

# 6.

## Développement Client/Serveur : Services Web

6.1 Services Web

6.2 Architectures

6.3 Services Web REST

6.4 Services Web SOAP

---

# Types d'applications mobiles

- **Application native** : conçue spécialement pour le système cible (Android, iOS...)
  - Meilleure rapidité, fiabilité et dotée d'une meilleure réactivité ainsi qu'une résolution supérieure ce qui assure une meilleure expérience utilisateur.
  - Elle permet un accès plus facile à toutes les fonctionnalités du téléphone, de l'accéléromètre en passant par la caméra et même le micro.
  - Les notifications push, uniquement disponibles sur les apps natives. Ces notifications vous permettent d'alerter vos utilisateurs et d'attirer leur attention chaque fois que vous le souhaitez. (voir GCM/FCM)
  - Ne consomme pas beaucoup de ressources réseaux.
  - Ne requiert pas forcément internet pour fonctionner, ce qui est un réel avantage. Même aujourd'hui, il existe encore des zones très peu couvertes par le réseau internet, et permettre à ses utilisateurs d'accéder à l'app sans connexion web est un très gros point fort à ne pas négliger.
- **Application Web** : pages web adaptées au Mobile (Mobile Friendly UI)
  - Avantages : Moins coûteux / une seule technologie à utiliser. (JavaScript, Php, CSS..)
  - Inconvénients : Pas d'accès aux ressources matériels du téléphone.
- **Application hybride** : entre les deux : application spécifique à chaque plateforme générée de manière plus ou moins automatique en utilisant des technologies Web (HTML, CSS, JS..)
  - Avantage: coût du développement des pages Web. Applications pour # OS
  - Exemple d'outils: Cordova, PhoneGap, Ionic, Xamarin...



# Architecture logicielle distribuée

Les services Web fournissent un lien entre applications. Ainsi, des applications utilisant des technologies différentes peuvent envoyer et recevoir des données au travers de protocoles compréhensibles par tout le monde.

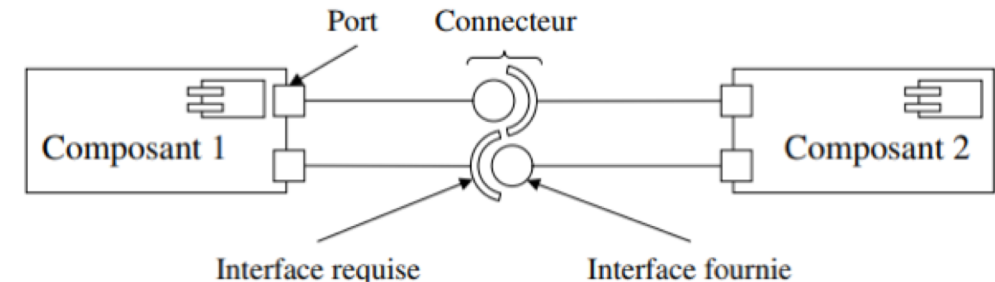
Cet ensemble d'applications en apparence **autonomes** mais en réalité dépendantes les unes des autres constituent une application **distribuée**.

Les services Web sont le moyen de communication principal entre composantes distantes d'une application distribuée. Ils utilisent des standards (XML, HTTP,...) pour **transférer des données** et ils sont compatibles avec de nombreux autres environnements de développement. **Ils sont donc indépendants des plates-formes.** C'est dans ce contexte qu'un intérêt très particulier a été attribué à la conception des services Web puisqu'ils permettent aux entreprises d'offrir des applications accessibles à distance par d'autres entreprises.



# Architecture distribuée et composants

- La notion de composant logiciel est introduite comme une suite à la notion d'objet.
- Le composant offre une meilleure structuration de l'application et permet de construire un système par assemblage de briques élémentaires en favorisant la réutilisation de ces briques.
- Une **application distante** peut être modélisée comme un composant ou un ensemble de composants d'un système distribué. (mais peut aussi faire partie d'un composant dans certains cas)



# 6.1 Services Web

Les services Web sont des services de **traitement de la donnée exposée sur internet**. Ils peuvent avoir plusieurs formes, provenir de plusieurs sites différents, faire des tâches diverses et être privés ou publics.

Exemples :

- données météorologiques ;
- calcul mathématique ;
- stockage d'informations ;
- etc.

