
Compte Rendu Android

Application Mobile pour Sécurisation et Surveillance des Enfants par les Parents

Cette application est définie pour des raisons pédagogiques.

Emery Bourget-Veccchio & Romain Fournier

M1 IMAGINA

Université de Montpellier

30/05

2020-2021

git Android: [git Android](#)

git serveur: [git serveur](#)

Lien vers la vidéo de démonstration: [Vidéo youtube](#)

Lien vers le premier dépôt: [Compte Rendu projet android phase 1](#)

Lien vers le deuxième dépôt: [Compte Rendu projet android phase 2](#)

Lien vers les diapositives de présentations de l'application: [Diapositives Android](#)

Logs pour accéder à la base de donnée

lien : <https://phpmyadmin.cluster030.hosting.ovh.net/>
server : atkzjkqlme.mysql.db
username : atkzjkqlme
pass : yL6VqUu9ejd5e8p

PREAMBULE	3
PARTIE ANDROID	4
Architecture de l'application	4
Principales fonctionnalités de l'application	10
PARTIE SERVEUR	20
Architecture & Technologies	20
Protocoles	21
Envoi de données	21
Réception de données	22
Format des requêtes	22
Cas particulier de la connection	24
Notes de sécurité	25
Volley plus	26
API serveur	28
Serveur	30
Base de données	32
Conclusion	33

PREAMBULE

Dans le cadre de l'UE développement Android , nous avons eu l'occasion de réaliser une application Android destinée à des parents afin de surveiller leurs enfants pour leur sécurité. Le but de l'application est de pouvoir localiser son enfant, accéder au contenu de son mobile, être alerté par rapport à sa présence dans certaines zones géographiques etc. Nous avons ainsi mis en place deux interfaces différentes :

-La première est l'interface disponible sur le téléphone de l'enfant qui va permettre de récupérer les données de son téléphone et de les envoyer à un serveur qui sera chargé de stocker les données.

-La deuxième interface est celle disponible sur le téléphone du parent qui va permettre d'afficher clairement toutes les données relatives à l'enfant choisi.

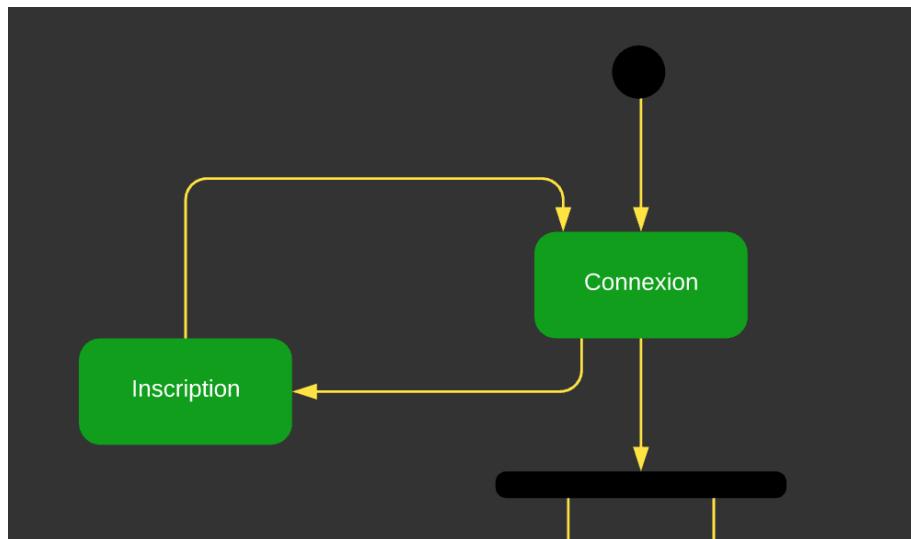
Nous allons dans ce compte rendu traiter dans un premier de la structure de l'application android en décrivant toutes les fonctionnalités disponibles, puis dans un second temps, nous nous intéresserons à la façon dont les données sont stockées dans le serveur.

I. PARTIE ANDROID

A. Architecture de l'application

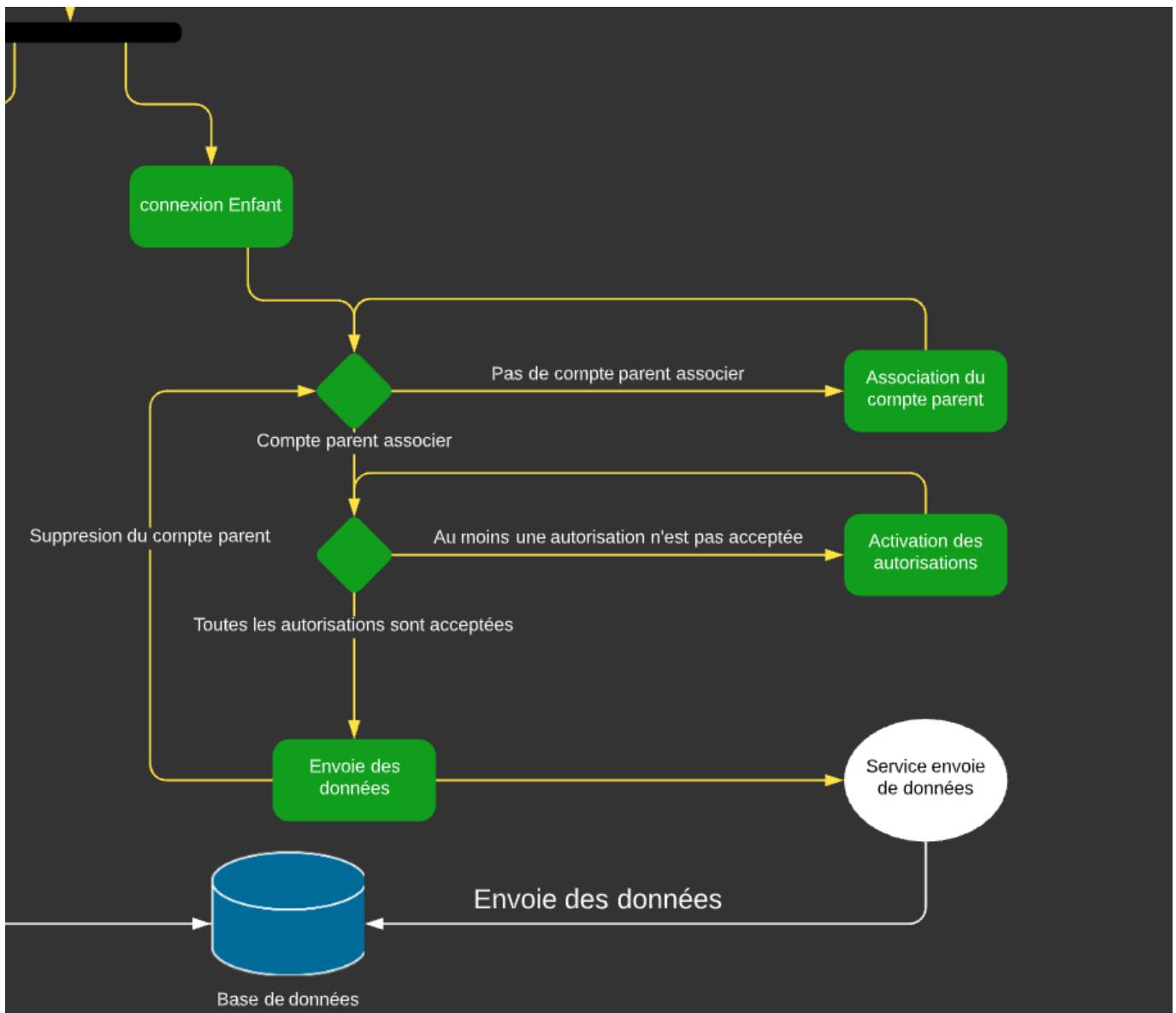
Pour la structure de l'application, nous avons imaginé réaliser la même application du côté enfant que parent. Le parent devra ainsi installer l'application sur le téléphone de son enfant et le sien puis relier les deux comptes. Du point de vue de l'enfant, l'application sera presque totalement vide, alors que du point de vue du parent, il aura accès à presque toutes les données de son enfant.

L'utilisateur devra se connecter avec son compte sur une page de connexion, s'il n'a pas encore de compte, il devra en créer un en stipulant le type d'utilisateur qu'il est (parent ou enfant). Une fois l'inscription terminée, il sera renvoyé sur la page de connexion.



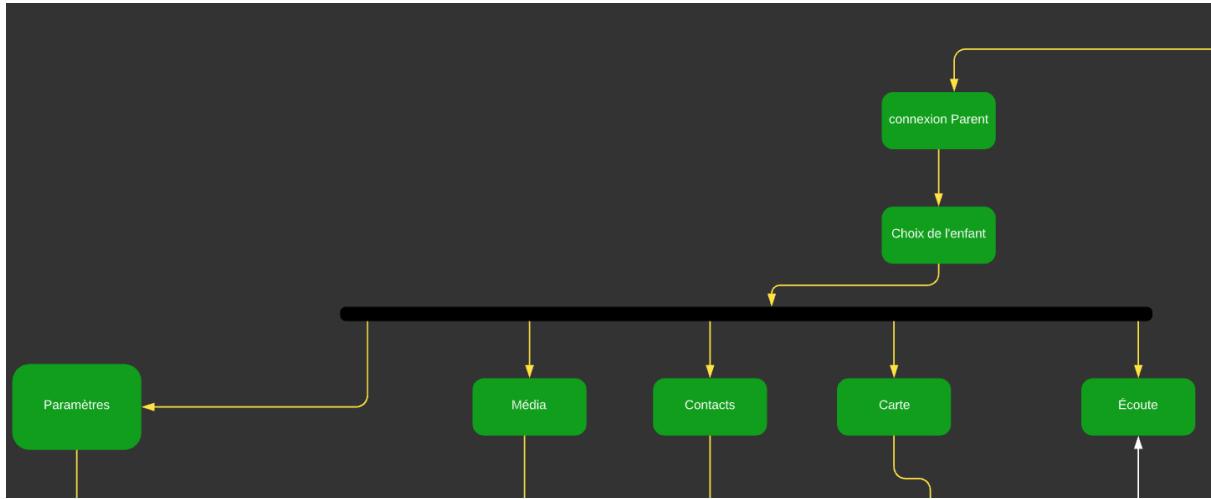
1. Application du point de vue de l'enfant

Si ce n'est pas déjà le cas, l'enfant devra relier son compte avec un parent. Pour ce faire, il devrait se connecter avec un compte parent. Une fois le compte lié, il devra accepter toutes les autorisations en rapport à l'accès aux données de contact, de messageries, de média, d'historique d'appels et de géolocalisation. Il arrivera ensuite sur page fixe qui lancera en parallèle un service permettant la récupération de toutes les données et leurs envois sur la base de données. L'enfant pourra supprimer un parent et reviendra alors sur la page permettant d'associer un nouveau parent.



