

7.

Développement Client/Serveur : Services Web

7.1 Services Web

7.2 Architectures

7.3 Services Web REST

7.4 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 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)
 - Avantage : une seule technologie à utiliser.
- 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.
 - Exemple d'outils: Cordova, PhoneGap, Ionic...

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.

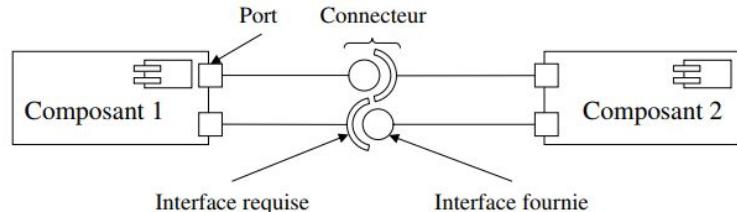
Cet ensemble d'applications en apparence autonomes mais en réalité dépendantes les unes des autres constituent une application distribuée.

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)



7.1 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, Ruby, 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.



7.2. 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.



7.3 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 sont bien résumés sur [la page wikipedia](#) leurs étant dédiée:

- 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. Par contre XML a l'avantage de la multitude des Parsers (DOM, SAX, PULL..)



7.3.1 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();
}
```



Exemple Serveur: Spring -> JSON

```
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import
org.springframework.web.bind.annotation.RestController;
```

```
@RequestMapping("/api")
public class RestApiController {

    @RequestMapping(value = "/user/", method = RequestMethod.GET)
    public ResponseEntity<List<User>> listAllUsers() {
        List<User> users =findAllUsers();
        if (users.isEmpty()) {
            return new ResponseEntity(HttpStatus.NO_CONTENT);
        }
        return new ResponseEntity<List<User>>(users, HttpStatus.OK);
    }
}
```

- exemple complet :
<http://websystique.com/spring-boot/spring-boot-rest-api-example/>

```
@RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
public ResponseEntity<?> getUser(@PathVariable("id") long id) {
    logger.info("Fetching User with id {}", id);
    User user =findUserById(id);
    if (user == null) {
        logger.error("User with id {} not found.", id);
        return new ResponseEntity(new CustomErrorType("User with id " + id
+ " not found"), HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<User>(user, HttpStatus.OK);
}
```



Connexion du client (A)

```
URL url = new URL("http://www.monservice.com/interface.json");
HttpURLConnection conn = (HttpURLConnection) 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 précédent

```
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 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", 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
json = new JSONObject(result);
Log.i(TAG,json.toString());

// Si on ne connaît pas les clés (names), on peut les lister
JSONArray nameArray=json.names();
JSONArray valArray=json.toJSONArray(nameArray);
for(int i=0;i<valArray.length();i++){
    Log.i(TAG,nameArray.getString(i)+" : " + valArray.getString(i)+"\n");
}

// Sinon on peut vérifier l'existence de chaque clé et récupérer sa valeur directement
if(json.has("error"))
    Toast.makeText(MainActivity.this,json.getString("error"),Toast.LENGTH_LONG).show();
```

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



XML

Sans surprise, Android fournit plusieurs parsers XML (*Pull parser, Document parser, Push parser*). SAX et DOM sont disponibles. L'API de *streaming* (StAX) n'est pas disponible. Cependant, une librairie équivalente est disponible: **XmIPullParser**. Voici un exemple simple:

```
//on peut aussi le lire à partir d'un fichier ou d'un service web

String result = new String("<plop><blup attr=\"45\">Coucou
!</blup></plop>");
InputStream f = new ByteArrayInputStream(s.getBytes());
XmlPullParser parser = Xml.newPullParser();
try {
    parser.setInput(f, null);

    parser.next();
    Toast.makeText(this, parser.getName(), Toast.LENGTH_LONG).show();
    parser.next();
    Toast.makeText(this, parser.getName(), Toast.LENGTH_LONG).show();
    Toast.makeText(this, parser.getAttributeValue(null, "attr"),
                  Toast.LENGTH_LONG).show();
    parser.nextText();
    Toast.makeText(this, parser.getText(), Toast.LENGTH_LONG).show();
} ...
```