Les services Web peuvent être codés en plusieurs langages (C#, Java, PHP, Python, C, etc.).

## **Citation Dico du Net :**

“C’est une technologie permettant à des applications de dialoguer à distance via Internet indépendamment des plates-formes et des langages sur lesquels elles reposent.”

Les services web ne sont pas une exclusivité du développement mobile.



## 6.2. Architectures

Les principales architectures des Web-Services sont **REST** et **SOAP**.

REST et SOAP sont souvent comparés l'un à l'autre, mais c'est une erreur. En effet, ces éléments ne sont pas d'un même type : SOAP est à la fois une architecture et un protocole tandis que REST est un style d'architecture.

La différence majeure entre ces 2 “architectures” est le degré de liaison entre le client et le serveur. Un client développé avec le protocole SOAP ressemble à un programme locale (instanciation, appel de méthodes...), car il est

étroitement lié au serveur. Si une modification est effectuée d'un côté ou de l'autre, l'ensemble peut ne plus fonctionner. Il faut effectuer des mises à jour du client s'il y a des changements sur le serveur et vice-versa.

Avec REST il y a beaucoup moins de couplage entre le client et le serveur : un client peut utiliser un service de type REST sans aucune connaissance de l'API. A l'inverse, un client SOAP doit tout savoir des éléments qu'il va utiliser pendant son interaction avec le serveur, sinon cela ne fonctionnera pas.



## 6.3 Architectures REST

Les applications clientes Android peuvent tirer partie d'une architecture de type REST car de nombreuses technologies côté serveur sont disponibles. Les principes de REST :

- Les données sont stockées dans le serveur permettant au client de ne s'occuper que de l'affichage
- Chaque requête est *stateless* c'est à dire qu'elle est exécutable du côté serveur sans que celui-ci ait besoin de retenir l'état du client (son passé i.e. ses requêtes passées)
- Les réponses peuvent parfois être mises en cache facilitant la montée en charge
- Les ressources (services) du serveur sont clairement définies; l'état du client est envoyé par "morceaux" augmentant le nombre de requêtes.

Pour bâtir le service du côté serveur, un serveur PHP ou Tomcat peut faire l'affaire. Il pourra fournir un web service, des données au format XML ou JSON. Du côté client, il faut alors implémenter un parseur SAX ou d'objet JSON



# REST + JSON

Nous allons nous intéresser plus précisément aux services de type REST retournant des données au format JSON (JavaScript Object Notation).

Pourquoi REST? Avoir un grand degré de liberté et une indépendance entre programmes/APIs client et serveur.

Pourquoi le JSON? Pour une question de facilité, bien entendu on peut utiliser le XML qui est aussi facile, mais le JSON a la particularité d'être plus léger que XML, un gain de vitesse n'est pas anodin dans le domaine de la mobilité.





## 6.3.1 Exemple serveur : PHP -> JSON

Exemple en PHP:

```
$json=array();
$sql="SELECT phone,name,isConnected FROM contacts ORDER BY name ASC";
try{
    $res=$db->query($sql);
}catch(Exception $e){
    die(json_encode(array('error'=>$db->errorInfo())));
}
if($res){
    while($clt = $res->fetch(PDO::FETCH_ASSOC)){
        $json[]=$clt;
    }
    $res->closeCursor();
}
echo json_encode($json);
```



# Exemple Serveur: PHP -> XML

## PHP :

```
$array=array();  
// $array doit être  
associatif  
echo array2xml($array);
```

```
function array2xml($array, $node_name="root") {  
    $dom = new DOMDocument('1.0', 'UTF-8');  
    $dom->formatOutput = true;  
    $root = $dom->createElement($node_name);  
    $dom->appendChild($root);  
    $array2xml = function ($node, $array) use ($dom, &$array2xml) {  
        foreach($array as $key => $value){  
            if ( is_array($value) ) {  
                $n = $dom->createElement($key);  
                $array2xml($n, $value);  
            }else  
                $n = $dom->createElement($key,$value);  
            $node->appendChild($n);  
        }  
    };  
  
    $array2xml($root, $array);  
    return $dom->saveXML();  
}
```