2. Application du point de vue du parent

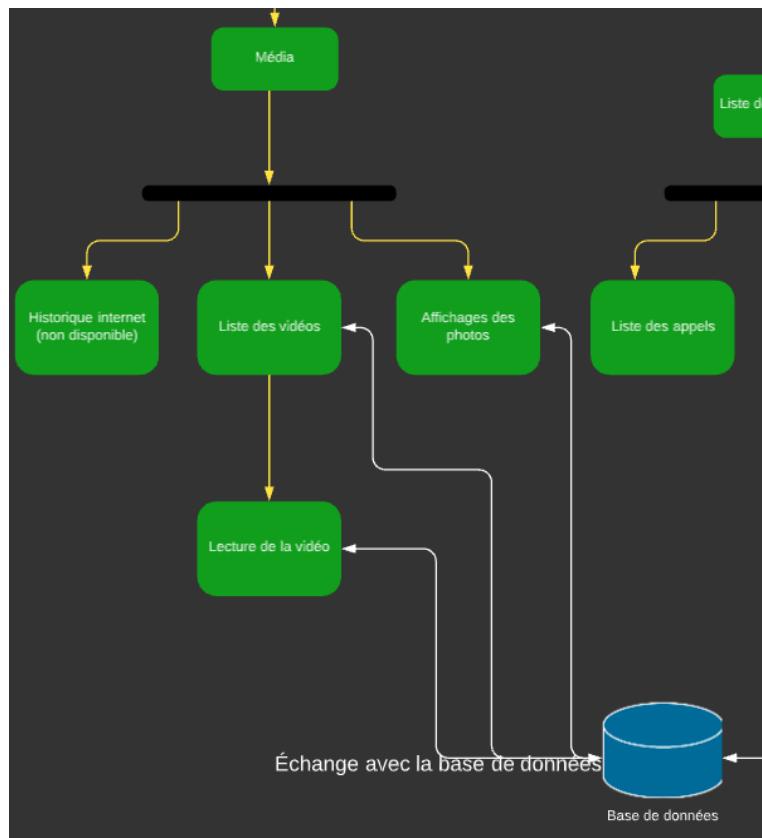
Si le parent n'a pas de compte enfant associé, il devra s'identifier sur l'application de son enfant. Il aura la possibilité de choisir parmi l'un d'entre eux afin de regarder les données de son téléphone. Une fois l'enfant choisi, il pourra consulter les médias, les contacts (Sms et historique d'appels), une carte contenant la position de l'enfant et enfin écouter le micro du téléphone enfant. Il pourra également changer le type d'alerte qu'il reçoit dans la section paramètres. Toutes les données seront chargées directement depuis la base de données, là où le téléphone de l'enfant les aura stockées



i. Média

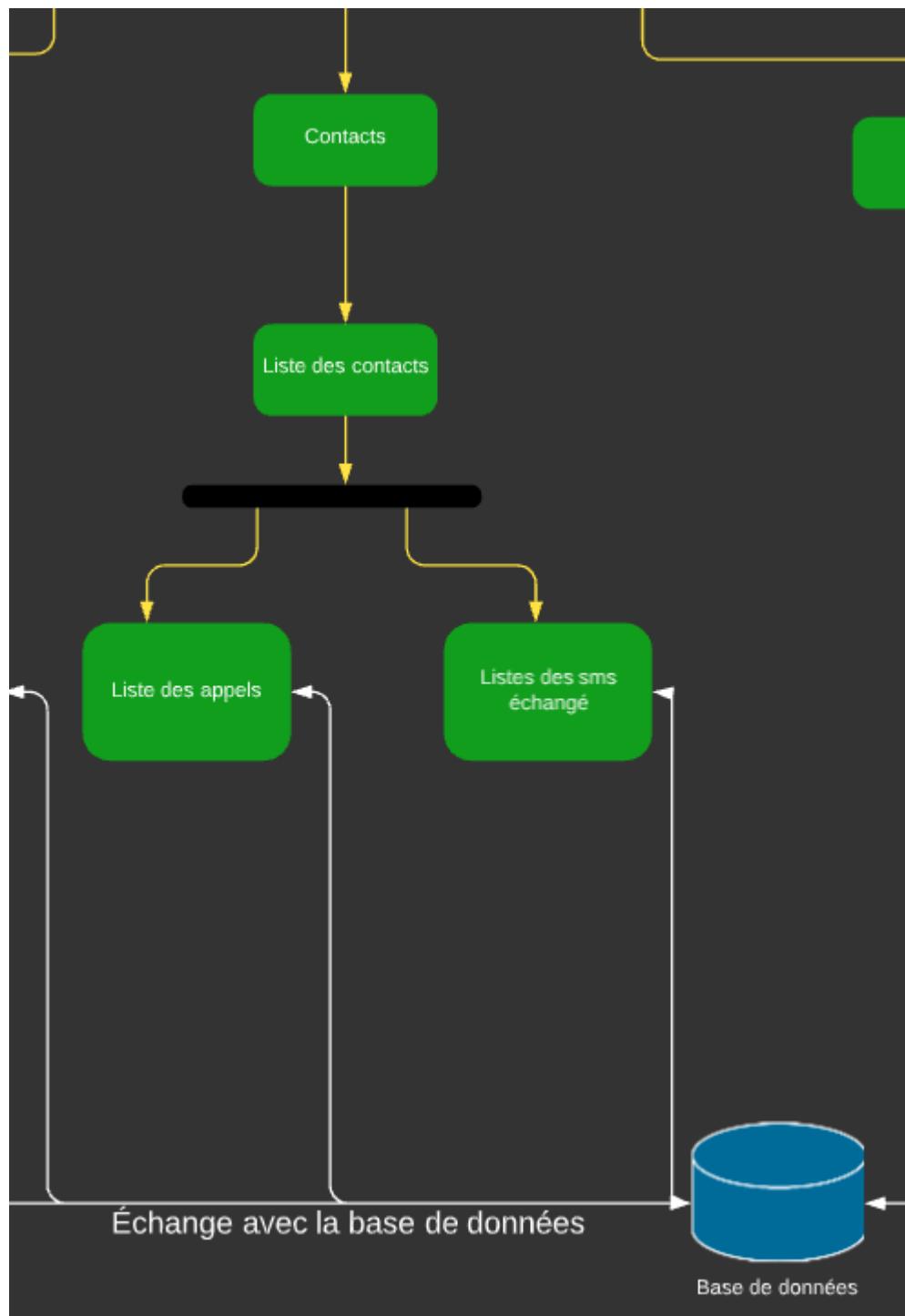
En accédant au **média**, le parent aura le choix entre regarder les photos et les vidéos.

Le sujet nous demande de pouvoir consulter également l'historique internet mais cette fonctionnalité n'est plus disponible depuis la version 23 d'Android. (Pour avoir plus d'informations, merci de cliquer [ici](#) et [ici](#)). Si l'utilisateur clique sur une vidéo, la vidéo se lira dans un lecteur intégré à l'application, et s'il clique sur une photo, il aura le lien de la photo sur le serveur.



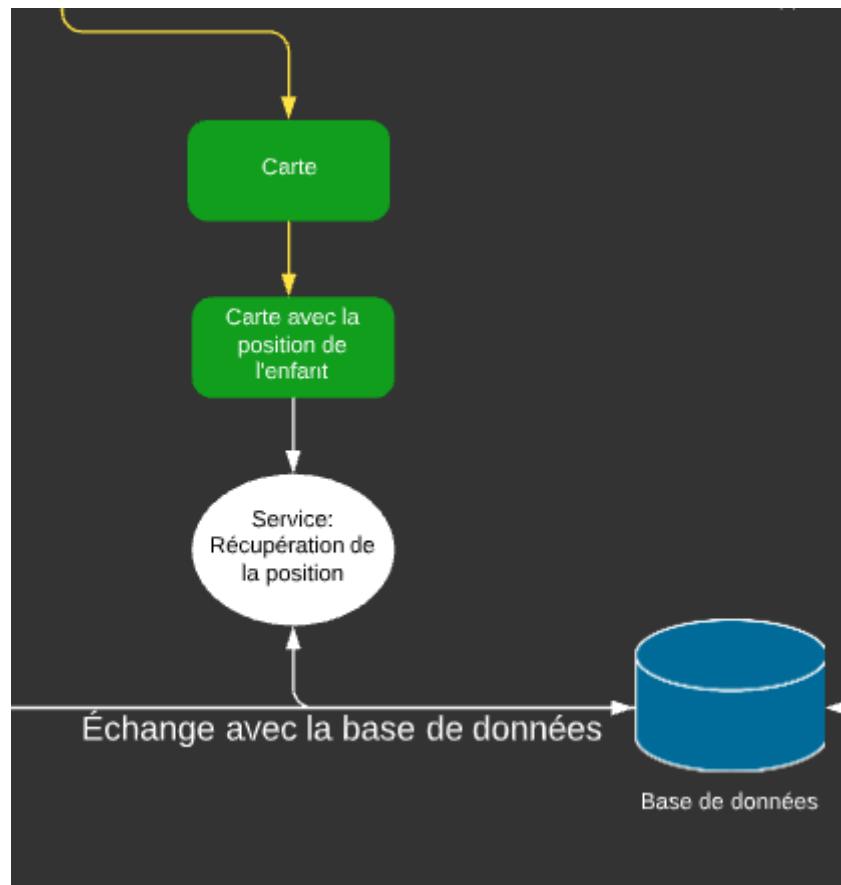
ii. Contacts

Depuis l'option **contact**, le parent aura une vue sur la liste de tous les contacts de l'enfant, pour chaque contact, il pourra avoir accès à tous les échanges (SMS et Appels) qu'a pu avoir l'enfant avec ce contact



iii. Carte

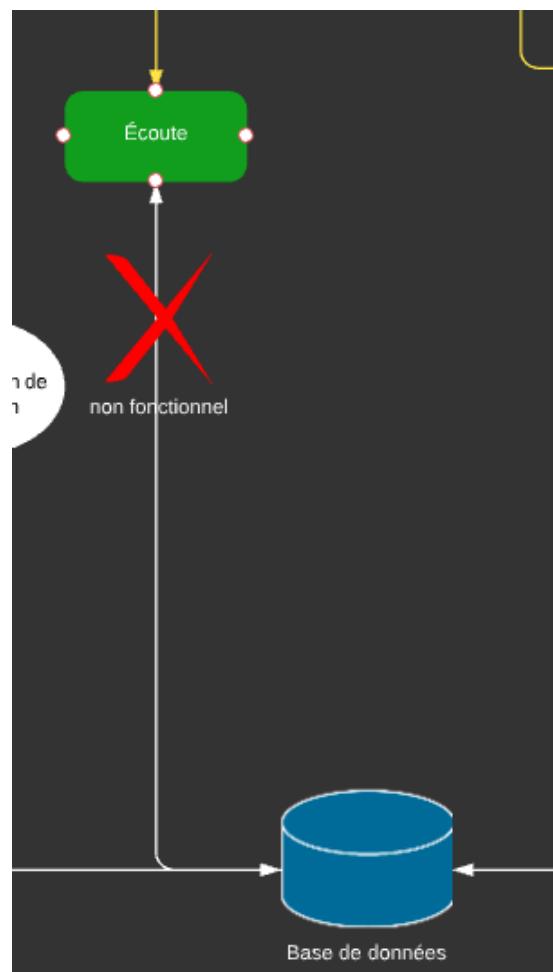
Avec l'option **carte**, le parent pourra voir en quasi temps réel les déplacements de son enfant grâce à l'api Google Map pour Android et à un service qui récupère toutes les 10 secondes la position de l'enfant dans la base de données. Il pourra également rajouter des zones pour vérifier la présence ou non de l'enfant dans ces dernières à certaines heures.



iv. Écoute

La dernière fonctionnalité est **écoute**. Cette fonctionnalité permet au parent d'envoyer un message au téléphone de l'enfant afin de lui demander de commencer à enregistrer le micro. Pour y mettre fin, le parent doit appuyer sur le bouton stop et il pourra alors entendre un enregistrement en différé de ce qui vient de se passer sur le téléphone.

