LAIGLE Sarah
LAHOUGUE Lucas

# Compte rendu des TP1 et 2 - SRIO

2021/2022

UNIVERSITÉ DE RENNES 1

esir ECOLE SUPERIEURE D'INGENIEURS DE RENNES

# TP1 : Développer un application hybride

Ce premier TP a pour but de nous familiariser avec le composant WebView. Dans celui-ci, on utilise l'émulateur Pixel 5.

## Partie I : Simple application avec WebView

Après avoir créé une simple application, on crée une instance de WebView dans le fichier activity_main.xml de celle-ci. C'est depuis ce fichier qu'on contrôle le visuel de notre application. On commence par enlever la TextView initialement créée car on n'en a pas besoin. On la remplace par le code suivant :

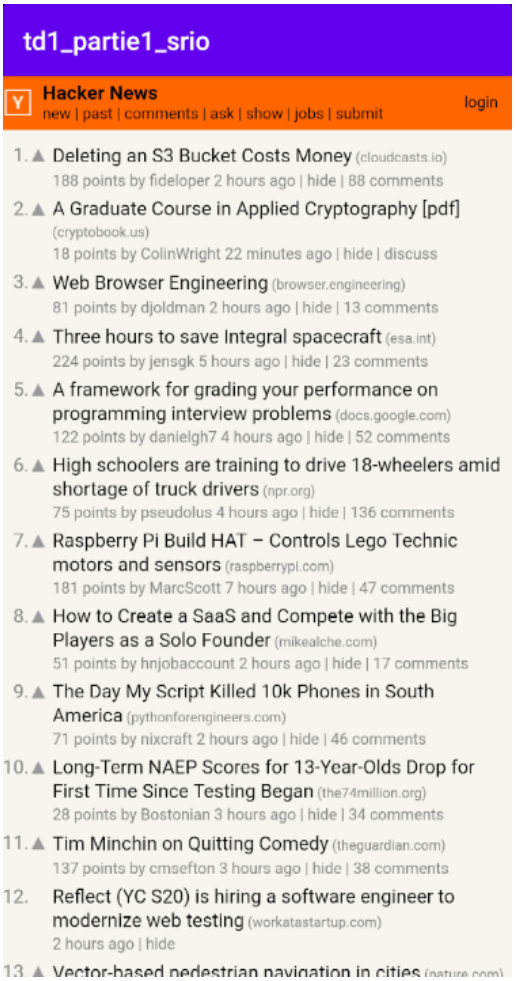
```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

Ce code crée une WebView qui s'adapte à la taille du parent, soit l'écran. Maintenant que la WebView est créée, on veut pouvoir afficher le site web de notre choix, dans notre cas https://news.ycombinator.com/. On ajoute le code suivant dans le fichier MainActivity.java et plus précisément dans la fonction onCreate(). Il s'agit de la fonction qui est lancée au lancement de l'application.

```
Context activityContext=this.getApplicationContext();
WebView myWebView = new WebView(activityContext);
setContentView(myWebView);
myWebView.loadUrl("https://news.ycombinator.com");
```

Dans la deuxième ligne du code, on instancie une WebView dans le contexte de l'activité qu'on récupère dans la ligne d'avant. La troisième ligne lie la WebView à l'activité. Finalement, la dernière ligne charge l'adresse du site internet sur la WebView.

Lorsqu'on lance l'application, on tombe bien sur la page web renseignée.




1. La technologie WebView est une sorte de cadre dans lequel on peut afficher du code web. La WebView est différente d'un navigateur car on est en quelque sorte coincé sur le site, mais on peut toujours se déplacer dans celui-ci.

2. Utiliser une WebView peut être intéressant pour une entreprise de petite taille si celle-ci ne veut pas investir dans le développement d'une application. Elle n'aura qu'à créer une application avec une WebView qui redirige vers son site et qui sera disponible sur Google Store.

## Partie II : Exécuter du code local au sein d'une instance WebView

Après avoir créé le répertoire et les fichiers index.html et main.js dans celui-ci, on écrit dans ceux-ci le code suivant :

```html
<!DOCTYPE html>
<html>
<body>

<script src="./main.js"></script>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

```
document.body.innerHTML = "<h1>bonjour</h1>";
```

On écrit dans le code html une balise script dans laquelle on indique que celle-ci doit exécuter le fichier main.js.

