Rapport
Projet de programmation de mobiles

Lucas Thietart Hugo Le Clainche
2 SN R 2

Département Sciences du Numérique - Deuxième année
Option Réseaux
2023-2024

Table des matières

1 Introduction

Dans ce rapport nous allons présenter le travail effectué au cours des séances de travaux pratiques et de projet. Dans ces TP nous avons utilisé android studio avec Java pour programmer une application mobile dont le principe est de simuler les actions d'une télécommande qui actionne les appareils connectés d'une smarthouse.
L'objectif de ce rapport est de présenter l'architecture de notre projet afin de faciliter la prise en main et la compréhension de l'application. Nous reviendrons sur les difficultés rencontrées lors du développement de l'application.

2 Plan de communication

Voici ci dessous un schéma qui représente des interactions entre les différents thread et classes de notre application. Nous y ferons appel régulièrement dans ce rapport. L'application est identique côté client et côté serveur.

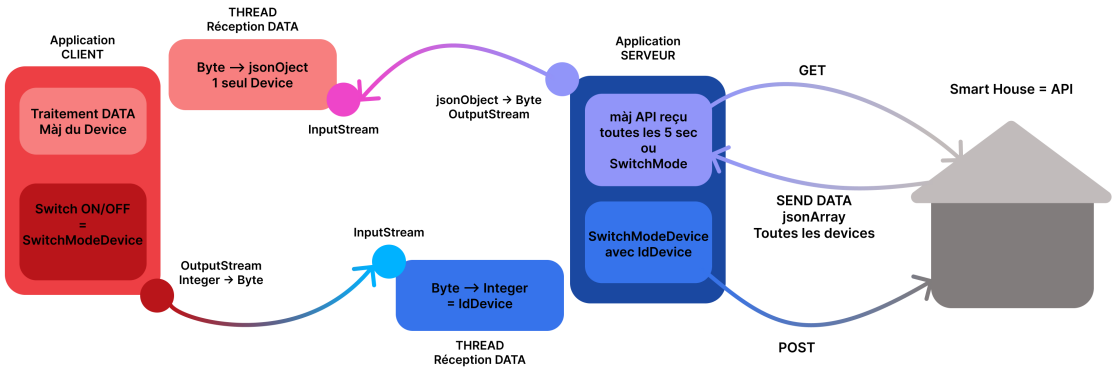

FIGURE 1 – Plan de communication de notre application

3 Architecture

Lorsque l'on ouvre l'application, la *MainActivity* se lance. On y trouve deux boutons de connexion (Serveur ou Client). Après avoir cliqué sur un des deux boutons de connexion, nous lançons une nouvelle activité (*ClientActivity* ou *ServeurActivity*). Le lancement de cette nouvelle activité permet de bloquer la possibilité de sélectionner l'autre bouton de connexion (cf figure ci-dessous). Cependant, nous aurions aussi pu désactiver la méthode onClick du bouton.

```
btnClient = findViewById(R.id.Client);
btnClient.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Lancer l'activité ClientActivity
        Intent intent = new Intent(MainActivity.this, ClientActivity.class);
        startActivity(intent);
    }
});

btnServeur = findViewById(R.id.Serveur);
btnServeur.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Lancer l'activité ServeurActivity
        Intent intent = new Intent(MainActivity.this, ServeurActivity.class);
        startActivity(intent);
    }
});
```

FIGURE 2 – Intent de l'activité Serveur

3.1 Communication bluetooth

Création du Socket pour la communication

Dans chacune des deux activité lancées par les boutons de la main activity on lance un Thread qui va obtenir le socket afin de communiquer en Bluetooth. Nous nous sommes inspirés du guide Android Studio Developers pour réaliser ces fonctions. L'une des difficultés rencontrées a été de récupérer cette socket bluetooth. Car nous obtenons le socket dans le thread, ainsi après avoir récupérer le socket nous appelons une nouvelle activité mais à partir de notre activité précédente et non du thread. On perd alors le socket de notre activité puisque c'est une variable local, appartenant uniquement à une activité. Pour y remédier, on met ce socket en variable globale en passant par l'intermédiaire d'une classe public *BluetoothSocketManager* avec comme attribut accessible le socket. (on devrait d'ailleurs utiliser une méthode pour récupérer le socket)

```java
public class BluetoothSocketManager {
    private static BluetoothSocket socket;

    public static BluetoothSocket getSocket() {
        return socket;
    }

    public static void setSocket(BluetoothSocket bluetoothSocket) {
        socket = bluetoothSocket;
    }
}
```