# Connexion du client (A)

```
URL url = new URL("http://www.monservice.com/interface.json?param1=jksq");
URLConnection conn = (URLConnection) url.openConnection();
try {
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    conn.setChunkedStreamingMode(0);

    OutputStream out = new DataOutputStream(conn.getOutputStream());
    // On peut utiliser out pour transmettre les paramètres et toute autre métadonnée

    InputStream inStream = new BufferedInputStream(conn.getInputStream());
    String result = readStream(inStream); //passer de InputStream à String
} finally {
    conn.disconnect();
}
```

Remarque:  
Il faut  
aussi  
demander  
les  
permissions  
nécessaires  
lors de  
l'exécution  
(en plus de  
celles du  
Manifest)



# Stream to String

- Lecture du contenu depuis le stream : fonction `readStream(...)` dans l'exemple précédant

```
BufferedReader r = new BufferedReader(new InputStreamReader(inStream));
```

```
StringBuilder sb= new StringBuilder();
```

```
String line = null;
```

```
try {
```

```
    while ((line = r.readLine()) != null) {
```

```
        sb.append(line).append('\n');
```

```
    }...
```

```
String result = sb.toString();
```



# NetworkOnMainThreadException

- Depuis Android 3.0, on ne peut pas effectuer des opérations de réseau dans le thread d'une activité ou d'une interface graphique en général.
- On peut le mettre dans un thread traditionnel mais la communication entre un thread et une interface graphique est un peu difficile.
- La méthode la plus adéquate pour remédier à cela est d'utiliser une tâche asynchrone (AsyncTask) qui permet facilement de gérer à la fois le code à exécuter en arrière plan (connexion..) et le code à exécuter dans le thread principal (affichage des résultats).
- Voir chapitre "Programmation concurrente"



# Connexion avec l'API Volley (B)

*// un singleton Volley pour la gestion de la queue de requêtes*

```
if (mRequestQueue == null) {  
    mRequestQueue = Volley.newRequestQueue(this(getApplicationContext()));  
}
```

```
StringRequest request = new StringRequest(Request.Method.GET, "http://www.monserveur.com/service.json", null, new  
MyResponseListener(this), new MyErrorResponseListener());
```

```
mRequestQueue.add(request);
```

```
class MyErrorResponseListener implements Response.ErrorListener{
```

```
    @Override
```

```
    public void onErrorResponse(VolleyError error) {
```

```
        Log.e(TAG, error.getMessage());
```

```
    }
```

```
}
```

```
private class MyResponseListener implements Response.Listener<String> {
```

```
    @Override
```

```
    public void onResponse(String data) {
```

```
        //Convertir String data en objet JSON pour récupérer les informations transmises
```

```
    }
```

```
}
```



## 5.3.3 Récupération des objets : JSON

```
// Création d'un objet JSON depuis un String
JSONObject j = new JSONObject(result);
// On peut vérifier l'existence de chaque clé et récupérer sa valeur directement
if(j.has("error"))
    Toast.makeText(MainActivity.this,j.getString("error"),Toast.LENGTH_LONG).show();
else{
    Student std=new Student(j.getLong("id"),j.getString("nom"),
        j.getString("prenom"),j.getString("classe"));
    std.setPhone(j.getString("phone"));
    if(j.has("notes")){
        JSONArray ja=j.getJSONArray("notes");
        for(int i=0; i<ja.length();i++){
            JSONObject jo=ja.getJSONObject(i);
            std.addNote(
                new Note(jo.getString("label"),jo.getDouble("score"))
            );
        }
    }
}
```

Une fois le texte est récupéré du serveur sous forme de String, on le convertit en objets JSON



## 6.4. Services Web de type SOAP

Les services Web de type SOAP permettent une communication très rapprochée entre le client et le serveur. Les deux parties de l'application sont très liées ce qui en fait à la fois un avantage et un inconvénient.

Le SDK Android ne comporte pas d'API spécifique pour les architectures SOAP, mais il existe plusieurs APIs Open Source exploitables à cet effet : la plus utilisée s'appelle **kSOAP2**. On peut télécharger et compiler la source avec notre projet ou inclure tout simplement le .jar adéquat comme "dependency".

Détails sur l'API et téléchargement :  
<https://code.google.com/archive/p/ksoap2-android/>

### Le WSDL (Web Services Description Language)

Le WSDL est une description fondée sur le XML qui indique comment utiliser le service.

Le WSDL sert à décrire :

- le protocole de communication;
- le format de messages requis pour communiquer avec ce service ;
- la définition des méthodes qu'il est possible d'appeler ;
- la localisation du service.