Quand on ouvre le .html dans un navigateur on tombe bien sur le résultat voulu.

**bonjour**

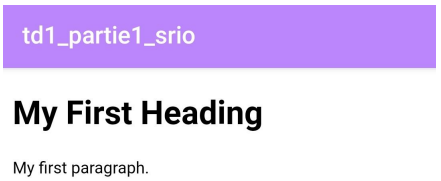
**My First Heading**

My first paragraph.

On cherche maintenant à charger le fichier html sur l'application. Pour cela, on transforme le fichier en string grâce au code ci-dessous. Celui-ci va chercher le fichier pour le lire, récupère chacune de ses lignes qu'il concatène dans un StringBuilder qu'il transforme finalement en string. On finit par charger ce string grâce à la fonction loadData de la WebView.

```
StringBuilder contentBuilder = new StringBuilder();
try {
    BufferedReader in = new BufferedReader(new InputStreamReader(getAssets().open( fileName: "www/index.html")));
    String str;
    while ((str = in.readLine()) != null) {
        contentBuilder.append(str);
    }
    in.close();
} catch (IOException e) {
}
String data = contentBuilder.toString();
myWebView.loadData(data,  mimeType: "text/html",  encoding: "utf-8"); //.loadUrl("https://news.ycombinator.com"); partie 1
```

Quand on lance ce code, on obtient le résultat du fichier html mais il manque le script JavaScript. En effet, quand on lit le code html pour le transformer en string, le code JavaScript de MainActivity ne sait pas qu'il est censé utiliser du script JavaScript. Pour pouvoir afficher le résultat du script, il faut ajouter les autorisations suivantes :

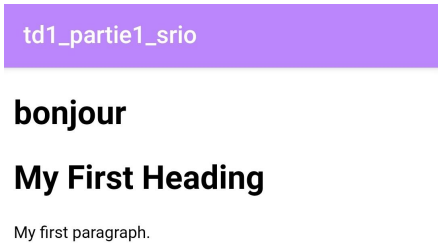



```
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

On a rencontré des problèmes à ce moment car le code JavaScript ne s'affichait pas. Après des problèmes rencontrés avec l'utilisation de loadData, on a remplacé cette méthode par le loadUrl de la première partie. On ne pensait pas que c'était possible de charger un fichier html à partir de loadUrl, d'où les complications rencontrées. On a maintenant :

