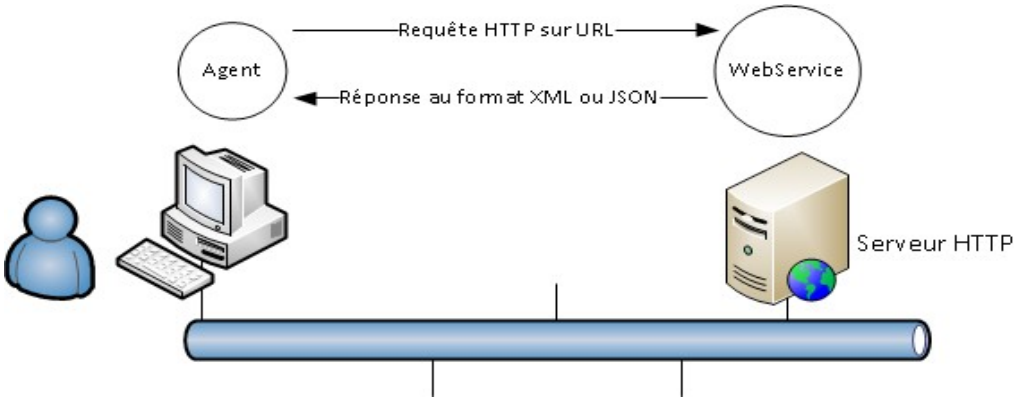
Technologie ReST

Sommaire :

I - Introduction

I.1. Présentation de ReST

ReST (ou **RESTful**) signifie **Re**presentational **S**tate **T**ransfer. Cette technologie permet de fournir à des **agents** un ensemble de **fonctionnalités** sous forme d'**URL** en exploitant le **protocole HTTP** :



Un **Web Service** est une **application réseau** qui offre un ensemble de **services** entre hôtes **hétérogènes** sur **internet**. Un **agent** vient solliciter le **fournisseur** du service via une **requête**, les deux parties adoptent le même **formalisme** pour **l'échange** des **informations**.

La **réponse** fournie par le **Web Service** se présente au format **XML** ou **JSON**. Afin de préciser le **statut** du **résultat** de la **requête**, les **codes** de **retour HTTP** sont également employés.

Hormis sa simplicité, un **Web Service ReST** permet de s'affranchir de **certains problèmes** liés à la présence de **proxy** (*ports non autorisés, ...*) car il utilise le protocole **HTTP**.

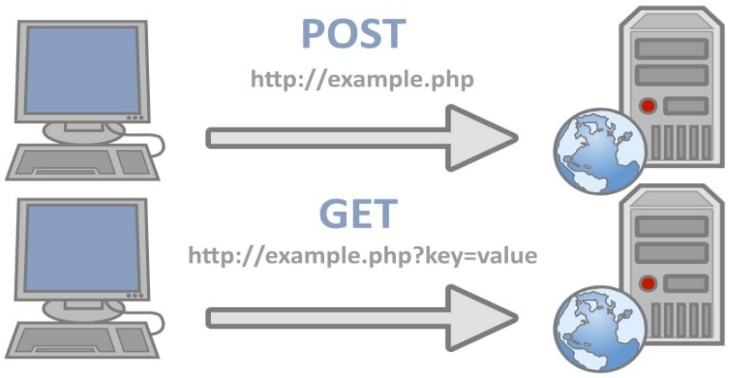
I.2. Les commandes HTTP utilisées par ReST

Le protocole **HTTP** permet généralement à un **client** (*navigateur*) de solliciter une **URL** en envoyant une **requête** à un **serveur**. Par exemple, si l'URL **www.gcrampe.fr** est saisie dans la **barre d'adresse** d'un **navigateur** et ensuite **validée**, une **requête** de type **GET** est envoyée au **serveur HTTP**.

Un **Web Service ReST** se base sur **4 types de requêtes HTTP** :

- **GET** : *Obtention d'une donnée stockée sur le serveur depuis le web service ;*
- **POST** : *Mise à jour d'une donnée stockée sur le serveur depuis le webservice ;*
- **PUT** : *Création d'une donnée sur le serveur depuis le web service ;*
- **DELETE** : *Effacement d'une donnée stockée sur le serveur depuis le webservice.*

Remarque *: On utilise en général uniquement les méthodes **GET** et **POST** car les méthodes **PUT** et **DELETE** peuvent être bloquées par certains pare-feux.*



La sollicitation d'un **Web Service ReST** est accompagnée d'un code de **retour http** permettant de **spécifier** le **résultat** de l'opération demandée. Les **codes** les plus couramment rencontrés sont les suivants :

	Code	Méthode	Signification
Succès	*200*	*GET, PUT, DELETE*	**[OK]** : Opération réussie.
Erreurs client	*404*	*GET, POST, PUT, DELETE*	**[NOT FOUND]** : Ressource non trouvée.
	405	*GET, POST, PUT, DELETE*	**[NOT ALLOWED]** : Méthode non autorisée sur la ressource
Erreur serveur	*500*	*GET, POST, PUT*	**[INTERNAL SERVER ERROR]** : Erreur interne au serveur.

I.3. Méthodes d'identification d'une API ReST

Un Web service Web ReST doit authentifier les requêtes avant de pouvoir envoyer une réponse. L'authentification est le processus de vérification d'une identité.