Exemple: DOM Parser

Enfin, le parser DOM permet de naviguer assez facilement dans la représentation arborescente du document XML. Ce n'est évidemment pas préconisé si le XML en question est volumineux. L'exemple suivant permet de chercher tous les tags "monitor" dans les sous-tags de la racine.

```
InputSource source = new InputSource(stream);

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document dom = builder.parse(source);
    Element root = dom.getDocumentElement();
    // Récupère tous les tags du descendant de la racine s'appelant monitor
    NodeList items = root.getElementsByTagName("monitor");
    for (int i=0;i<items.getLength();i++){
        Node item = items.item(i);
        // Traitement:
        // item.getNodeName()
        // item.getTextContent()
        // item.getAttributes()
        ...
    }
} catch (...)
```



7.4. 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.

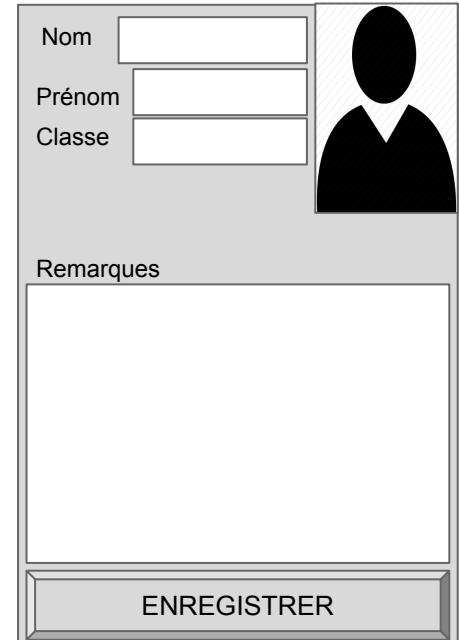


kSOAP: exemple d'implémentation

```
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)objet.getProperty("meteo");  
        String meteo = meteoObjet.getProperty("temp").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 :
<http://belatar.name/tests/profile.php?login=test&passwd=test>
 - 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 HttpURLConnection,
 - déplacez le traitement de la connexion dans une tâche asynchrone,
 - 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,
- Refaire le travail en utilisant l'API Volley
- Que doit-on faire pour enregistrer les modifications du profile?



The form consists of several input fields: 'Nom' (Name) with a text input field, 'Prénom' (First Name) with a text input field, and 'Classe' (Class) with a text input field. To the right of these fields is a black silhouette of a person's head and shoulders. Below these fields is a large text area labeled 'Remarques' (Remarks). At the bottom is a wide, rectangular button labeled 'ENREGISTRER' (REGISTER).

8. **Programmation concurrente**

8.1 Présentation

8.2 Tâches concurrentes

8.3 Les services

8.1 Présentation

Une application Android n'est pas qu'une simple activité. Dans de nombreux cas, une application Android est amenée à tirer partie de la programmation concurrente afin de réaliser des tâches parallèles.

Dans ce chapitre, on s'intéresse à la notion de processus, nous verrons comment réaliser des tâches particulières qui sont souvent concurrentes à l'activité principale. Ces tâches s'appuient principalement sur les *threads/runnables* comme vous les connaissez dans JAVA et autres plate-formes, mais aussi sur de nouveaux composants basés sur les threads comme les "services" et les tâches asynchrones.

8.2 Tâches concurrentes

En plus des classes de programmation concurrentes de Java (**Thread**, **Executor**, **ThreadPoolExecutor**, **FutureTask**, **TimerTask**, etc...), Android fournit quelques classes supplémentaires pour programmer des tâches concurrentes (**AsyncTask**, **IntentService**, etc...).

Si la tâche concurrente est lancée:

- par l'activité principale: sa vie durera le temps de l'activité
- par un service: la tâche survivra à l'activité principale

Tâche régulière

S'il s'agit de faire une tâche répétitive qui ne consomme que très peu de temps CPU à intervalles réguliers, une solution consiste à programmer une tâche répétitive à l'aide d'un `TimerTask`.