```
myWebView.loadUrl("file:///android_asset/www/index.html");
```

Maintenant, le code JavaScript est bien affiché sur l'application :




1. Pour pouvoir exécuter du code JavaScript dans le contexte d'une WebView, il n'est pas nécessaire que l'application ait la permission d'accéder à Internet.

2. L'étape de sécurité qui nous permet d'activer ou de désactiver l'exécution du code JavaScript est importante car si celle-ci est activée, alors cela ouvre la porte à de nombreux problèmes.

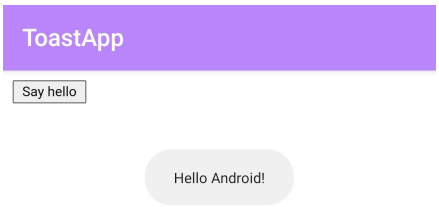
3. En effet, si le code HTML dans notre WebView ne vient pas totalement de nous, alors celle-ci peut être piégée et permettre à la personne derrière tout cela d'exécuter ou modifier le code.

## Partie III : Exécuter du code Java à partir de JavaScript

Dans cette partie, on va voir le mécanisme d'interface Java-JavaScript pour exécuter du code Java à partir du code JavaScript. On va utiliser ce mécanisme dans deux applications : ToastApp et ensuite PhoneApp qu'on a réalisé dans le deuxième TP.

ToastApp :

Dans cette première application, notre premier but est d'écrire un bridge qui va déclencher un toast Java depuis une JavaScriptInterface lorsqu'on appuie sur un bouton. Pour cela, on commence par ajouter une JavaScriptInterface à la WebView. Ensuite, la WebView utilise un code html dans laquelle on lui indique qu'on veut créer un bouton qui déclenche un code JavaScript qui lui à son tour déclenche un code Java annoté d'un @JavascriptInterface indiquant que ce code peut être utilisé par du code JavaScript. Ce code Java fait apparaître le toast. Quand on lance le code, on obtient bien la première fonctionnalité souhaitée :




Ensuite on veut écrire un bridge qui affiche l'identifiant unique de l'appareil dans la WebView. On n'a pas réussi la deuxième partie qui consiste à afficher l'identifiant unique du téléphone car malgré avoir ajouté la permission suivante,

```
<uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"
```

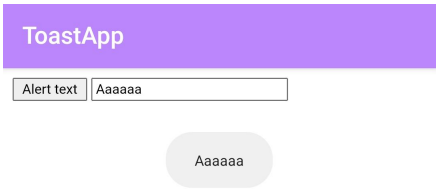
lorsqu'on lance l'application on obtient le message d'erreur suivant qu'on n'arrive pas à résoudre :

```
The user 10356 does not meet the requirements to access device identifiers.
```

Pour la dernière partie de cette application, il fallait que l'alerte affiche un texte qu'on a renseigné dans un champ <input>. Pour cela, j'ai créé ce nouveau champ de type text et j'ai juste modifié le code de la première partie de l'application en remplaçant le String contenant l'alerte par défaut par la valeur de mon input de type text :