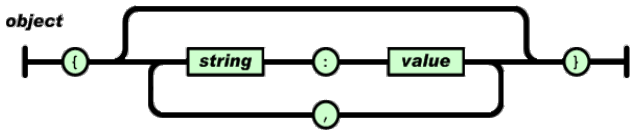
Les **clés API** sont une option simple pour l'authentification ReST. Dans cette approche, le serveur attribue une valeur générée unique à un client qui se présente pour la première fois. Chaque fois que le client essaie d'accéder aux ressources, il utilise la clé API unique pour se vérifier. Les clés API sont peu sûres, car le client doit transmettre la clé, ce qui la rend vulnérable au vol sur le réseau.

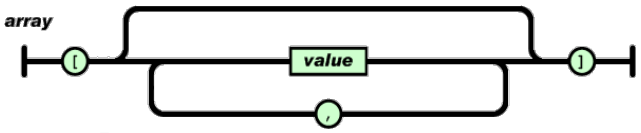
I.4. La réponse du Web Service ReST

Le corps de la réponse contient la représentation de la ressource. Le serveur sélectionne un format de représentation approprié en fonction de ce que contiennent les en-têtes de la demande. Les clients peuvent demander des informations dans les formats **XML** ou **JSON**, qui définissent la manière dont les données sont écrites en texte clair.

Par exemple, si le client demande la valeur et la date d'une acquisition, le serveur renvoie une représentation **JSON** comme suit : '**{"valeur":"63", "date":"2022-06-23 10:03:47"}**'

Remarque ***: JSON*** *(**J**ava**S**cript **O**bject **N**otation) est un format léger d'échange de données.* ***JSON*** *se base sur 2* ***structures*** *: l'****objet*** *ou le* ***tableau****.*






```
{
      "Prenom": "Didier",
      "Nom": "DUPOND",
      "Adresse":
}
```

```
[
   {
     "Prenom": "Didier",
     "Nom": "DUPOND"
   },
   {
     "Prenom": "Pascal",
     "Nom": "DURAND"
   }
]
```

II - Réalisation d'un Web Service Rest en PHP

II.1. Objectif

L'objectif est de récupérer par le biais d'un **Web Service ReST** développé en **PHP** une **date** et une **mesure** (comprise entre **0** et **100**), depuis un **agent** (*client*).

L'agent sollicitera le **Web Service** via la **méthode GET** (*tout autre méthode entraînera une erreur* ***405***).

L'information reçue est un objet **JSON** (**J**ava**S**cript **O**bject **N**otation) qui aura la forme suivante :

{"valeur":"80","date":"2022-06-23 10:03:47"}.

II.2. Web Service coté serveur

Le **Web Service ReST** est codé en **PHP** (*fichier* ***ws.php***) :

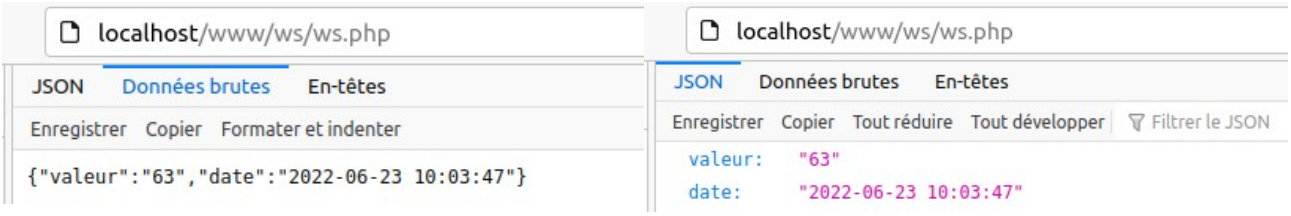
```
<?php
header('Cache-Control: no-cache, must-revalidate');
header('Content-type: application/json');
date_default_timezone_set('Europe/Paris');
if($_SERVER['REQUEST_METHOD']!= 'GET')
      {
      header("HTTP/1.1 405 NOT ALLOWED");
      }
```

```
else {
      header("HTTP/1.1 200 OK");
      $resultat = strval(rand(0,100));
      $date = date('Y-m-d H:i:s');
      $retour=array("valeur"=>$resultat,"date"=>$date);
      echo json_encode($retour);
      }
?>
```

Remarque *: la fonction* **`header('Content-type: application/json')`** *en* ***php*** *permet de spécifier l'en-tête du fichier HTML retourné ici* ***json****, selon la spécification de ce protocole.*

II.3. Web Service coté client - Agent

On peut valider le **Web Service** précédemment développé, à l'aide d'un navigateur :



En langage **C**, afin de développer un **agent** de **Web Service** on peut utiliser la libraire **curl**. Le code de l'agent est celui-ci :

```
#include <stdio.h>
#include <curl/curl.h>
int main(void)
{
CURL *curl;
CURLcode res;
curl = curl_easy_init();
if(curl) {
      curl_easy_setopt(curl, CURLOPT_URL, "http://localhost/www/ws/ws.php");
      // Si on veut traverser le proxy:
      curl_easy_setopt(curl,CURLOPT_PROXY,"http://user:mdp@172.17.2.254:3128");
      curl_easy_setopt(curl,CURLOPT_NOPROXY,"192.168.X.4");
      res = curl_easy_perform(curl);
      if(res != CURLE_OK)
      fprintf(stderr, "curl_easy_perform() failed: %s\n",
            curl_easy_strerror(res));
      curl_easy_cleanup(curl);
      }
return(0);
}
```

Résultat :

```
{"valeur":"80","date":"2022-06-23 10:03:47"}
```