INP TOULOUSE ENSEEIHT

Projet Programmation des mobiles

Projet Connexion Bluetooth et maison connectée

Département Sciences du Numérique
Parcours systèmes de télécommunication
2ème Année 2023/2024

Auteurs : CHEMLAL Ibrahim
BTEICH Ramy

Introduction :

Nous souhaitons développer une application qui peut être installée sur deux smartphones. L'objectif est de réaliser une communication **Bluetooth** entre deux smartphones, un qui prend le rôle du client et l'autre qui prend le rôle du serveur.

Cette communication permet au client de contrôler les équipements de la maison connectée à distance.

Comment assurer au client l'accès aux éléments de la maison connectée et leur contrôle ?

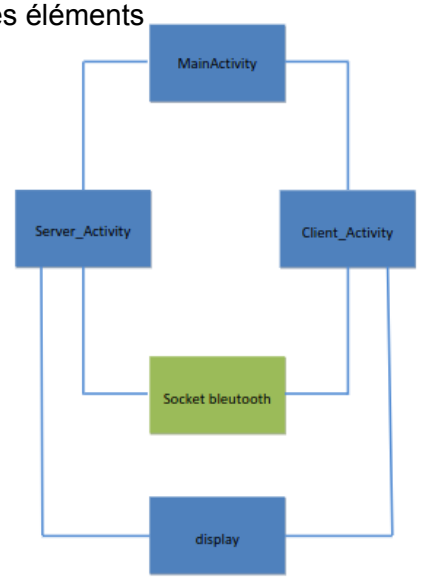
Comment gérer la communication client-serveur ?

Pour répondre à ces questions problématiques, nous allons représenter l'architecture de notre application et détailler l'implémentation et le fonctionnement de ses éléments

Description de l'application :

L'architecture représente la conception logique de notre application.

Nous détaillons ci-après la conception des 4 activités: MainActivity, ServerActivity, ClientActivity et display ainsi la mise de communication entre Client et Serveur et le client via les Sockets Bluetooth



1 - MainActivity :

Au lancement de notre application on instancie les boutons pour inviter l'utilisateur à choisir entre Client et Serveur et on les met sur écoute **(figure 1) .** A condition que le smartphone soit connecté au Bluetooth, sinon un message **" Please turn on Bluetooth "** pour inviter l'utilisateur à l'activer **(figure 2)**.

Le code qui permet de permettre à l'utilisateur de choisir entre le role CLIENT ou role SERVER et lui notifier à d'activer bluetooth est présenté en **figure 3**