Cette fonctionnalité n'a malheureusement pas pu être terminée et ne marche que localement, c'est-à-dire que le parent peut enregistrer le micro que sur son téléphone.

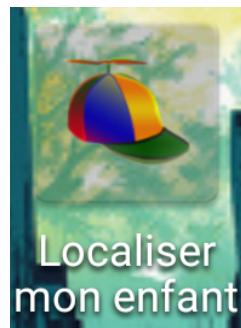


B. Principales fonctionnalités de l'application

Nous allons dans cette partie montrer directement dans l'application les points principaux qu'on a pu aborder dans la partie précédente. On rappelle que pour voir une démonstration du fonctionnement de l'application, une vidéo démonstrative est disponible sur youtube en cliquant [ici](#).

1. Le lancement et l'enregistrement

Pour se connecter, l'utilisateur aura à cliquer sur l'application avec comme icône une casquette à hélice qui représente bien la jeunesse.



Une fois dans l'application, l'utilisateur devra s'enregistrer si ce n'est pas déjà le cas et se connecter.

The image shows two screenshots of the application's interface side-by-side. Both screenshots have a dark grey header bar at the top showing signal strength, battery level (37%), and time (17:54 for the left, 01:00 for the right). The main title "Localiser mon enfant" is centered at the top of both screens.

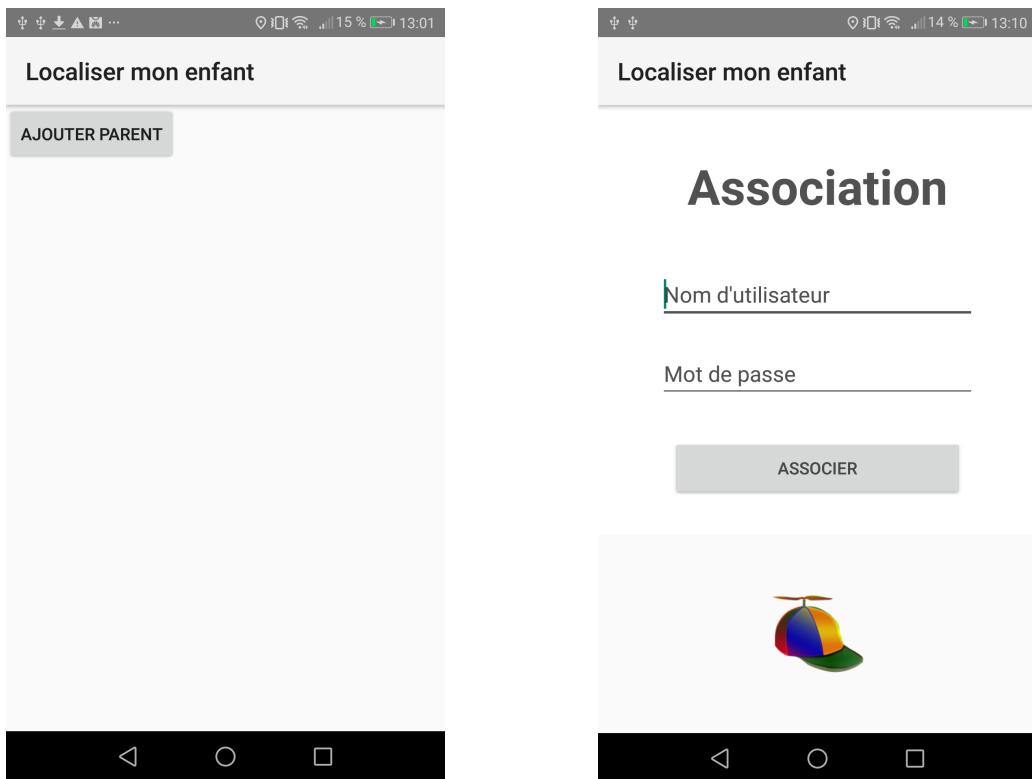
Connexion (Left Screen):

- A text input field labeled "Nom d'utilisateur".
- A text input field labeled "Mot de passe".
- A grey button labeled "SE CONNECTER".
- A small "S'inscrire" link below the button.
- An icon of a colorful propeller hat in the center.
- At the bottom, there is a black navigation bar with three white icons: a triangle pointing left, a circle, and a square.

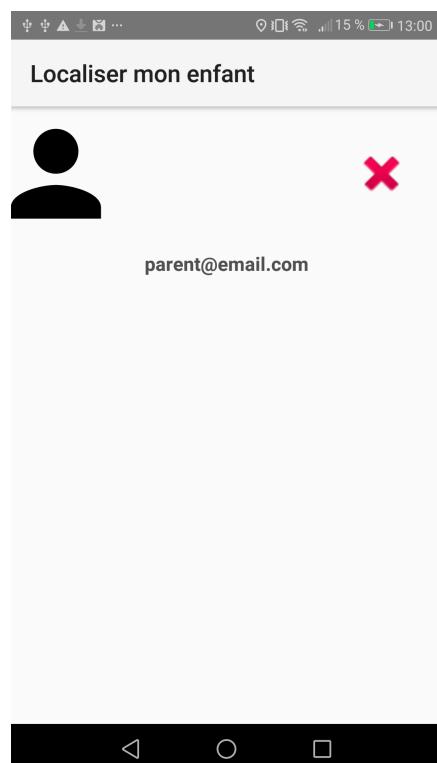
S'inscrire (Right Screen):

- A text input field labeled "Pseudo".
- A text input field labeled "Mot de passe".
- Two radio buttons: one selected (filled with a green dot) labeled "Parent" and one unselected (white outline) labeled "Enfant".
- A grey button labeled "INSCRIPTION".
- A link "Déjà inscrit ? Connectez-vous" below the button.
- An icon of a colorful propeller hat in the center.
- At the bottom, there is a black navigation bar with three white icons: a triangle pointing left, a circle, and a square.

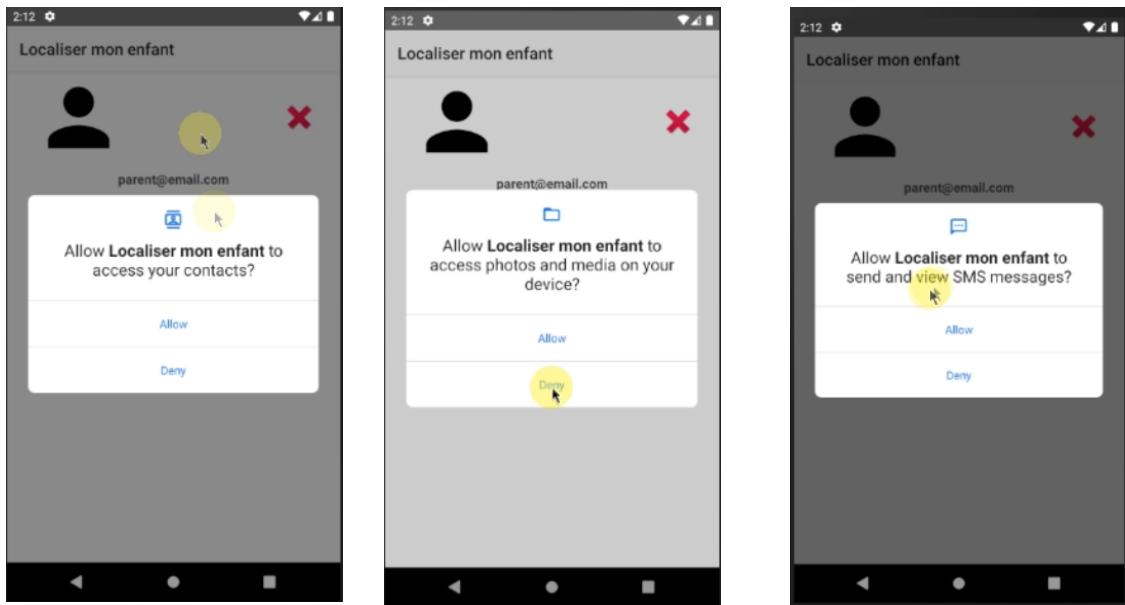
Lorsqu'un enfant se connecte, il devra associer un parent si ce n'est pas déjà le cas. Pour cela, il devra se connecter avec le compte parent.



C'est alors que le parent apparaîtra et c'est à partir de ce moment que l'application demandera à l'utilisateur d'accepter toutes les autorisations telles que la récupération des contacts, des médias, des SMS etc...



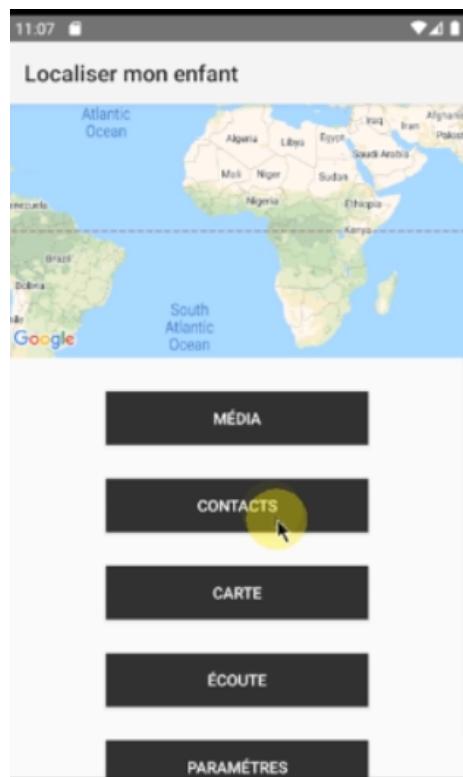
Voici quelques exemples d'autorisations à accepter.



