

# Systeme mobiles

## Compte rendu de TP n°6 :

Requêtes HTTP



## Introduction :

Dans ce TP nous allons créer un client et mettre en place une requête **GET HTTP** dans notre application. Nous allons également faire une requête sur un service web et parser la réponse afin de l'afficher.

### 1. Requête HTTP :

Afin de pouvoir afficher le texte de la page web nous utilisons la fonction *readStream* qui va venir remplir un *TextView* avec ces informations. Son implémentation est la suivante:

```
private void readStream(InputStream inputStream) {  
    Scanner s = new Scanner(inputStream).useDelimiter("\\A");  
    String result = s.hasNext() ? s.next() : "";  
    TextView textView = findViewById(R.id.textview);  
    textView.setText(result);  
}
```

Lorsque l'on teste l'application on remarque qu'elle ne fonctionne pas, l'exception "android.os.StrictMode\$AndroidBlockGuardPolicy.onNetwork" est levée. En effet, il est déconseillé de faire des requêtes HTTP directement dans le thread de l'activité.

```
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: android.os.NetworkOnMainThreadException  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at android.os.StrictMode$AndroidBlockGuardPolicy.onNetwork(StrictMode.java:1147)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at java.net.InetAddress.lookupHostName(InetAddress.java:418)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at java.net.InetAddress.getAllByNameImpl(InetAddress.java:252)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at java.net.InetAddress.getAllByName(InetAddress.java:215)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.HostResolver$1.getAllByName(HostResolver.java:29)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.RouteSelector.resetNextInetSocketAddress(RouteSelector.java:232)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.RouteSelector.next(RouteSelector.java:124)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.HttpEngine.connect(HttpEngine.java:272)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.HttpEngine.sendRequest(HttpEngine.java:211)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.HttpURLConnectionImpl.execute(HttpURLConnectionImpl.java:373)  
10-03 15:35:40.766 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.HttpURLConnectionImpl.getResponse(HttpURLConnectionImpl.java:323)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at com.android.okhttp.internal.http.HttpURLConnectionImpl.getInputStream(HttpURLConnectionImpl.java:190)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at com.example.polytech.tp6http.Fenetre1.onClick(Fenetre1.java:43)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at android.view.View.performClick(View.java:4757)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at android.view.View$PerformClick.run(View.java:19757)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at android.os.Handler.handleCallback(Handler.java:739)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at android.os.Handler.dispatchMessage(Handler.java:95)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at android.os.Looper.loop(Looper.java:135)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at android.app.ActivityThread.main(ActivityThread.java:5219) <2 internal calls>  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:898)  
10-03 15:35:40.775 948-948/com.example.polytech.tp6http W/System.err: at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:693)
```

Pour éviter cette erreur nous devons autoriser les requête HTTP dans notre thread en modifiant sa *policy*. On le spécifie de la façon suivante:

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();  
StrictMode.setThreadPolicy(policy);
```


De plus, nous devons aller modifier le manifest pour que l'application puisse accéder à internet. Nous pouvons l'y autoriser en ajoutant la ligne suivante dans notre manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

## 2. Ajout de la classe CallWebAPI :

Une méthode plus recommandée est de créer une classe spécialement pour cela, ainsi la bonne méthode est de créer une tâche asynchrone s'exécutant en tâche de fond puis une fois la requête effectuée transmettre le résultat au thread de la vue graphique pour affichage. Dès que cette tâche aura reçue le résultat il sera affiché dans la vue de l'application. Cela évite le blocage de l'application si le service HTTP prend trop de temps à répondre.

De plus il est possible de passer en paramètre l'URL du site vers lequel on souhaite exécuter la requête pour cela il est nécessaire de faire appel au paramètre *params[0]* de la méthode *doInBackground* qui contient l'URL passé en paramètres lors de l'appel à la méthode *execute()* de la classe *CallWebAPI*.

A screenshot of an Android emulator window titled "TP6Http". The status bar at the top shows the time as 15:38. The main content area displays raw HTML code instead of a rendered web page. The code includes a DOCTYPE declaration, a head section with meta tags for viewport, telephone, and format-detection, and a title "Google". The body contains a search bar with a cancel button. The code is not properly escaped, showing raw tags like <div> and <input>.