FIGURE 3 – Socket manager

Côté serveur : *AcceptThread*

Lorsque que nous voulons utiliser l'application du côté serveur, le thread "AcceptThread" est lancé. Il va écouter les demande de communication et accepter uniquement l'appareil possédant l'UUID valide. Nous avons ici généré l'UUID du serveur manuellement et l'avons aussi donné au client pour qu'il envoi la bonne information.

```java
private static final UUID MY_UUID = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
// ...
mmServerSocket = bluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
```

FIGURE 4 – Socket serv

Dans cette partie du code il y a une grande partie de gestion de permission. Il faut alors donner toutes les permissions à notre application en ajoutant cela dans notre *manifest*.

Dans ce thread on écoute en continu, les demande d'ouverture de communication avec le bon UUID . Si la communication est possible on obtient notre socket.

```java
private static final UUID MY_UUID = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
// ...
mmServerSocket = bluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
```

FIGURE 5 – Socket serv

Une fois que notre socket est accepté, nous pouvons lancer la nouvelle activité depuis le Thread. Tout d'abord, on place notre socket en variable globale puis il faut rajouter notre contexte actuel (notre activité) pour permettre la transition d'un Intent au nouveau.

→ On lance *ServeurDevicesActivity*

Côté client : *ConnectThread*

Lorsque que nous voulons utiliser l'application côté client (ce qui est le cas normal) le thread ConnectThread est lancé. On va alors demander l'ouverture de la communication en signant la demande avec l'UUID que nous avons rentré. Pour le reste c'est le même concept que pour le serveur avec la gestion de permission.

```
mmSocket = mmDevice.createRfcommSocketToServiceRecord(MY_UUID);
mmSocket.connect();
// Connection successful, perform further actions if needed.
// On met le socket en variable globale pour pouvoir l'utiliser dans la prochaine activité !
BluetoothSocketManager.setSocket(mmSocket);
// START NEW ACTIVITY
Intent intent = new Intent(mContext, ClientDevicesActivity.class);
mContext.startActivity(intent);
```

FIGURE 6 – Socket client

Nous ne faisons pas de boucle pour ouvrir la communication puisque nous attendons uniquement la réponse de notre requête.

→ On lance *clientDevicesActivity*

3.2 Envoie de données

La partie envoie de données entre le serveur et la BDD en ligne à été traitée intégralement lors des TP. Une requête http GET permet au serveur d'obtenir les informations de la BDD actualisées toutes les 5s (class *ServeurDevicesActivity*).

```
public void RequestDevices() {
    RequestQueue queue = Volley.newRequestQueue( context: this);
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET, url,
            jsonRequest: null, requestArraySuccessListener(), requestArrayErrorListener());
    queue.add(jsonArrayRequest);
}
```

FIGURE 7 – Socket client

3.2.1 Communication Serveur to Client

Envoie de la data des devices

Une fois les information récupérées grâce à l'API, *le serveur push automatiquement l'entiereté des données mises à jour de la BDD vers notre client.*

Nous avons eu des difficulté à mettre en place ce fonctionnement de mise à jour complète et régulière des données du fait de la taille des données à passer dans le socket. Nous avons donc choisi d'envoyer les informations device par device au client, c'est ce qu'illustre l'extrait de code ci-dessus. En effet, nous reprenons la partie mise à jour de l'interface qui s'occupe un à un des devices pour envoyer l'information.

```
//for (int i = 0; i < data_recup.length(); i++) {
for (int i = 0; i < data_recup.length(); i++) {
    // 4 Car on sait que ça rentre dans le buffer
    // On envoie les données au client pour chaque device
    // Progressivement pour éviter les problèmes de buffer
    byte[] buffer = data_recup.getJSONObject(i).toString().getBytes();
    try {
        mmOutStream.write(buffer);
```

FIGURE 8 – MAJ des données client

Nous aurions aussi pu reformer le JSON et l'envoyer au client (méthode *arraydevicelistener* de la classe *ServeurDevicesActivity*). Car nous avons constaté bien après qu'une grande majorité de l'information envoyée était uniquement les noms des différents attributs de nos JSONObject. Ainsi, donner des noms courts ou numéroter les attributs était une bonne solution.

