyConfiguration() {
        register(MyResource.class);
    }
}
```

PersonneResource

```
@Path("personnes")
```

```
public class PersonneResource {
```

```
    @GET
```

```
    @Produces(MediaType.APPLICATION_JSON)
```

```
    public Personne hello(@QueryParam("prenom") String prenom,  
        @QueryParam("age") int age) {
```

```
        return new Personne(prenom, age);
```

```
    }
```

```
}
```

Personne

```
public class Personne implements Serializable {  
    private static final long serialVersionUID = -8537962680206576813L;  
    private String prenom;  
    private int age;  
  
    public Personne() {  
        super();  
    }  
    public Personne(String prenom, int age) {  
        super();  
        this.prenom = prenom;  
        this.age = age;  
    }  
    // Getters, Setters ...  
}
```