Une description WSDL est un document XML qui commence par la balise "definitions" et qui contient les balises suivantes :

- "binding" : définit le protocole à utiliser pour invoquer le service web ;
- "port" : spécifie l'emplacement actuel du service ;
- "service" : décrit un ensemble de points finaux du réseau.





# kSOAP: exemple d'implémentation

```
public class AppelService {
    private static final String NAMESPACE = "http://mon-site-web.fr";
    private static final String URL = "http://mon-exemple-web-services/wsdl.WSDL";
    private static final String SOAP_ACTION = "getMeteo";
    private static final String METHOD_NAME = "getMeteo";
    private String getMeteo(String ville) {
        String meteo = null;
        try {
            SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
            request.addProperty("ville", ville);

            SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
            envelope.setOutputSoapObject(request);

            AndroidHttpTransport androidHttpTransport = new AndroidHttpTransport(URL);
            androidHttpTransport.call(SOAP_ACTION, envelope);
            SoapObject objetSOAP = (SoapObject)envelope.getResponse();
            meteo = this.parserObjet(objetSOAP);

        } catch (Exception e) {
            Log.e("getMeteo", "", e);
        }
    }
    private String parserObjet(SoapObject objet) {
        SoapObject meteoObjet = (SoapObject)positionSoap.getProperty("meteo");
        String meteo = meteoObjet.getProperty("temps").toString();

        return meteo;
    }
}
```

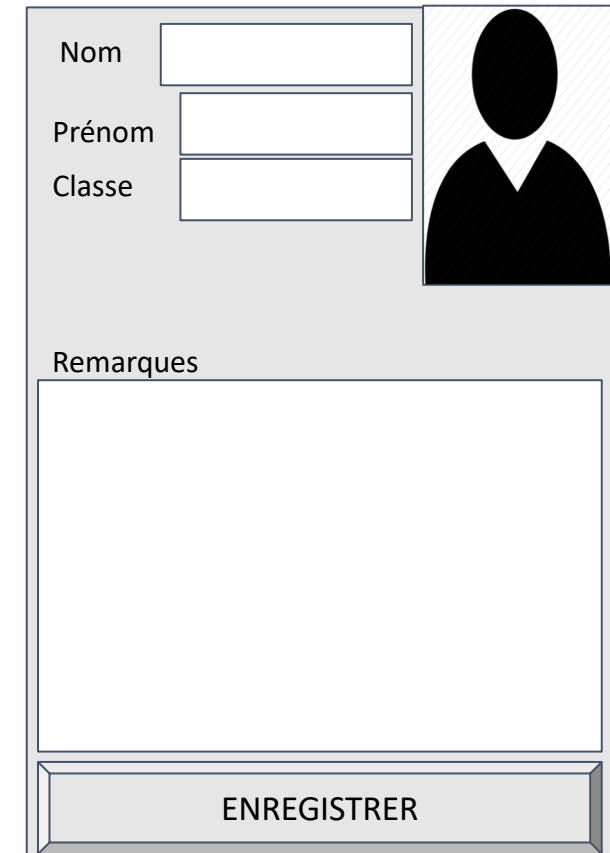


# Exercice : consommation d'un webservice REST au format JSON

- En utilisant l'interface graphique réalisée précédemment dans le chapitre 3, et en utilisant le service web hébergé à l'URL suivante :

<https://belatar.name/rest/profile.php?login=test&passwd=test&id=xxxx>

- développez le code qui permet de consommer le webservice et d'afficher les données sur l'interface graphique dès l'ouverture de l'activité en utilisant l'API **VOLLEY**,
  - Id = 9998
  - déclarez les autorisations nécessaires et faites les vérifications nécessaires surtout pour l'état de la connectivité,
  - téléchargez l'image de profile et affichez la dans l'ImageView,
- Que doit-on faire pour enregistrer les modifications du profile?



The image shows a user profile form interface. It has a light gray background. On the right side, there is a placeholder for a profile picture, represented by a black silhouette of a person's head and shoulders. To the left of this, there are three input fields stacked vertically, labeled 'Nom', 'Prénom', and 'Classe'. Below these fields is a large text area labeled 'Remarques'. At the bottom of the form, there is a button labeled 'ENREGISTRER'.