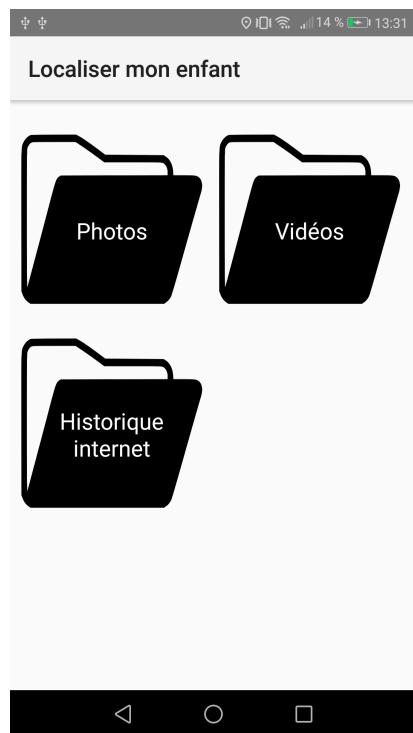
On s'intéresse maintenant à la partie parent. Une fois connecté, le parent devra choisir lequel de ses enfants il veut surveiller.



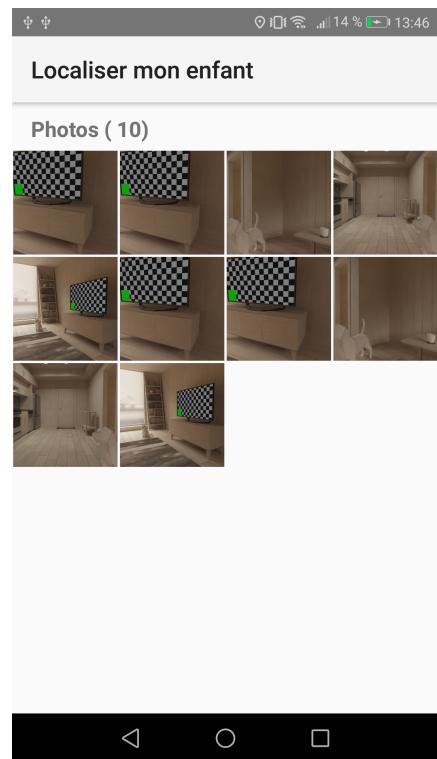
Une fois l'enfant choisi, on arrive dans le menu principal.



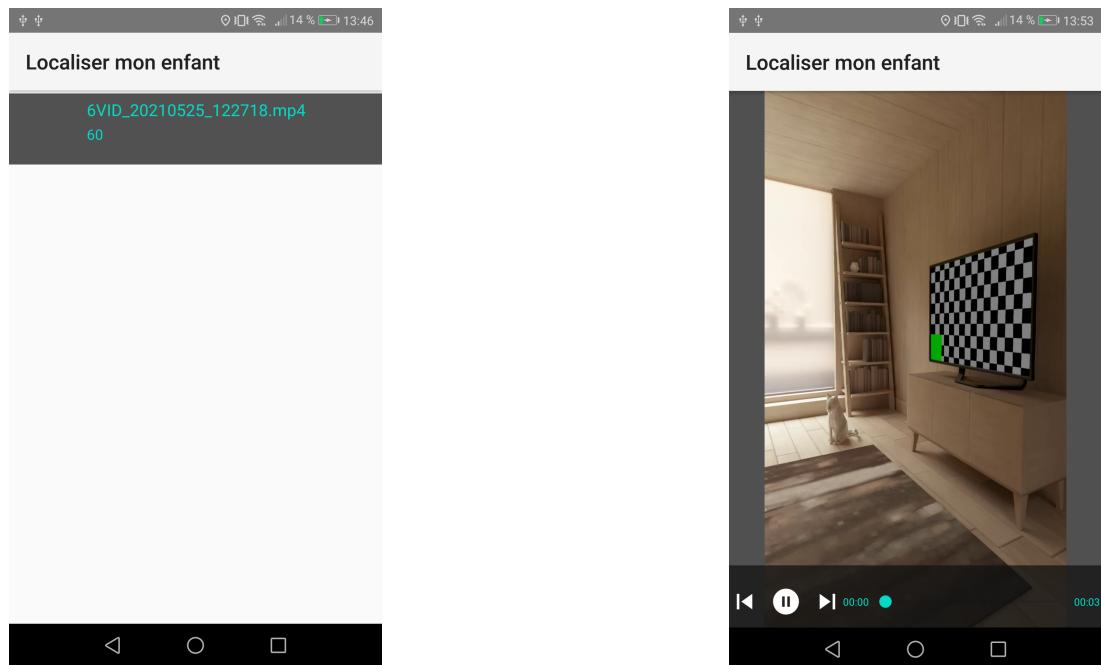
En cliquant sur média, le parent arrive sur cette page.



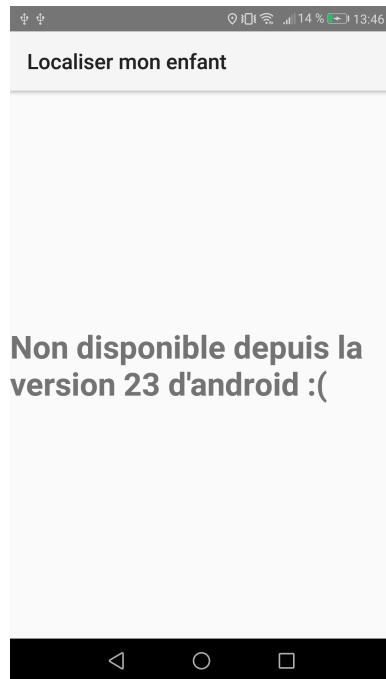
Il pourra ainsi consulter les photos de l'enfant



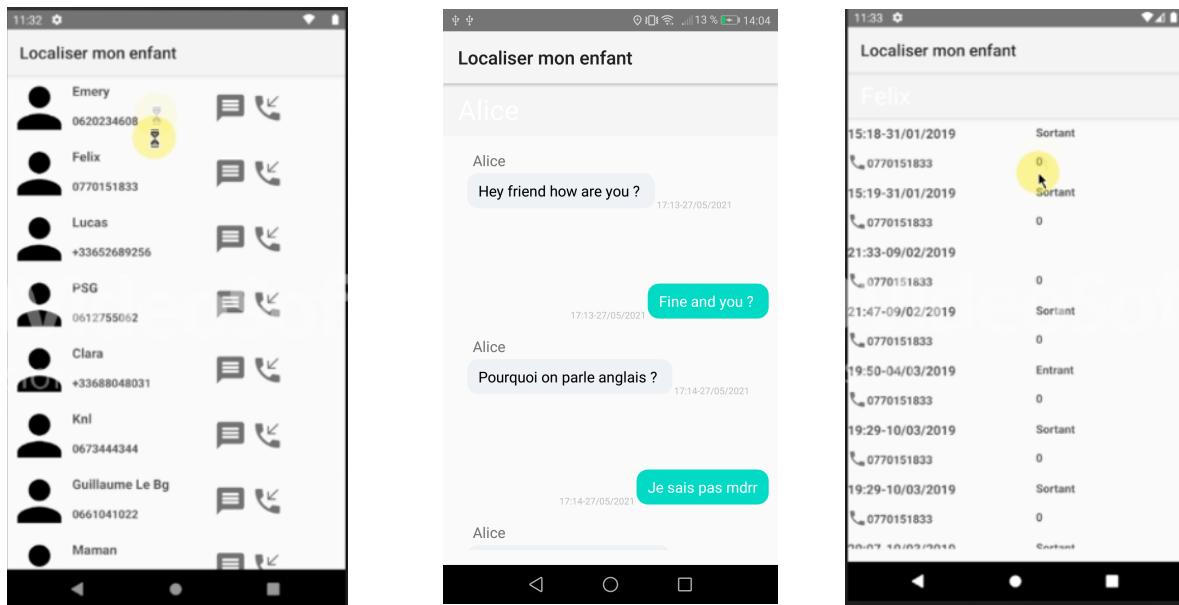
et avoir la liste des vidéos disponible sur l'appareil de l'enfant. En cliquant sur une vidéo, la vidéo se lira dans un lecteur vidéo intégré à l'application.



Comme indiqué dans la première partie l'historique internet n'est pas disponible depuis la version 23 d'Android.

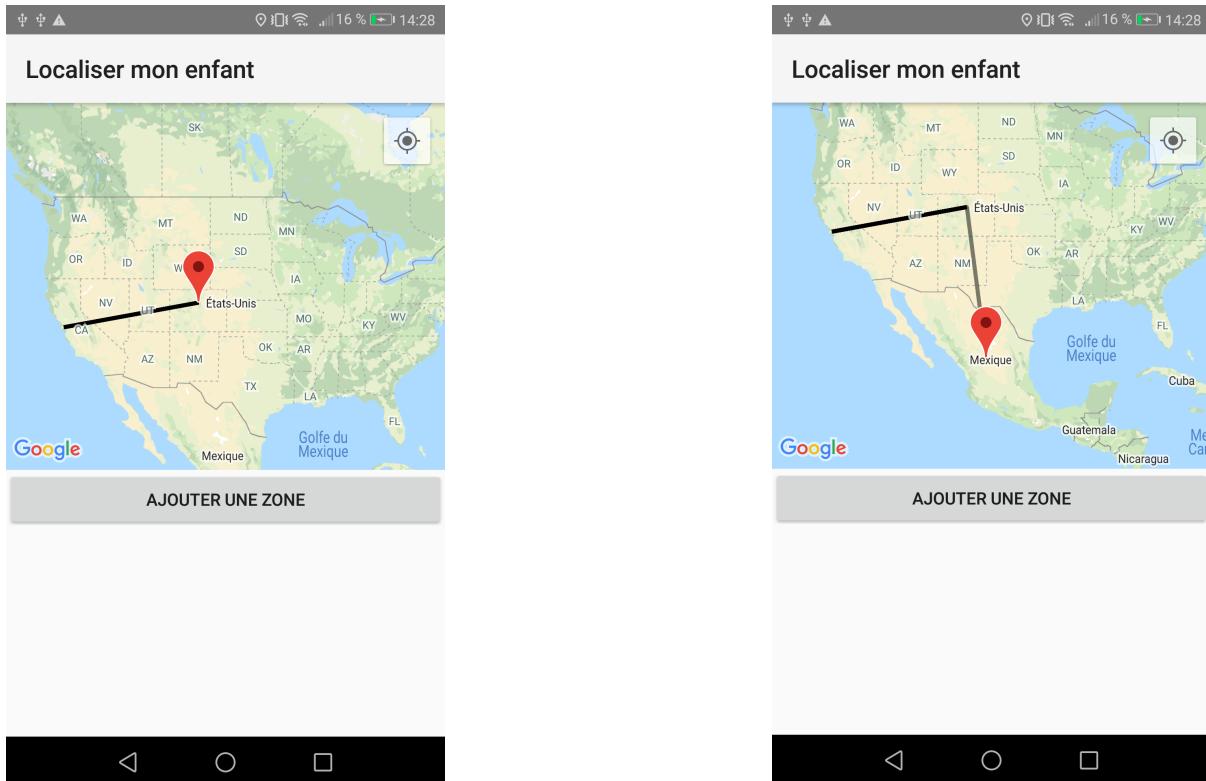


Dans la catégorie contact, le parent pourra examiner tous les contacts contenus dans le téléphone de l'enfant. Il pourra ensuite accéder au SMS échangé avec un fil de discussion style WhatsApp . Il pourra également avoir accès à l'historique des appels échangés.