```
<!doctype html><html lang="fr"> <head> <meta
content="width=device-width,minimum-scale=1.0"
name="viewport"><meta content="telephone=no" name="format-
detection"><meta content=" RECHERCHE 10" name="format-
detection"> <title>Google</title> <style>._wtf{display:inline-
block;fill:currentColor;height:24px;line-height:
24px;position:relative;width:24px;._wtf svg{display:block;height:
100%;width:100%}</style> <style>.hp #sfcnt{margin:8px;margin-
top:160px}.srp #sfcnt{background:#fff;height:35px;padding-
bottom:6px;padding-right:8px;padding-top:7px;border-bottom:
1px solid #E5E5E5;padding-left:8px}.no_outline a,.no_outline
div{outline:none;}.msb{position:relative}.srp .msb{top:
1px}.msfo{padding-right:38px}.msfi{background-color:#fff !
important;border-color:#c7d6f7;border-style:solid;border-width:
2px 1px 2px 2px;border-right:none;padding:0;height:
38px;padding-right:0;border:1px solid #d9d9d9 !
important;border-right:none !important;border-top:1px solid
silver !important;}.srp .msfo{overflow:hidden}form#tsf{margin-
left:auto;margin-right:auto;max-width:
736px;overflow:hidden;width:auto}.hp .msfi{margin-
top:-1px}.sb_ifc{display:-webkit-box;display:-webkit-
flex;display:flex;padding:5px 0}.sb_chc{display:-webkit-box;-
webkit-box-ordinal-group:1;-webkit-order:1;order:1;-webkit-box-
flex:0;flex:0 0 auto;margin:-5px 0}.lst.lst-
tbb{appearance:none;tap-highlight-color:rgba(0,0,0,0);display:-
webkit-box;-webkit-box-ordinal-group:2;-webkit-order:2;order:2;-
webkit-box-flex:1;-webkit-flex:1 0;flex:1 0;padding-left:
8px}.sb_ifc[dir='rtl'] .lst{padding-left:0;padding-right:8px}.msfi
input[type="search"]::search-cancel-
```

Cela correspond parfaitement au résultat de la requête, c'est le code HTML non parsé et affiché directement en dur.

## 3. Application à un web service:

Nous allons dans cette partie interroger un web Service et afficher la réponse de celui-ci. Il a été choisi d'utiliser un web service fournissant des informations sur une adresse IP qui lui est passée en paramètres.

Dans un premier temps il a été nécessaire de créer une nouvelle interface composée de trois champs. Un **textView** permettant d'afficher le résultat de la requête **GET**, un

**textEdit** permettant à l'utilisateur de saisir l'adresse IP et un bouton permettant de lancer la recherche.

Une fois les gestionnaire d'événements et les objets représentant les éléments graphiques mis en place lors d'un clic sur le bouton il était nécessaire de construire l'URL correspondant à la requête **GET**. Celle-ci est présentée sous la forme :

```
url = new URL(params[0]);
```

Une fois cette URL définie il faut faire appel à la classe *CallWebApi* avec celle-ci en paramètre. Nous avons choisi de ne passer que l'adresse IP en paramètre, l'adresse du service web est incluse en dur dans l'appel à l'URL.

```
button.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // callGoogle(); // To use the ugly technique.  
        CallWebAPI c = new CallWebAPI(textView);  
        c.execute("http://ip-api.com/xml/" + editText.getText());  
    }  
});
```

Ensuite la classe *CallWebApi* doit être modifiée de façon à ce que le résultat renvoyé par la requête est utilisé pour être envoyé à un parseur que nous avons écrit. Son utilisation est simple, on donne le code à parser et une valeur de recherche et on récupère le résultat. Nous avons choisi de faire ce dernier nous même, le code est simple et fonctionne parfaitement :

```
public static final String get(String code, String value) {  
    int i;  
    String code_tab[] = code.split("<");  
    i = 0;  
    while(!code_tab[i].startsWith(value))  
        i++;  
    return code_tab[i + 1].replace("![CDATA[", "").replace("]]>", "");  
}
```

Toutes les valeurs parsées sont envoyées au constructeur de la classe *GeolP*, ainsi on construit un objet de ce dernier ce qui est très pratique si par exemple nous souhaitons envoyer ces données à une autre activité ou même ajouter des fonctionnalités. Le résultat sera affiché dans la vue par l'intermédiaire du *textView* et d'une redéfinition de la méthode *toString()*. Le résultat obtenu est le suivant :



Ici par exemple l'adresse IP interrogée est l'adresse 8.8.8.8 située aux Etats-Unis (ce sont les serveurs DNS de Google). La requête s'étant bien déroulée le code reçu est un code de réussite.

## Conclusion :

Ce TP nous a permis de découvrir les web services et comment communiquer avec eux par le biais de requête HTTP. Grâce à cela nous sommes en mesure d'utiliser des services ayant déjà été créés par d'autres personnes sans avoir à reproduire tout ce qui a déjà été fait. Nous pouvons également utiliser internet dans nos applications avec les requêtes **HTTP**.