```
<input type="button" value="Alert text" onClick="showAndroidToast(document.getElementById('uname').value)" />
<input type="text" id="uname" name="name"/>
```

On obtient alors :



# TP2 : Comprendre les enjeux de sécurité sur Android

Ce deuxième TP a pour but de nous familiariser avec certaines notions de sécurité informatique et plus particulièrement avec les vulnérabilités de WebView. Le but est de concevoir une application Android qui collecte des données sur un téléphone et les envoie sur un serveur distant. Ici, on va créer un carnet d'adresse qui enverra les contacts de l'utilisateur.

## Partie I : Implémenter un carnet d'adresses

Ce deuxième TP a pour but de nous familiariser avec certaines notions de sécurité informatique et plus particulièrement avec les vulnérabilités de WebView.

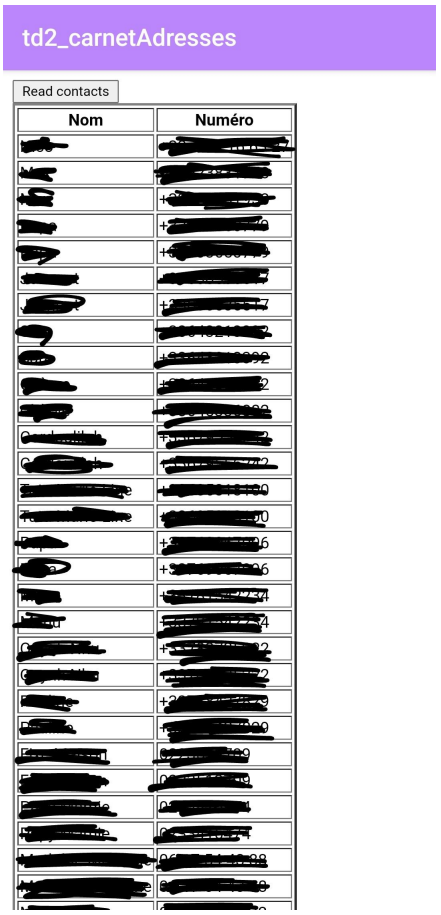
On commence par récupérer les contacts du téléphone grâce à la fonction Java readContacts(). Celle-ci rend un un tableau sous format Json qui est composé de plusieurs tableaux qui correspondent à chaque contact de l'utilisateur, plus précisément son nom et son numéro de téléphone. Cette fonction est appelée par la fonction JavaScript readTheContacts() lorsque le bouton "Read contacts" est activé.
La fonction readTheContacts() récupère alors le String Json de readContacts() et le parse. Pour pouvoir les afficher, on crée un tableau HTML dans lequel on met tous les contacts de ce tableau.

On veut maintenant  que l'application puisse envoyer un SMS à un utilisateur défini. Pour faire cela, on commence par créer une TextArea. Ensuite, on ajoute du code dans readTheContacts pour que lorsqu'on clique sur un contact, le numéro soit ajouté dans la TextArea. On ajoute un bouton "Send SMS" qui, lorsqu'on clique dessus, récupère le contenu de la TextArea, transforme les 10 premiers caractères en numéro et le reste en texte à envoyer. La fonction appelle ensuite la fonction sendSMS() de la JavaScriptInterface qui envoie le message au numéro et notifie si le message a bien été envoyé grâce à BroadcastReceiver qui affiche un Toast si un broadcast a bien été reçu.

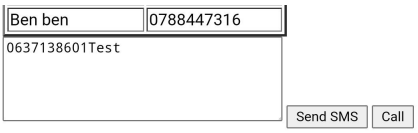
Pour finir, on veut écrire un bridge qui permet de composer le numéro de téléphone d'un contact sélectionné (PhoneApp du TP1). Pour cela on ajoute un troisième bouton "Call" qui appelle la fonction JavaScript call() qui récupère le numéro de téléphone de la TextArea comme le fait la fonction sendSMS() et qui appelle la fonction de la JavaScriptInterface call() qui va ouvrir l'application Téléphone avec le numéro renseigné.

Au démarrage de l'application, on arrive sur cette interface :

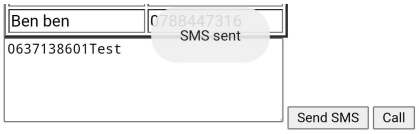

Lorsqu'on clique sur le bouton "Read contacts", on obtient un tableau de contacts avec sur la colonne de gauche leur nom et sur la colonne de droite leur numéro.

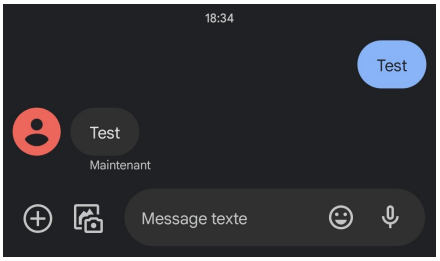


Pour envoyer un SMS à un numéro, il n'y a qu'à cliquer sur ce numéro et il apparaîtra dans la TextArea. On peut alors renseigner le texte souhaité juste après.