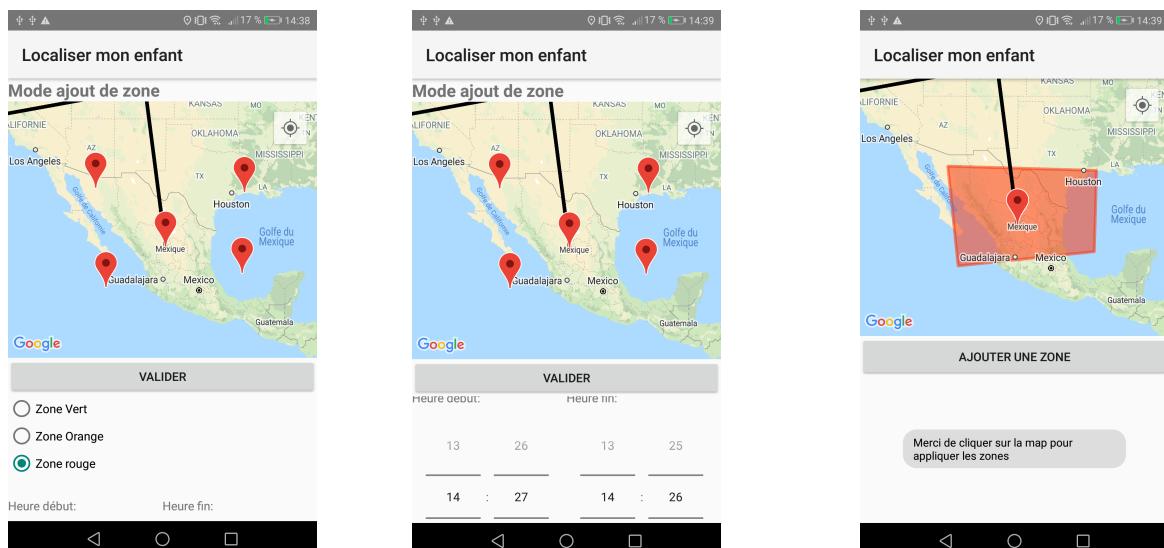
Dans la catégorie carte, le parent pourra suivre la position de l'enfant en quasi temps réel grâce à un service.

Un itinéraire se dessinera derrière l'enfant grâce à des lignes noires reliant ses précédentes positions.



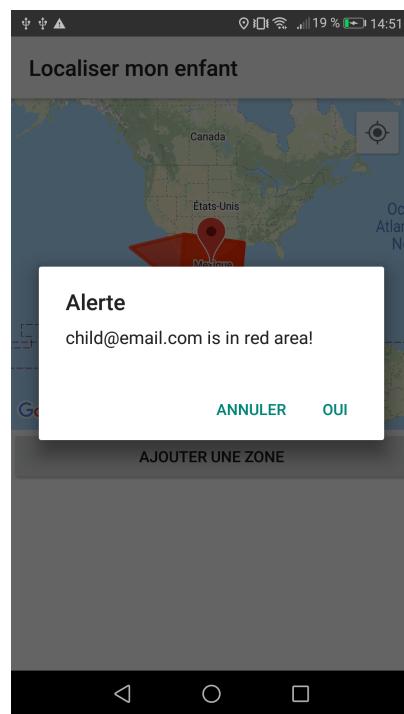
Il pourra ensuite ajouter des zones pour vérifier la présence ou non de l'enfant dans ces dernières entre une heure de début et de fin. Pour créer une zone, l'utilisateur devra appuyer sur le bouton "Ajouter une zone" et cliquer sur la map dans le but de former des polygones. Il peut former 3 types de zones :

- Rouge: Zone à haut risque (Ex: bar, Tabac etc.)
- Orange: Zone à risque modéré (Ex: à l'école après une certaine heure)
- Vert: Zone où l'enfant ne doit pas sortir (Ex: Maison entre 18h et 7h)

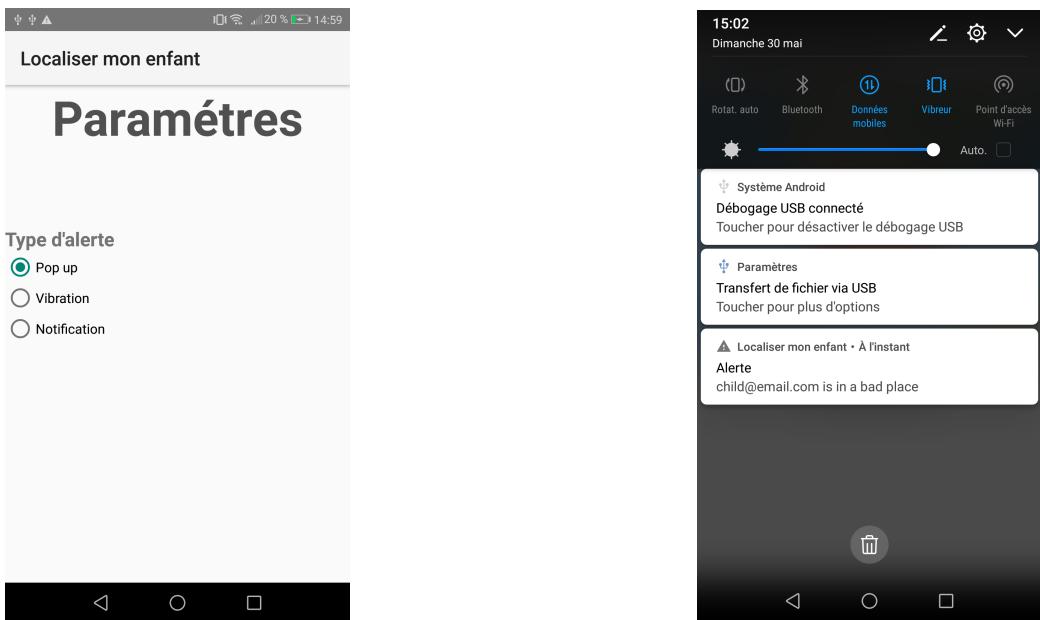




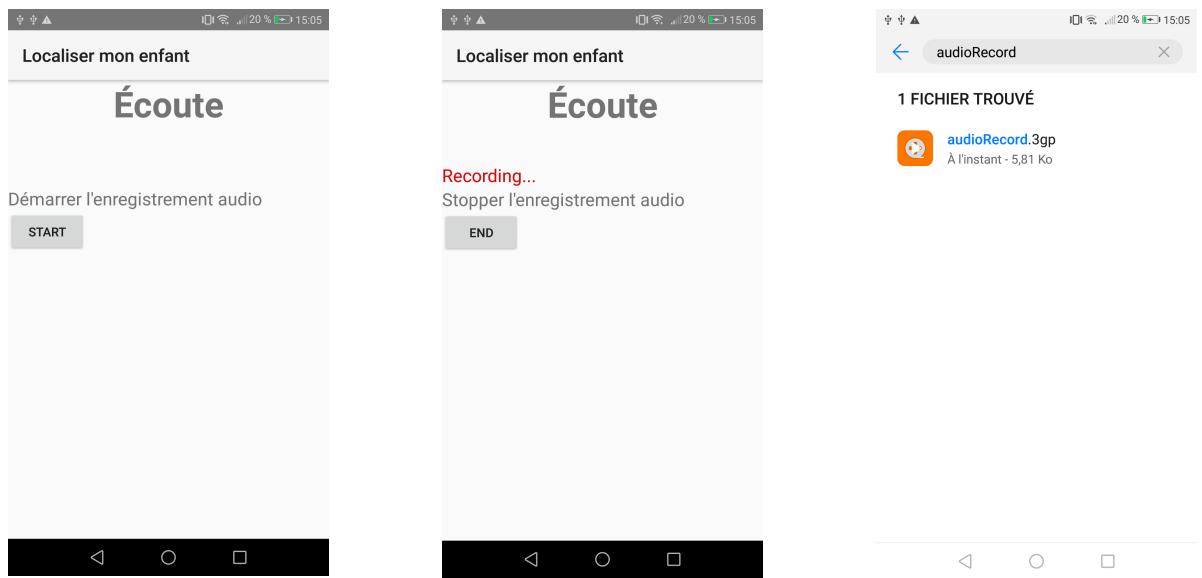
Si l'enfant se trouve dans une zone rouge ou orange, ou s'il se trouve en dehors d'une zone verte, le parent recevra une alerte indiquant l'enfant et la zone concernée.



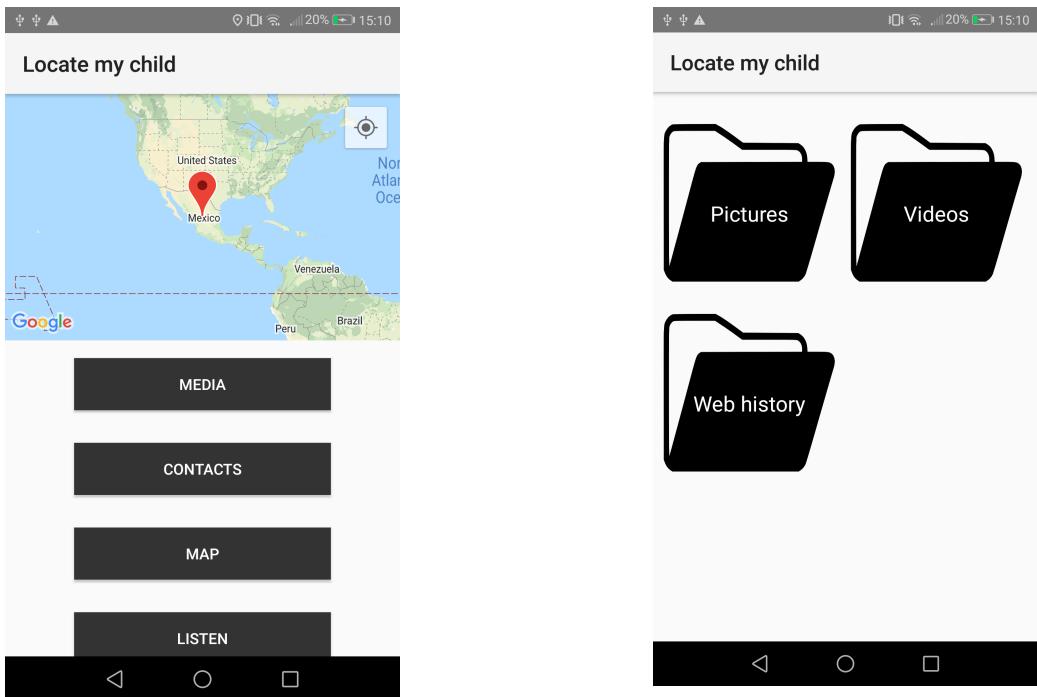
Il peut cependant changer les notifications dans la catégorie paramètre du menu principal pour avoir à la place une alerte par vibration ou par une notification.