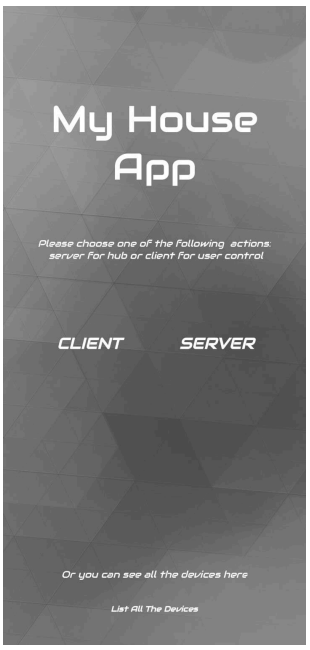


figure 1 : MainActivity

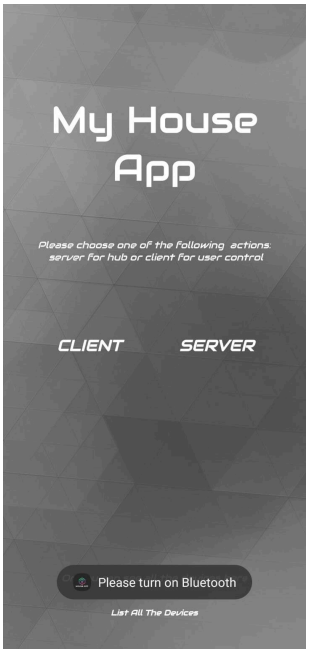


figure 2 : Notification pour activer Bluetooth

```
public void onClick(View v) {
    if (v.getId() == R.id.display_but) {
        // Lancer l'activité d'affichage des appareils
        Intent displayIntent = new Intent(this, display.class);
        displayIntent.putExtra("name", "DEVICES");
        startActivity(displayIntent);
    } else if (v.getId() == R.id.client) {
        // Vérifier les permissions Bluetooth et lancer l'activité client
        if (bluetoothpermission()) {
            if (ensureBluetoothEnabled()) {
                clientactivity();
            } else {
                Toast.makeText(this, "Please turn on Bluetooth", Toast.LENGTH_SHORT).show();
            }
        }
    } else if (v.getId() == R.id.server) {
        // Vérifier les permissions Bluetooth et lancer l'activité serveur
        if (bluetoothpermission()) {
            if (ensureBluetoothEnabled()) {
                serveractivity();
            } else {
                Toast.makeText(this, "Please turn on Bluetooth", Toast.LENGTH_SHORT).show();
            }
        }
    }
```

figure 3 : Choix entre CLIENT et SERVER et notification d'activation du Bluetooth

2 - display :

L'activité qui centralise le projet. C'est le fruit des TPs réalisés pour pouvoir accéder à l'ensemble des machines connectés dans la maison.

l'URL : https://www.bde.enseeiht.fr/~bailleq/smartHouse/api/v1/devices/42 permet d'accèder à l'ensemble des équipements en indiquant le vecteur (ID, BRAND, MODEL, NAME, TYPE, AUTONOMY, STATE, DATA) pour chacun d'eux.

Deux notions sont indispensable à ce stade, les deux requêtes HTTP : **GET** et **POST**

GET : Pour extraire la liste des équipements et leurs informations et pouvoir les organiser dans l'activité display pour une meilleur visibilité pour l'utilisateur **(figure 4)**

POST : Pour pouvoir visualiser la modification des états des machines au niveau de la page web

Les codes pour les requêtes GET et POST sont illustrés dans les deux figures **(figure 5)** et **(figure 6)**

figure 4 : display

```
queue = Volley.newRequestQueue(this); // Initialiser la file de requêtes
JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(url, this::fetching,
 error -> {
    // Gestion des erreurs ici si nécessaire
});

// Mises à jour périodiques avec Handler
handler = new Handler();
runnableCode = new Runnable() {
    @Override
    public void run() {
        queue.add(jsonArrayRequest); // Ajouter la requête à la file
        handler.postDelayed(this, 10000); // Délai de 10 secondes
    }
};
handler.post(runnableCode); // Commencer les mises à jour périodiques

queue.add(jsonArrayRequest); // Ajouter la requête initiale à la file
```

figure 5 : GET

```
public void onClick(View v) {
    currentStatus = !currentStatus; // Basculer l'état
    toggleButton.setText(currentStatus ? "ON" : "OFF");

    // URL pour la requête POST
    String toggleUrl = "https://www.bde.enseeiht.fr/~bailleq/smartHouse/api/v1/devices/42";
    StringRequest postRequest = new StringRequest(Request.Method.POST, toggleUrl,
            response -> {
                // Gérer la réponse
            },
            error -> {
                // Gérer l'erreur
            }) {
```

figure 6 : POST

3 - Client_Activity :

Lorsqu'on choisit le bouton CLIENT **(figure 1 + figure 3)**, l'utilisateur peut alors visualiser l'ensemble des équipements connectés comme qu'on peut peut voir en **figure 7**.

L'utilisateur peut donc modifier l'état des équipements et recevoir des mises à jour des informations régulièrements (Processus implémentée au niveau de l'activité display)

4 - Server_Activity :

Lorsqu'on choisit le bouton SERVER **(figure 1 + figure 3)**, on mentionne à l'utilisateur l'attente d'une connexion de la part d'un client. Il s'agit de l'attente d'une requête Bluetooth envoyée par le client.

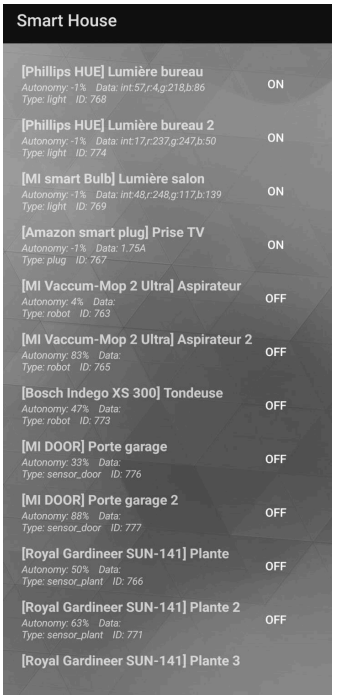


figure 7 : Client_Activity

5 - Socket_Bluetooth :

Il s'agit pas d'une activité comme les 4 mentionnées précédemment. C'est pour cela que nous avons choisi une couleur différente pour le représenter au niveau de l'architecture.