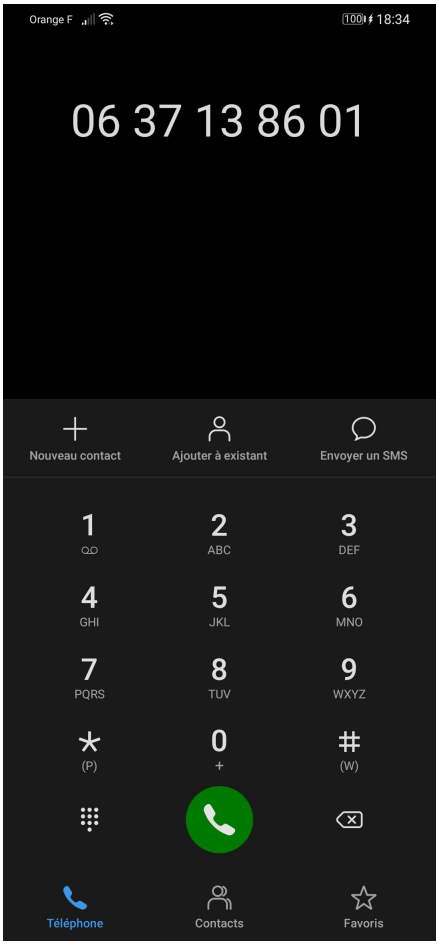



Pour envoyer le SMS, il n'y a qu'à cliquer sur le bouton "Send SMS". On reçoit alors une notification que le SMS a bien été envoyé si c'est le cas.




Si on veut appeler ce numéro, il n'y a qu'à cliquer sur le bouton "Call" et l'application Téléphone sera alors lancée avec le numéro de téléphone souhaité.



1. Il est nécessaire de sérialiser les données de contacts entre le contexte Java et le contexte WebView car les données de base ne sont pas compatibles entre les deux contextes. La sérialisation, ici en Json, est une sorte de pont entre les deux contextes qui leur permet de partager des données.

2. Les tableaux ou encore les chaînes de caractères sont utilisés pour passer des données de Java à WebView

3. Pour que l'application soit complètement fonctionnelle, il faut ajouter plusieurs permissions : INTERNET pour pouvoir utiliser la WebView, READ_CONTACTS pour pouvoir récupérer les contacts, SEND_SMS pour pouvoir envoyer des SMS et CALL_PHONE pour pouvoir utiliser l'application Téléphone.

## Partie II : Voler les contacts de l'utilisateur

On va maintenant modifier le code pour que l'application puisse envoyer les contacts de l'utilisateur à un serveur pirate distant.

### Étape 1 : Envoyer les contacts à un serveur distant

On commence par écrire le code Java qui aura pour but d'envoyer des requêtes HTTP au serveur. Il s'agit de deux fonctions getRequest() et postStringRequest(). Ensuite, on modifie la fonction JavaScript readTheContacts() de telle sorte qu'elle appelle postStringRequest() après qu'elle ait récupéré les contacts. De cette façon, elle peut maintenant envoyer le JSON au serveur.

1. On peut se prémunir d'une telle attaque en n'autorisant pas l'application à accéder à Internet.

2. Pour s'assurer qu'une application fait seulement ce qu'elle est censée offrir et pas plus, il faut faire attention aux permissions qu'on lui donne.

### Étape 2 : Développer un simple serveur web pirate

Maintenant que l'application envoie les informations dérobées, il faut désormais créer un serveur qui les récupérera et les stockera. Pour cela, on en conçoit un en Node.js. Il analyse les différents types de requêtes et s'il s'agit d'un POST, on parse les données et on les écrit dans un fichier. Au niveau du routage, on décide d'écouter les requêtes si elles finissent par /contacts.

Le serveur écoute une certaine adresse et un certain port qu'on est censé mettre en paramètre dans postStringRequest(). Le téléphone n'ayant pas accès à l'adresse localhost du PC, on utilise Ngrok pour créer une adresse publique qui sera disponible pour les deux. Avec la commande ngrok http 8080, on a obtenu l'adresse `312c-148-60-138-232.ngrok.io` et on peut maintenant envoyer les contacts du téléphone au serveur grâce à l'URL :
`https://312c-148-60-138-232.ngrok.io/contacts`.

Maintenant, en appuyant sur Read contacts, un fichier est écrit côté serveur avec comme contenu :


```
"\"[{\\\"name\\\":\\\"Nico\\\",\\\"phoneNumber\\\":\\\"+33 7 83 [redacted]\\\"},
```