Enfin la dernière catégorie écoute est non fonctionnelle mais permet néanmoins d'enregistrer le micro du téléphone du parent.



L'application est également disponible en anglais lorsque l'on passe le téléphone en mode anglais.



II. PARTIE SERVEUR

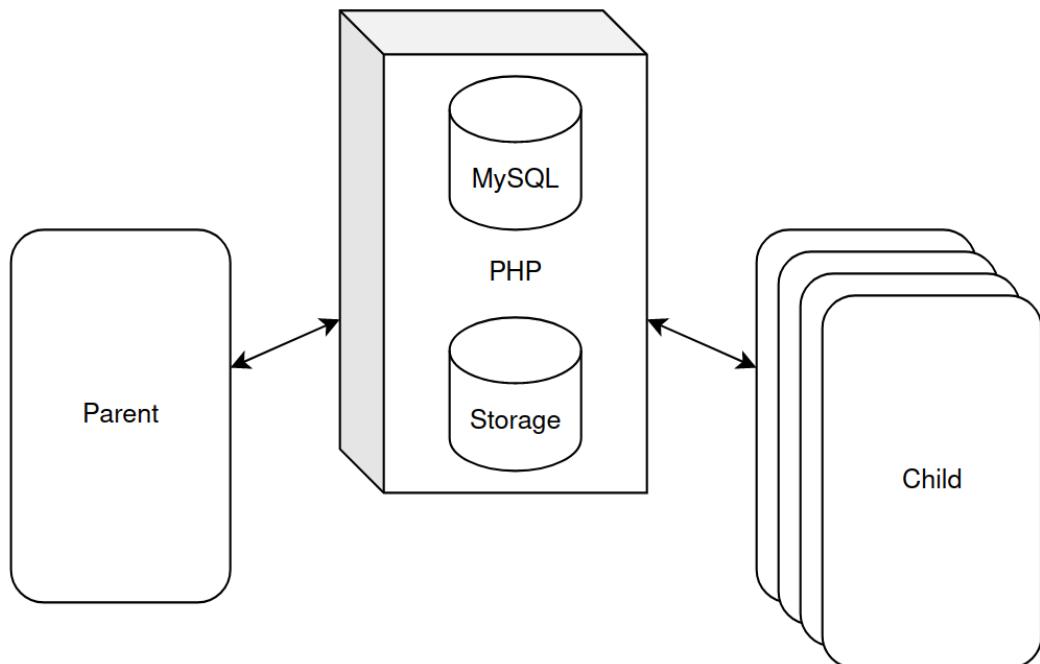
Pour la communication entre l'enfant et le parent, nous avons choisi d'implémenter un serveur qui centraliserait les données. En effet, une connexion directe, bien que moins coûteuse car pas de serveur, n'est pas une solution optimale pour notre problématique. Un téléphone ayant une connexion internet peu stable, cela obligerait le parent à garder l'intégralité des médias et ressources du téléphone de l'enfant en cache s'il voulait accéder au contenu à n'importe quel moment.

C'est pour cette raison que nous souhaitons centraliser les données. Chose que *Firebase* implémente très bien, cependant nous avons choisi de créer notre propre serveur, initiative qui nous a permis d'étudier en détail chaque étape de l'échange de données qui peut se produire entre un client Android et un serveur. Dans cette partie, nous verrons les différents protocoles mis en place, les APIs utilisées et créées pour communiquer avec la base de données.

A. Architecture & Technologies

Nous avons choisi de travailler sur un serveur HTTP, sur du PHP et du MySQL. Cette combinaison "classique" avait l'avantage d'avoir toutes les fonctionnalités dont nous avions besoin à savoir :

- Un moyen simple de communiquer avec l'application (HTTP)
- Une base de données
- Un système de fichiers



Pour la partie client, nous avons eu recours à Volley plus, une fork de Volley qui est une API Android de haut niveau gérant les requêtes HTTP.

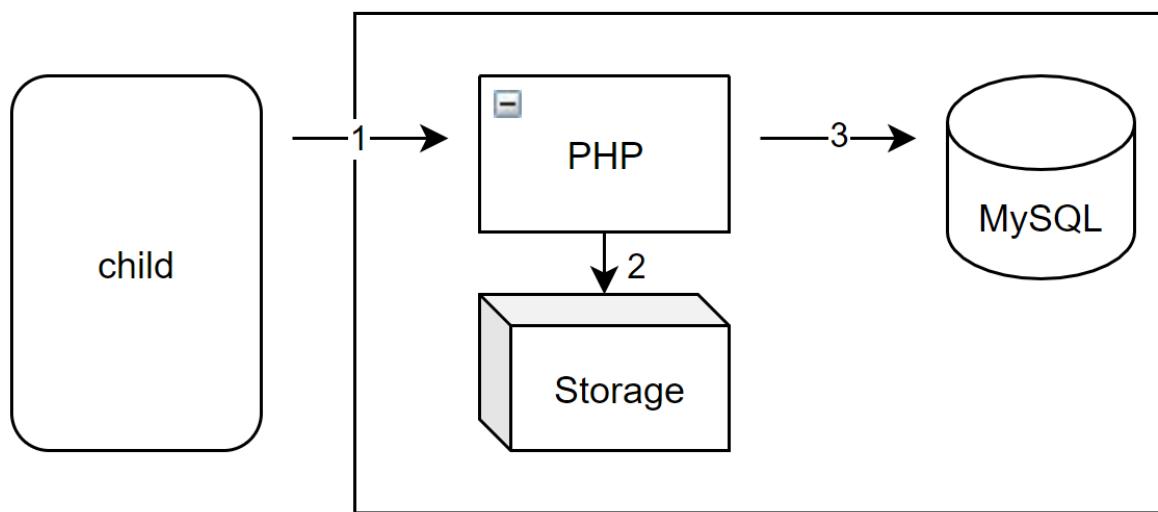
B. Protocoles

Pour la communication entre Android et le serveur, nous avons 2 grands schémas :

Envoi de données

Pour l'envoi de données au serveur, par exemple pour l'envoi des médias de l'enfant:

1. On envoie une requête HTTP au serveur.
2. On stocke les médias reçus
3. On inscrit les données dans la base de données.

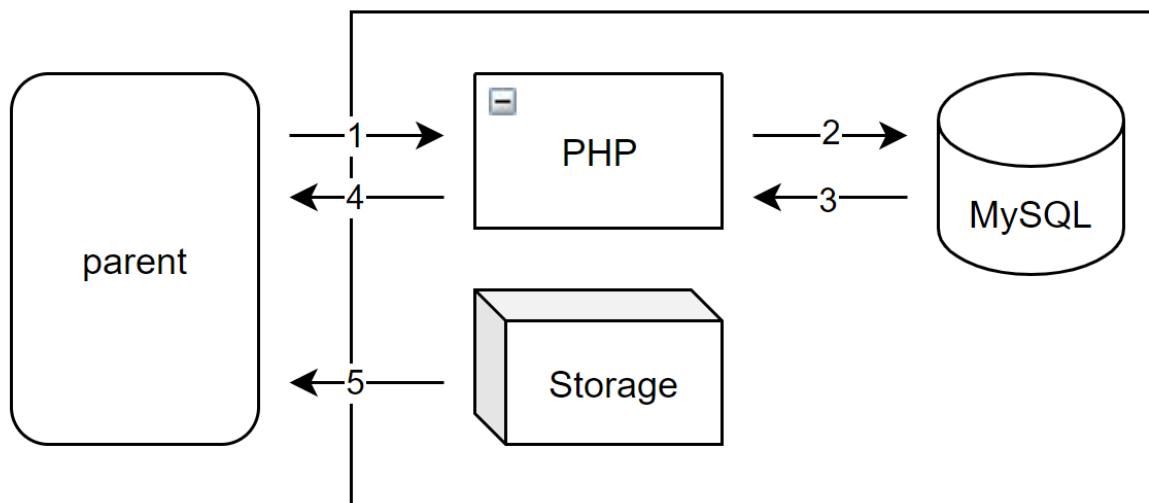


Dans le cas de médias, nous envoyons une requête multi-part, composée d'une requête POST et d'un envoi de fichiers. Si nous n'envoyons pas de fichiers, une simple requête POST est émise.

Réception de données

Pour la réception de données nous avons :

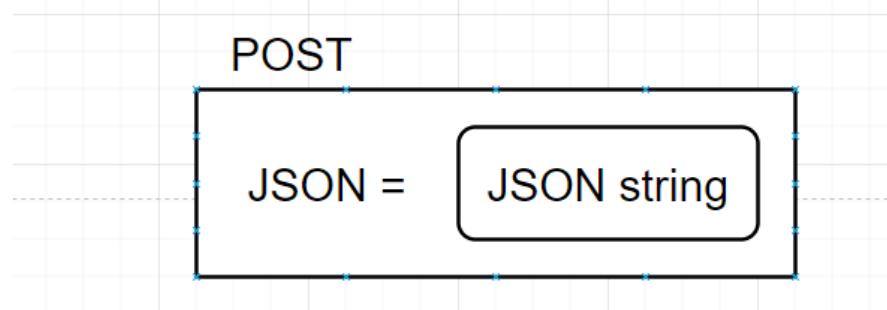
1. L'application envoie une requête au serveur
2. Le serveur interroge la base de données
3. La BDD retourne des données
4. Le serveur répond à l'application
5. Pour les médias, l'application accède ensuite aux ressources.



