
Compte Rendu phase 2

Projet Android

Emery Bourget-Veccchio & Romain Fournier

M1 IMAGINA

Université de Montpellier

30/04

2020-2021

git android: https://github.com/EmeryBV/LocaliserMonEnfant_enfant

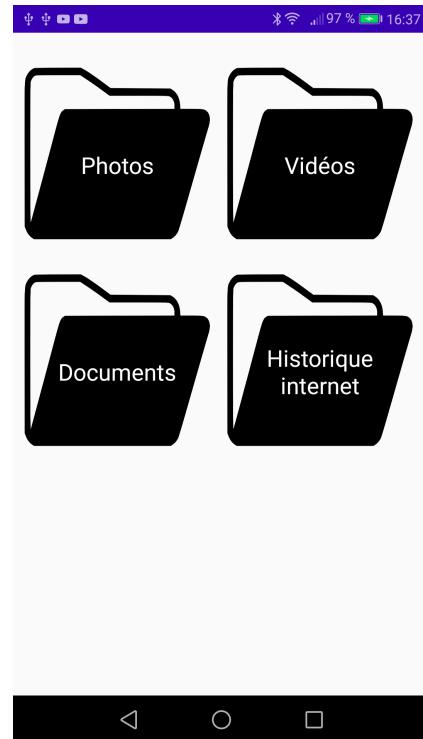
git serveur: https://gitlab.com/Romimap/lme_server

Lien vers le premier dépôt: [Compte Rendu projet android](#)

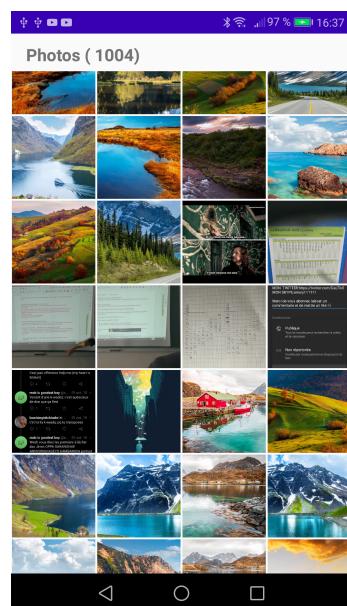
Partie Android

Pour cette seconde phase, nous avons développés la récupération des historiques d'appels, la récupération des photos et la récupération des vidéos du téléphone.

Maintenant, quand l'utilisateur va dans la partie "Média", il a le choix de pouvoir consulter les données du téléphone, en l'occurrence ici les photos et les vidéos.

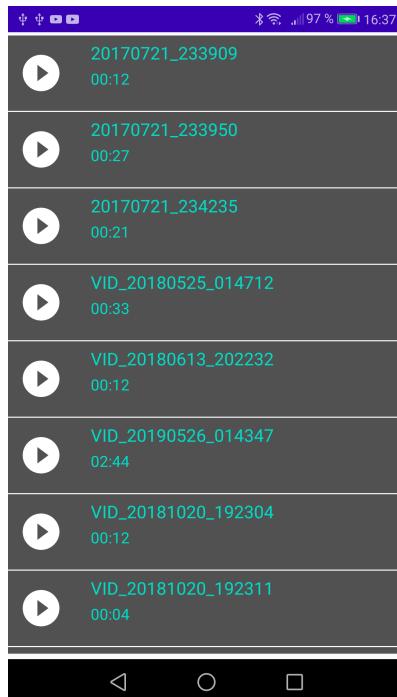


En cliquant sur "Photos", nous avons accès à toutes les photos contenues dans le téléphone.

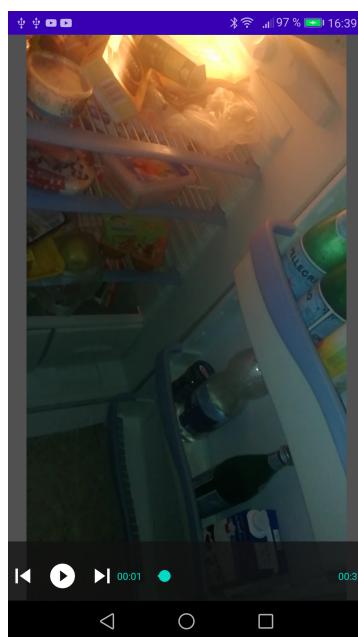


Cliquez sur une photo permet d'avoir la localisation de la photo dans le téléphone avec la date et le nom de la photo.

Si on clique sur le dossier vidéo, nous avons la liste de toutes les vidéos contenues dans le téléphone qui s'affiche.