Réception de la data des devices

De la même façon que côté serveur, le client écoute en permanence le socket bluetooth. Lorsque des données y sont lues le client met à jour l'affichage des devices avec la méthode *UpdateDeviceViews* (Le processus de création et de mise à jour de l'affichage des devices est identique côté serveur).
Comme il reçoit un a un les devices, il va mettre à jour chaque device. Or, lors de la première fois que le serveur envoie un device il faut créer cette vue. On va donc tester à chaque réception d'un device si son ID est dans le liste des devices enregistrées (On fera alors une mise à jour) sinon, on l'ajoute à la liste et on crée une vue.

```
while (true) {
    try {
        // Il faut lire tout le buffer
        // Tout mettre dans une liste
        // mais n'extraire qu'un seul Json à la fois
        // Pour ensuite faire updateDeviceViews


        bytes = mmInStream.read(buffer);

        String json = new String(buffer, offset: 0, bytes);
        JSONObject jsonObject = new JSONObject(json);

        // Mise à jour des vues
        handler.post(() -> updateDeviceViews(jsonObject));
```

FIGURE 9 – Reception des mises à jour de la BDD

3.2.2 Communication Client to Serveur

Envoie de la requête (POST)

Lors d'un clique sur le bouton switch d'un devices côté client, celui va lancer la méthode *SwitchModeDevice*.

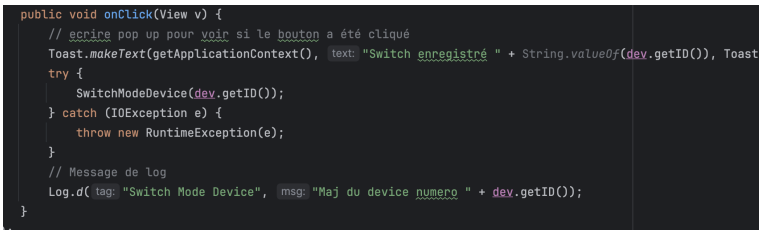

FIGURE 10 – Ecoute du click sur un switch

La méthode *SwitchModeDevice* va écrire l'ID du Device concerné dans le socket client.

```
public void SwitchModeDevice(int id) throws IOException {
    // Convertir int en string puis en byte !
    byte[] buffer = String.valueOf(id).getBytes();
    // ENVOI PAR SOCKET
    mmOutStream.write(buffer);
}
```

FIGURE 11 – Ecriture dans le socket côté client

Réception de la requête (POST)

Le serveur écoute en permanence le socket client. Lorsque qu'une information (ID du device) est lue, le serveur appelle la méthode *SwitchModeDevice* avec en paramètre l'ID du device à switch ON ou OFF. Cette méthode transmet la mise à jour à la BDD via une requête POST. Comme dans notre méthode, la requête POST est toujours suivie d'un GET, on aura une mise à jour immédiate de la BDD sur le serveur et donc du Client.

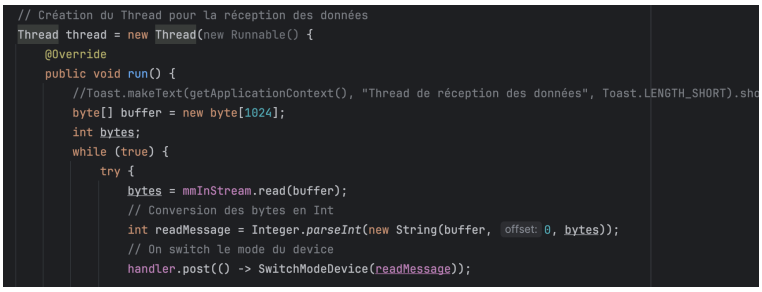

FIGURE 12 – Reception des mises à jour du client

4 Conclusion

Ce projet nous a permis d'approfondir nos compétences en développement mobile avec Android Studio, en particulier dans la gestion de la communication Bluetooth et l'interaction avec des bases de données via des requêtes web. Nous avons également acquis une expérience précieuse en résolution de problèmes et en optimisation de l'application.