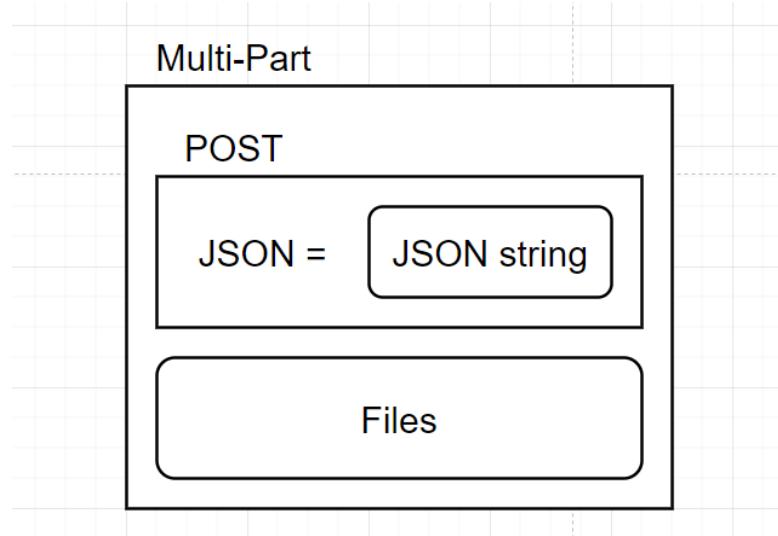
Format des requêtes

Comme cité précédemment, nous communiquons avec le serveur par le biais de requêtes POST et Multi-part. Le contenu de ces requêtes est en revanche écrit en JSON. Cela nous permet d'envoyer et de recevoir des objets complexes, mais simples à lire et à écrire.

Dans le cas d'un envoi de données simple (pour tous les échanges qui ne nécessitent pas d'envoyer un fichier), une simple requête POST est émise.



En revanche, si on veut envoyer un fichier au serveur, principalement pour l'envoi des médias de l'enfant, on doit émettre une requête multi-part composée d'un POST et de nos fichiers.



Le JSON émis par l'application est sous la forme suivante :

```
{  
    "sid"      : 0123456789  
    "type"     : "GetImages"  
    "childId"  : 3  
    ...  
}
```

On à :

- L'ID de session PHP (sid), qui doit être transmis à chaque requête
- type, la commande que l'on veut exécuter sur le serveur, ici récupérer des images
- puis tous les paramètres dont le serveur a besoin pour exécuter la requête.

Le serveur répond alors à son tour du JSON :

```
{  
    "images" : [  
        "https://www.lme.romimap.com/images/image1.png" ,  
        "https://www.lme.romimap.com/images/image2.png" ,  
        "https://www.lme.romimap.com/images/image3.png"  
    ]  
}
```

Dans le cas de *GetImages*, un tableau de liens est retourné. Liens qui serviront à accéder aux médias.

Cas particulier de la connection

L'id de session PHP n'est transmis à l'application qu'après l'authentification de l'utilisateur. La commande *Connexion* ne doit pas transmettre de SID, pour laisser PHP en générer un.

```
{  
    "type"      : "Connexion"  
    "username"  : "Mon@Email"  
    "password"  : "Mon.Mot.De.Passe"  
}
```

Le serveur répond ensuite une valeur de retour et le SID. Si *return* vaut false, l'authentification a échouée

```
{  
    "return"  : true,  
    "sid"     : 0123456789  
}
```

Notes de sécurité

Nous pouvons noter une possible faille de sécurité ici, le mot de passe n'est pas hashé à l'envoi au serveur. Les requêtes sont envoyées en HTTPS donc normalement les données ne sont pas envoyées en clair, mais cela ne les protège que partiellement. Quelqu'un pourrait se faire passer pour le serveur et récupérer les informations en clair à l'autre bout du tunnel en contrôlant le point d'accès auquel le mobile est connecté ou en faisant un spoofing de DNS.

Cela dit, bien que cette possible faille existe, cet échange entre le serveur et l'application à des fins de démonstration est parfaitement acceptable.

Cela vaut aussi pour l'ID de session PHP, ce dernier ne peut être hashé car le serveur a besoin de connaître la donnée d'origine. Pour pallier à ce problème et éviter un vol de SID, nous pouvons mettre en place une chaîne de token. À chaque requête, le serveur retourne un token qui devra ensuite être transmis par le client pour la prochaine requête. Si le SID et le token sont volés, on sera face à 2 cas :

- L'utilisateur émet une requête avant le pirate, ce dernier se retrouve donc avec un token expiré.
- Le pirate émet une requête avant l'utilisateur, ce dernier enverra donc un token expiré au serveur.

Le serveur n'aura donc plus qu'à mettre fin à la session PHP lors de la réception d'un token expiré, ce qui dans notre cas se produira nécessairement car une requête pour connaître la position de l'enfant est émise automatiquement toutes les 10 secondes.

Tous ces soucis pourraient être résolus de manière simple en échangeant des clés à l'inscription. Si l'application possède par défaut la clé publique du serveur et qu'à l'inscription l'application génère une clé et transmet au serveur sa partie publique, les échanges entre l'application et le serveur pourraient être chiffrées garantissant la confidentialité et l'intégrité du message, quant à l'authenticité, on pourrait la garantir en limitant le nombre de connections à 1 appareil et en mettant en place une double authentification.

Cette méthode pourrait (et devrait) aussi être utilisée pour chiffrer les informations sensibles de l'enfant d'appareil à appareil. Ainsi, même le serveur ne pourrait lire les données de l'enfant.

C. Volley plus

Nous avons utilisé pour émettre des requêtes HTTP *Volley plus*, une fork de *Volley*. Cette API de haut niveau nous a permis d'envoyer simplement des données au serveur et possède une structure avec laquelle il est simple d'interfacer. Son fonctionnement est le suivant :

- Un Callback est défini à travers une classe anonyme, y sont spécifiés le code à exécuter en cas de succès, et en cas d'erreur.
- Une instance de requête est créée, on y passe des paramètres et le callback
- Cette instance est inscrite dans une file. Elle sera émise dès que possible.

Notre implémentation est la suivante :

```
➤ L'interface du Callback
interface VolleyCallback {
    public void OnSuccess(JSONObject response);
    public void OnError(VolleyError error);
}

➤ On passe l'URL, les paramètres et un Callback
private static void Post (Context context, String url, final JSONObject params, final VolleyCallback
volleyCallback) {

    ➤ La file de requêtes, on en crée une par requête mais peut être optimisée en en créant une pour
    toutes.
    RequestQueue queue = Volley.newRequestQueue(context);

    ➤ L'instance de la requête, on à une StringRequest, qui retourne la chaîne de caractère que le
    serveur répond.
    StringRequest sr = new StringRequest(Request.Method.POST, url,
        new Response.Listener<String>() {

    ➤ Appel à notre callback OnSuccess quand le serveur réponds, on parse au passage la réponse dans
    un JSONObject
        @Override
        public void onResponse(String response) {
            try {
                volleyCallback.OnSuccess(new JSONObject(response));
            } catch (JSONException e) {
                volleyCallback.OnSuccess(null);
            }
        }
    },
    new Response.ErrorListener() {

    ➤ Appel à notre callback OnError en cas d'erreur.
        @Override
        public void onErrorResponse(VolleyError error) {
            volleyCallback.OnError(error);
        }
    )
    {

    ➤ Définition des paramètres, on ajoute notre JSON à $_POST['json']
    @Override
    protected Map<String, String> getParams () {
        Map<String, String> p = new HashMap<String, String>();
        p.put("json", params.toString());
        return p;
    }
    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        Map<String, String> params = new HashMap<String, String>();
        params.put("Content-Type", "application/x-www-form-urlencoded");
        return params;
    }
};

    ➤ Enfin, on ajoute la requête à la file.
    queue.add(sr);
}
```

D. API serveur

Nous avons implémenté une API pour notre serveur, permettant de communiquer simplement avec. Là où Volley à fait le pont entre l'application et le serveur, l'API serveur fait le pont entre le front-end et Volley. Elle transcrit de manière triviale les commandes que l'on veut exécuter sur le serveur.

Le fonctionnement est le suivant :

- On crée une instance de notre API *Connection*. Lors de la création de l'instance, l'API va nous connecter sur notre serveur. Un callback est propagé pour savoir si la connexion est un succès ou non.
- Une fois connecté, on peut utiliser l'instance *c* de *Connection* pour interfaçer avec le serveur.

Un exemple de méthode de notre API :

➤ L'interface du Callback

```
public interface GetContactsCallback {  
    void Success (ArrayList<Contact> contacts);  
    void Error ();  
}
```

➤ On passe des paramètres simples et une implémentation du callback

```
public void GetContacts (Context context, Child child, final GetContactsCallback  
getContactsCallback) {
```

```
try {
```

➤ On définit le JSON qui sera envoyé au serveur

```
JSONObject params = new JSONObject();  
params.put("sid", sid);  
params.put("type", "ContactList");  
params.put("ChildId", child.id);
```

➤ Appel à la méthode Post décrite précédemment

```
Post(context, CommandURL, params, new VolleyCallback() {
```

➤ On parse le JSON en objets digestes et on propage le callback

```
    @Override  
    public void OnSuccess(JSONObject response) {  
        try {  
            ArrayList<Contact> contacts = new ArrayList<Contact>();  
            JSONArray jsonArray = response.getJSONArray("contacts");  
            for (int i = 0; i < jsonArray.length(); i++) {  
                Contact c = new Contact();  
                JSONObject jo = jsonArray.getJSONObject(i);  
                c.id = jo.getInt("id");  
                c.name = jo.getString("nom");  
                c.num = jo.getString("numero");  
                contacts.add(c);  
            }  
            getContactsCallback.Success(contacts);  
        } catch (Exception e) {  
            getContactsCallback.Error();  
        }  
    }  
}
```

➤ Ou on propage l'erreur

```
    @Override  
    public void OnError(VolleyError error) {  
        getContactsCallback.Error();  
    }  
});
```

```
} catch (Exception e) {  
    getContactsCallback.Error();  
}
```

Toutes les méthodes de l'API suivent ce modèle d'implémentation :

- Définition d'une interface Callback
- On parse nos paramètres dans un objet *JSONObject*
- On appelle *Post* (ou *MultiPartPost*)
- On propage le callback

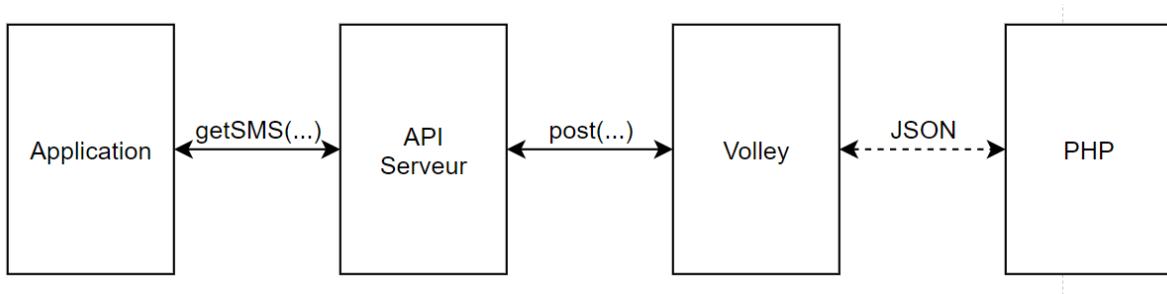
Cela nous permet d'appeler l'API de manière simple depuis le front-end, par exemple pour récupérer et afficher la liste des enfants :

```
> c Notre instance de Connection, on appelle getChildren
Log_in.c.getChildren(getApplicationContext(), new Connection.GetChildrenCallback() {

    > Si succès, on affiche la liste des enfants
    @Override
    public void Success(ArrayList<Connection.Child> children) {
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        recyclerView.setLayoutManager(new LinearLayoutManager(context));
        ChildAdapter monAdapter = new ChildAdapter(children);
        if(monAdapter.getItemCount()== 0) plsAddKid.setVisibility(View.VISIBLE);
        recyclerView.setAdapter(monAdapter);
    }

    > Sinon, on affiche un toast
    @Override
    public void Error() {
        Toast.makeText(getApplicationContext(),
            "Erreur lors du chargement des contacts"
            , Toast.LENGTH_LONG).show();
    }
});
```

La chaîne de données est donc la suivante :



- Le front-end a accès à des méthodes simples
- L'API du serveur s'occupe de transformer les données en un format que le serveur peut comprendre
- Volley se concentre sur l'envoi et la réception de données

Les méthodes sont appelées avec des paramètres simples et les retours sont fait par callback car les requêtes sont traitées de manière asynchrone.