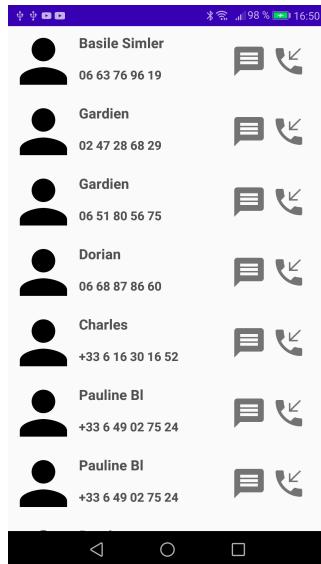


Si on clique sur une vidéo, cette dernière est lancé sur un lecteur intégré à l'application



On peut voir en bas que nous pouvons lire, lancer la vidéo précédente, lancer la vidéo suivante et grâce au slider de lecture avancer ou reculer dans la vidéo.

Si nous allons dans la partie “contact”, nous pouvons constater que la mise en forme à changer depuis la phase 1, en effet, nous avons rajouté 2 icônes qui permettent respectivement de soit de voir la conversation SMS avec le dit contacte, ou bien de voir l'historique des appels avec ce même contact.



En cliquant sur “l'historique des appels”, nous voyons la date, l'heure, le numéro de téléphone (au cas où le contact aurait plusieurs numéros) le type d'appel (entrant, sortant, raté) et enfin le temps de l'appel en seconde.

| Gardien | | |
|---|-----|---------|
| Fri Nov 20 11:55:22 GMT+01:00 2020 s | | Sortant |
| | 169 | |
| 0247286829 | | |
| Fri Nov 20 11:54:31 GMT+01:00 2020 s | | Sortant |
| | 0 | |
| 0247286829 | | |
| Fri Nov 20 11:40:12 GMT+01:00 2020 s | | Raté |
| | 0 | |
| 0247286829 | | |
| Fri Nov 20 11:39:40 GMT+01:00 2020 s | | Raté |
| | 0 | |
| 0247286829 | | |
| Thu Apr 23 17:28:17 GMT+02:00 2020 s | | Entrant |
| | 19 | |
| 0247286829 | | |
| Wed Apr 22 11:52:35 GMT+02:00 2020 s | | Entrant |

Enfin nous avons implémenté une page de connexion/inscription non fonctionnelle du fait que le serveur ne soit pas encore 100% opérationnel.

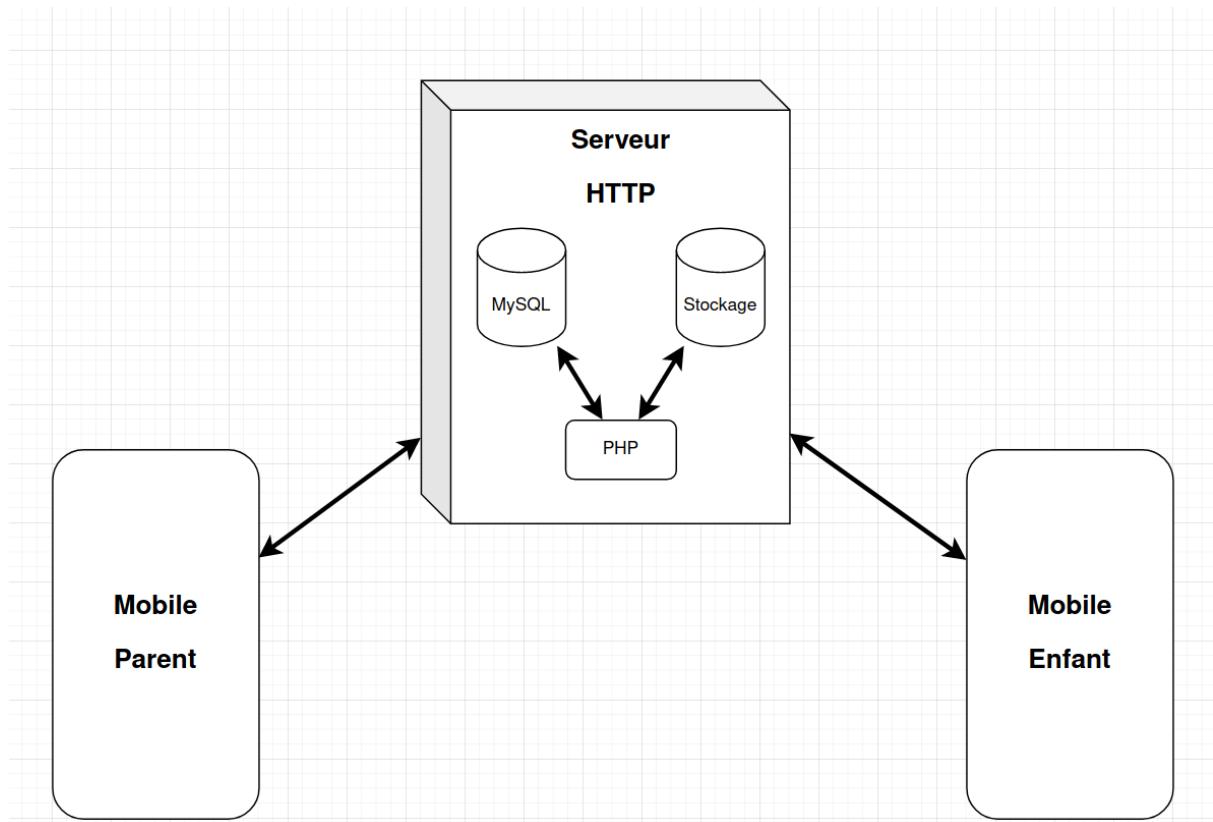
The image shows two screenshots of a mobile application interface. Both screenshots have a dark blue header bar at the top with icons for signal strength, battery level (99%), and time (16:57). The left screenshot shows a "Connexion" (Login) screen with fields for "Nom d'utilisateur" (Username) and "Mot de passe" (Password), a "SE CONNECTER" (Connect) button, and a "S'inscrire" (Sign Up) link. Below the form is a colorful logo of a stylized bird or plane. The right screenshot shows an "S'inscrire" (Sign Up) screen with fields for "Pseudo" (Nickname) and "Mot de passe" (Password), an "INSCRIPTION" (Sign Up) button, and a "Déjà inscrit ? Connectez-vous" (Already registered? Log in) link. Below the form is a black navigation bar with three white icons: a triangle pointing left, a circle, and a square.