Notre raisonnement était le suivant : Lorsque le client choisit d'allumer ou d'éteindre un équipement, ceci sera traduit par un click sur bouton qu'on peut déterminer son **Id** via **View.getId().** Ensuite, le client crée un socket Bluetooth pour envoyer une requête sous forme d'un message contenant l'id de la machine. Donc, le rôle du serveur est d'inverser l'état de la machine.

La première partie du raisonnement a été implémentée au niveau de **display** et la fonction **applyCommand** implémentée au niveau du client . La seconde partie est implémentée au niveau de **Server-Activity** et **Client_Activity**.

Le serveur crée un socket Bluetooth pour réaliser le listing sur le port Bluetooth de l'application. Une fois que le client envoie une requête, le serveur accepte et crée le véritable socket de communication pour les flux INPUTSTREAM et OUTPUTSTREAM. De l'autre côté, une fois l'utilisateur partie client presse un bouton, le client crée un socket Bluetooth et écrit au niveau de OUTPUTSTREAM l'id de l'équipement.

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        BluetoothServerSocket serverSocket = null;

        try {
            // Vérification de la permission BLUETOOTH_CONNECT
            if (ActivityCompat.checkSelfPermission(this, "android.permission.BLUETOOTH_CONNECT") !=
PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this, new String[]{"android.permission.BLUETOOTH_CONNECT"},
1);
                return;
            }

            // Création d'un serveur Bluetooth avec un UUID spécifique pour identifier le service
            UUID uuid = UUID.fromString("5289df73-7df5-3326-bcdd-22597afb1fac");
            serverSocket = bluetoothAdapter.listenUsingRfcommWithServiceRecord("MonServeur", uuid);

            BluetoothServerSocket finalServerSocket = serverSocket;
            // Création d'un thread pour accepter les connexions entrantes
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        // Accepter une connexion entrante
                        clientSocket = finalServerSocket.accept();
                        outputStream = clientSocket.getOutputStream();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }).start();
```

figure 9 : Gestion des sockets côté serveur

```
private void connectToServer() {
    // Initialiser l'adaptateur Bluetooth
    BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    if (bluetoothAdapter == null) {
        // Vérifier si Bluetooth est supporté
        Log.e(TAG, "Bluetooth not supported on this device");
        return;
    }

    // Obtenir l'appareil serveur par son adresse MAC
    BluetoothDevice serverDevice = bluetoothAdapter.getRemoteDevice(SERVER_MAC_ADDRESS);
    new Thread(new Runnable() {
        @Override
        public void run() {
            BluetoothSocket socket = null;
            try {
                // Créer et connecter un socket Bluetooth
                socket = serverDevice.createRfcommSocketToServiceRecord(MY_UUID);
                socket.connect();
                InputStream inputStream = socket.getInputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

                // Lire les commandes en boucle et les appliquer
                while (true) {
                    String command = reader.readLine();
                    if (command != null) {
                        runOnUiThread(() -> applyCommand(command));
                    }
                }
            } catch (Exception e) {
                // Gestion des erreurs de connexion
                Log.e(TAG, "Error connecting to server", e);
                if (socket != null) {
                    try {
                        socket.close();
                    } catch (Exception closeException) {
                        Log.e(TAG, "Error closing socket", closeException);
                    }
                }
            }
        }
    }).start();
```

figure 8 : Gestion des sockets Bluetooth coté client

Malheureusement, nous n'avons pas pu faire fonctionner cette communication correctement.

Conclusion :

Ce projet a mis l'accent sur plusieurs aspects de programmation au sein des mobiles Android. En partant de la notion la plus basique de la programmation de mobiles Activity vers le contrôle des équipements d'une maison connecté à distance via les sockets bluetooth.

Les points de réflexes de ce projet permettent de générer des points de réflexes pour d'autres projets plus complexes, notamment nous sommes actuellement dans le monde des internets des objets.

Nos remerciements à Mme.Katia JAFFRES-RUNSER pour son encadrement et son construction de projet instructif.