E. Serveur

Le serveur que nous avons mis en place est codé en PHP. Il répond à une liste de commandes simples (GetImages, GetSMS, SendCalls ... etc), gère les connexions à travers des sessions PHP et communique avec la base de données MySQL en utilisant PDO.

En terme de fichiers, sa structure est la suivante :

commands.php	Le script PHP appelé par l'application, est concrètement composé d'un grand <i>switch</i> représentant les commandes disponibles. C'est entre autres le <i>Main</i> de notre serveur.
gettersSetters.php	Les fonctions appelées par <i>commands.php</i>
sql.php	Crée un objet \$pdo connecté à la base de données
images/	stocke les images
videos/	stocke les vidéos

En pseudocode (car le sujet n'est pas le PHP), notre serveur traite les requêtes de cette manière :

```
(Commands.php)
JSON ← POST[json]
si JSON[type] = Ø
    echec
si JSON[sid] ≠ Ø
    ID de session ← JSON[sid]
    démarrer la session

si JSON[type] = Connexion
    On connecte le client
si JSON[type] = Inscription
    On inscrit le client
si JSON[type] = Récupérer SMS
    On récupère les SMS de l'enfant sélectionné
[...]
```

On va premièrement gérer l'ID de session, cette étape est normalement automatique sur un navigateur mais nous devons l'effectuer manuellement ici.

Ensuite, en fonction du type de requête (il s'agit d'une connexion, d'une récupération de SMS ...), nous exécutons des fonctions. Les paramètres de ces fonctions sont stockées dans la variable *JSON*

Globalement le serveur fonctionne simplement. On peut noter qu'il s'agit d'un serveur squelette censé fonctionner uniquement dans les cas valides. Un vrai serveur aurait quelques

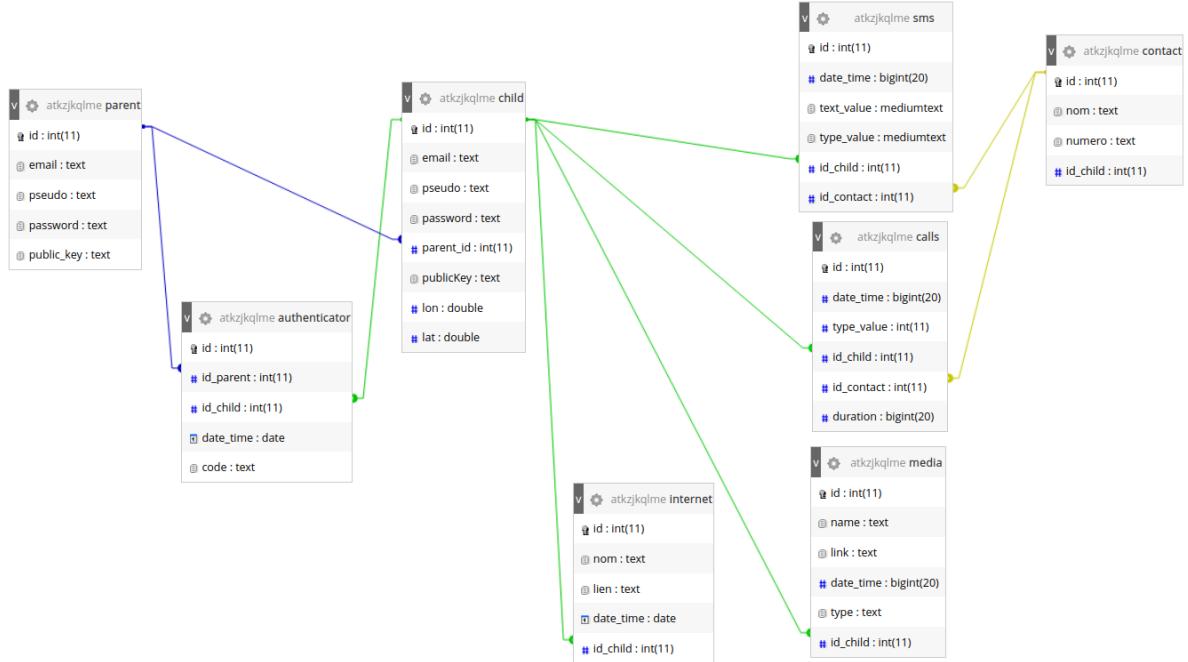
éléments en plus comme une vérification des données avant l'appel aux fonctions, comme par exemple :

*si $JSON[type]$ = Récupérer SMS
si $SESSION[role]$ = parent
et si $JSON[id enfant]$ est défini
et si $JSON[id enfant]$ est un entier
et si $SESSION[id]$ est parent de $JSON[id enfant]$
On récupère les SMS de l'enfant sélectionné*

On vérifie que le compte de cette session est bien un parent, que le paramètre qui nous intéresse est proprement défini et que l'on récupère bien les SMS de notre enfant.

F. Base de données

Pour la base de données, nous avons une BDD MySQL. Nous stockons à l'intérieur toutes données non médias et des liens vers les données médias.



La base est composée de 8 tables, nous n'en utilisons que 6 :

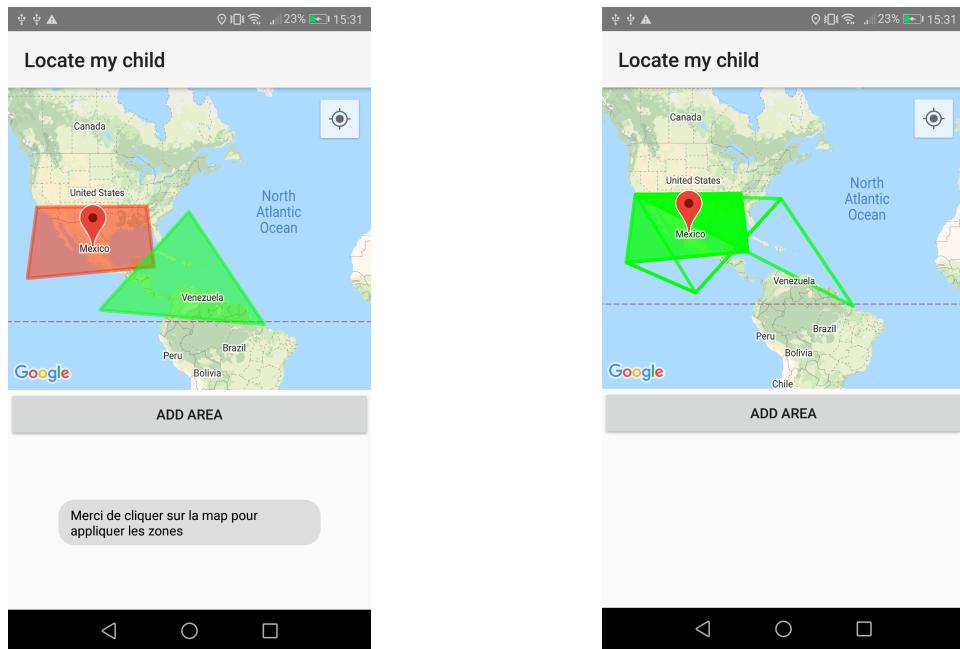
- Parent, table des parents
- Child, table des enfants
- Contact, contacts des enfants
- SMS, sms des enfants
- Calls, historique d'appels des enfants
- Media, liens vers les médias des enfants

Les deux tables authenticator et internet auraient respectivement servi à authentifier le parent par rapport à l'enfant (nous avons utilisé une approche différente) et à stocker l'historique internet de l'enfant.

Conclusion

Nous sommes globalement très satisfaits du travail que nous avons réalisé. Nous avons essayé au maximum d'implémenter toutes les fonctionnalités. Même si certaines ne sont pas complètement opérationnelles comme l'écoute en direct et le système de blocage du téléphone, nous pensons savoir comment faire pour réussir à les implémenter.

Il reste également à corriger quelques bugs comme le stockage local des zones dans le téléphone (actuellement les zones se réinitialisent à chaque redémarrage du téléphone) et également la distinction des zones lorsque l'on quitte et que l'on revient sur l'application map.



De plus, actuellement toutes les données sont envoyées à chaque fois que l'enfant se connecte, nous avons ainsi implémenté un système permettant de récupérer la date de la dernière donnée envoyée afin de ne pas envoyer les données qui sont antérieures à cette date. Cependant cette fonctionnalité ne marchait pas et nous n'avons pas eu le temps de corriger ce problème.

De plus, la lecture des vidéos sur le téléphone parent se fait en téléchargeant entièrement puis en les lisant, il faudrait améliorer ça en lisant la vidéo en streaming avec un protocole comme RTMP, RTSP,MSS,HLS etc..

Il faudrait également passer les services en mode foreground pour être sûr que le service soit toujours en fonctionnement et que si l'enfant éteint et rallume le téléphone, le service ce rallume tout seul.

Du côté du serveur, l'échange de données fonctionne bien. L'implémentation d'une API pour ce dernier et la liaison avec le serveur avec Volley s'est faite sans (trop) d'accrocs, mais le nombre de commandes à implémenter à été le facteur qui nous a le plus ralenti. Il persiste donc encore des failles de sécurité, non pas à cause de l'architecture du serveur mais parce que ce dernier est encore à l'état de squelette. Si nous venions à reprendre le projet, le prochain ajout serait certainement l'encodage d'appareil à appareil et la vérification des données reçues côté serveur.

Malgré tout, le serveur fonctionne bien et inscrit ou récupère toutes les données qu'on peut lui envoyer ou demander. Le serveur aurait certainement été plus rapide à créer sur une technologie comme Firebase et nous aurions peut être dû considérer cette option mais nous restons satisfaits de notre back-end fait maison. L'élaboration d'un serveur et d'une API soulève autant de problématique en termes d'architecture que de sécurité et nous ne regrettions absolument pas le ce choix bien qu'il nous ait pris beaucoup de temps.