Les fonctionnalités actuellement présentées sont incorporées de façon à ce que la connexion avec la BD soit le plus simple possible.

Serveur

Pour ce projet, nous avons besoin d'effectuer des requêtes à un serveur pour récupérer les données qui nous intéressent, stockées sur ce dernier et dans une base de données. L'enfant envoie ses données à un serveur, et le parent les récupère ensuite.

(Schéma du précédent rapport, le transfert de données entre un serveur et les applications)



À noter, encore beaucoup de code reste non testé, la liaison entre le serveur et l'application n'étant pas triviale dans le cadre d'un serveur local et d'un téléphone émulé. Il nous reste encore à héberger notre code sur un serveur distant pour vérifier que tout fonctionne en situation réelle.

Serveur HTTP : Les Requêtes

Nous sommes donc partis sur un serveur HTTP, où le parent, en envoyant une requête POST au serveur, pourrait demander certaines données. Par exemple :

| URL | contenu POST |
|--------------------------------------|---|
| <code>https://serveurdistant/</code> | <code>type="SMS"&idenfant="12"&idcontact="4"</code> |

Ici, on envoie une requête pour récupérer **les SMS** entre l'**enfant 12** et le **contact 4**.

| contenu POST |
|---|
| <code>type="Connection"&login="parent1234"&passwd="motDePasse"</code> |

Ici, on envoie une requête pour se **Connecter** avec le login **parent1234** et le mot de passe **MotDePasse**. Le serveur va alors considérer que pour cette session PHP, le parent "parent1234" est authentifié. Il se servira de cette information pour vérifier qu'il ne demande pas les données d'un autre enfant. Si jamais le parent n'est pas authentifié sur la session PHP courante, le serveur lui fera savoir et n'enverra aucune données.

Serveur HTTP : Les Réponses

Le serveur en retour, va répondre à la requête par une chaîne de caractère JSON. Nous avons choisi ce format car simple à traiter et il existe 1000 et une API nous permettant de le faire.

Pour une requête :

contenu POST

type="SMS"&idenfant="12"&idcontact="4"

Le serveur va :

- Vérifier que **l'enfant 12** à bien pour parent le parent authentifié
- Vérifier que le **contact 4** est bien dans la liste des contacts de **l'enfant 12**
- Récupérer les **SMS** entre **l'enfant 12** et le **contact 4**

Puis retourner une chaîne de caractère sous cette forme (sujet à changer) :

```
[  
  {  
    "Texte": "Texte du SMS 1"  
    "Type": "Sender (l'enfant l'as envoyé)"  
    "Date": "1 janvier 1970"  
  }, {  
    "Texte": "Texte du SMS 2"  
    "Type": "Receiver (l'enfant l'as reçu)"  
    "Date": "1 janvier 1970"  
  },  
  ...  
  {  
    "Texte": "Texte du SMS 156"  
    "Type": "Receiver (l'enfant l'as reçu)"  
    "Date": "1 janvier 1970"  
  }  
]
```

Ici, on à un tableau d'éléments représentant un SMS, chaque élément contient le Texte du SMS, le type (si l'enfant l'a envoyé ou non), et la date.