```
task = new TimerTask() {  
    public void run() {  
        //Do something  
    }};  
timer.schedule(task, 0, 5000);
```

Si le traitement à effectuer affecte l'interface graphique, il faut déléguer l'exécution à l'UI Thread pour exécution différée (pour éviter de bloquer l'UI).

La documentation donne les quelques méthodes utiles pour ces cas délicats:

- `Activity.runOnUiThread(Runnable)`
- `View.post(Runnable)`
- `View.postDelayed(Runnable, long)`

Tâches Asynchrones

Pour gérer par exemple le chargement d'images ou de données à partir d'un serveur, il faut opter pour une solution asynchrone afin d'améliorer la fluidité d'une application; sinon on risque rapidement de voir planter l'application.

Pour réaliser cela, il faut se baser sur la classe **AsyncTask<U,V,W>** basée sur 3 types génériques:

- U: le type du paramètre envoyé à l'exécution
- V: le type de l'objet permettant de notifier de la progression de l'exécution
- W: le type du résultat de l'exécution

Un autre exemple consistant à charger une image peut avoir la signature suivante :

AsyncTask<String, int, Bitmap>

Dans un exemple où nous serons amenés à charger une liste de Personnes depuis un service Web, nous pouvons faire une implémentation utilisant 3 paramètres: l'url du service Web (String), un Void représentant les étapes d'avancement de notre tâche (nous ne nous intéressons pas à la progression, sinon on peut utiliser un int) et la classe **ArrayList<Personne>** renvoyant la liste des contacts chargée, soit la classe **AsyncTask<String, Void, ArrayList<Personne>>**.

Les méthodes de AsyncTask

onPostExecute(W): invoquée quand la tâche est terminée et qu'il faut mettre à jour l'UI avec le résultat calculé dans l'objet de la classe **W**.

```
protected void onPostExecute(Bitmap bitmap) {  
    if (imageView != null) {  
        if (imageView != null) {  
            imageView.setImageBitmap(bitmap);  
        }  
    }  
}
```

Pour lancer l'exécution de notre tâche asynchrone, il faut en créer une instance et appeler la méthode **execute(U)** . Dans le cas des webservice REST par exemple le type U correspond à un String (URL+paramètres)

C'est cette méthode qui prendra un certain temps à s'exécuter. L'objet reçu en paramètre est de type **U** et permet de gérer la tâche à accomplir. Dans notre exemple, l'objet est l'url où télécharger l'image. A la fin, la tâche doit renvoyer un objet de type **W**.

```
protected Bitmap doInBackground(String... params) {  
    String url = params[0];  
    publishProgress(new Integer(0));  
    DefaultHttpClient client = new DefaultHttpClient();  
    HttpGet getRequest = new HttpGet(url);  
    HttpResponse response = client.execute(getRequest);  
    publishProgress(new Integer(1));  
    ... }
```



Les méthodes de AsyncTask

void onProgressUpdate(V...): invoquée dans le thread qui gère l'UI, juste après que la méthode **doinBackground(U...)** ait appelé **publishProgress(V...)**. On reçoit donc l'objet V qui contient les informations permettant de mettre à jour l'UI pour notifier de la progression de la tâche.

```
protected void onProgressUpdate(Integer...  
values) {  
    Integer step = values[0];  
    if (textView != null) {  
        textView.setText( step.toString() + "%" );  
    }  
}
```

void onPreExecute(): invoquée juste après que la tâche soit démarrée. Elle permet de modifier l'interface graphique juste après le démarrage de la tâche. Cela permet de créer une barre de progression ou de charger un élément par défaut. Cette méthode sera exécutée par l'*UI thread*.

```
protected void onPreExecute() {  
    if (imageView != null) {  
        imageView.setImageResource(R.drawable.interro);  
    }  
}
```

W doInBackground(U...): invoquée dans le thread qui s'exécute en tâche de fond, une fois que **onPreExecute()** est terminée.