Pour l'instant, les données sont stockées en texte, mais il nous semble évident que ces derniers devraient être encryptés d'appareil à appareil :

- encryptés par l'enfant avant l'envoi au serveur à l'aide de la clé publique du parent et de sa clé privée;
- décryptés par le parent une fois les données récupérées à l'aide de sa clé privée et de la clé publique de l'enfant

Implémentation de Volley

le squelette du serveur étant fonctionnel, nous avons réalisé la partie client sur l'application. Pour se faire, nous avions besoin de :

- Réaliser des requêtes POST à un serveur
- Parser les données JSON dans un objet
- Gérer la composante asynchrone des requêtes

Après (de longues) recherches, nous avons décidé d'utiliser la librairie Volley, qui est une librairie de haut niveau permettant de réaliser des requêtes POST, qui s'occupe de parser les données, et de gérer une pile de requêtes pour nous.

Nous avons implémenté une classe autour de cette librairie, permettant de s'authentifier et de récupérer des données auprès du serveur.

Volley fonctionne à base de Callbacks, quand on fait une requête, cette dernière est ajoutée à une pile de requêtes et traitée quand le temps le permets. Une méthode est appelée quand une réponse du serveur est reçue.

Le but de notre classe est de pouvoir appeler simplement des méthodes du type GetPictures(...), GetSMS(...) etc... Nous avons une méthode Post() qui propage le callback de Volley à une de ces méthodes, qui sera ensuite propagé vers l'extérieur de la classe. Pour comprendre un peu plus en détail comment le code fonctionne, il est disponible ci-dessous.

Dans notre méthode Post, nous exécutons une requête. Un callback est propagé à l'aide d'une interface "ConnectionCallBack". Cette méthode est privée à notre classe, elle n'est jamais appelée depuis les activités de l'application.

Prends en paramètre un contexte, une URL, des paramètres à envoyer en POST et une implémentation de l'interface ConnectionCallBack

```
void Post (Context context, String strUrl, Map<String, String> params,  
          final ConnectionCallBack connectionCallBack) {
```

On crée une file de requêtes, ici on en crée une par requête, mais ça pourrait être optimisé si on en créait une pour toutes.

```
RequestQueue queue = Volley.newRequestQueue(context);
```

On prépare les paramètres à passer par POST

```
JSONObject p = new JSONObject();  
try {  
    for (Map.Entry<String, String> pair : params.entrySet()) {  
        p.put(pair.getKey(), pair.getValue());  
    }  
} catch (Exception e) {  
    Log.d("tag", "ERROR");  
}
```

On prépare la requête. C'est ici qu'on à un Callback.

```
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(strUrl, p,  
    new Response.Listener<JSONObject>() {
```

Sur une réponse, on appelle la méthode onSuccess de l'instance de ConnectionCallBack passée en paramètre.

```
@Override  
public void onResponse(JSONObject response) {  
    connectionCallBack.onSuccess(response);  
}  
, new Response.ErrorListener() {
```

Pareil sur une erreur.

```
@Override  
public void onErrorResponse(VolleyError error) {  
    connectionCallBack.onError(error);  
}  
});
```

On ajoute la requête à la file

```
queue.add(jsonObjectRequest);
```

```
}
```

Ainsi, grâce à cette implémentation, nous pouvons propager les Callbacks jusqu'à nos méthodes de haut niveau (GetPictures(), GetSMS(), etc...) utilisables de cette manière dans nos activités :

```
Quelque part dans l'application,  
où on a besoin de photos...  
connection.GetPictures(context, idEnfant, idContact, new ConnectionCallback {  
    Ici, on implémente l'interface ConnectionCallBack  
    Ce code sera exécuté quand et si le serveur nous retourne une réponse  
    @Override  
    void OnSuccess(JSONElement response) {  
        Les données sont déjà parsées et simple à traiter  
        String link = response.get("Link");  
    }  
  
    On a aussi la possibilité de gérer les erreurs  
    @Override  
    void OnError(Error error) {  
    }  
});
```