



UNIVERSITE MOHAMMED V ÉCOLE NATIONALE SUPÉRIEURE
D'INFORMATIQUE ET D'ANALYSE DES SYSTÈMES
ENSIAS - RABAT

FILIÈRE SÉCURITÉ DES SYSTÈMES D'INFORMATION

Développement mobile

Recherche du meilleur répéteur-wifi dans un réseau SPAN

Réalisé par :
SABIR YASSINE
AMAR ILYAS

Sous l'encadrement de :
M. BOUZIDI DRISS

Année Universitaire 2022 - 2023

Résumé

À travers ce projet, nous avons réussi à créer un réseau SPAN connectant les téléphones portables possédant la technologie WiFi-Hotspot et résidant à proximité.

Dans un premier moment de ce rapport, nous avons introduit brièvement le but de ce projet ainsi que les spécifications demandées dans le cahier des charges de l'application. Ensuite, nous avons décrit dans différentes sections les besoins fonctionnels ainsi que les besoins non fonctionnels, et nous avons conclu cette partie réservée à l'analyse des besoins par une étude de faisabilité technique.

Plus loin dans ce rapport général, nous avons effectué une étude conceptuelle de notre application dans laquelle nous avons décrit les fonctionnalités de l'application, puis nous avons conçu une architecture de l'application.

Finalement, nous avons fait une démonstration de notre application en montrant des captures d'écran des différentes fonctionnalités de cette application, ainsi que des extraits de code responsables de ces fonctionnalités.

Table des matières

Introduction générale	5
1 Analyse de besoin	6
1.1 Description des besoins fonctionnels	6
1.2 Description des besoins non fonctionnels	6
1.3 Étude de faisabilité technique	6
2 Conception de l'application	8
2.1 Description des fonctionnalités de l'application	8
2.2 L'architecture de l'application	8
3 Implémentation de l'application	9
3.1 Lister les réseaux WIFI disponibles	9
3.2 Avoir les informations et se connecter à un réseau WIFI	13
3.3 Lancement d'un réseau WIFI	19
3.4 Les permissions données à l'application	24
4 Conclusion	25

Liste des figures

1	Réseau SPAN généré par l'application	7
2	Diagramme de cas d'utilisation de notre application	8
3	Architecture de l'application	9
4	Attributs du classe reseauWIFI	10
5	Distinction entre les deux types de WIFI	11
6	Proposer à l'utilisateur de donner la permission de localisation . . .	11
7	Menu donnant à l'utilisateur la possibilité de donner l'accès à la localisation à l'application	11
8	Assurer que le WIFI et la localisation sont activés	12
9	Notification contextuel affiché	12
10	Liste des réseau WIFI disponibles	13
11	Informations réseau	14
12	Réseau sécurisé et réseau ouvert	14
13	Se connecter à un réseau WIFI	15
14	Se connecter à un réseau	15
15	Base de données des réseau WIFI	16
16	Enregistré le mot de passe d'un réseau à un réseau	16
17	Mise à jour d'un mot de passe	17
18	Gestion des mots de passe	17
19	Récupération d'informations depuis un autre smartphone	18
20	Récupérer les informations d'une machine à l'aide des sockets . . .	18
21	Code de visualisation des information WIFI	19
22	Récupération d'informations d'un réseau	19
23	Serveur en écoute dans le port 7800	20
24	Service du serveur fonctionne en arrière plan	21
25	Code de notification	21
26	La notification	21
27	Arrêt de serveur	21
28	Lancement d'un point d'accès	22
29	Activité de lancement de WIFI et du serveur	22

30	Ridirection vers les paramètres de WIFI	23
31	Notification lors du lancement du serveur	23
32	Arrêt du serveur	24
33	Les permissions données à l'application	24

Introduction générale

De nos jours, les smartphones sont de plus en plus populaires. Il est rare de trouver une maison dont les propriétaires ne possèdent pas de smartphone. Ils ont aidé les gens à échanger des données dans les réseaux privés en intégrant la fonctionnalité de Bluetooth, ainsi que dans les réseaux publics en offrant la possibilité de se connecter aux réseaux soit en utilisant les données mobiles, soit en utilisant le réseau WIFI.

Les smartphones récents incluent une fonctionnalité assez importante, celle de se connecter à un réseau WIFI et de partager la connexion avec d'autres machines à l'aide d'un hotspot. Dans ce cas, le smartphone fonctionne comme un répéteur ou un point d'accès et permet de garantir que les machines distantes d'un point d'accès d'un réseau WIFI auront accès à Internet avec des performances de qualité.

Le projet que nous sommes censés développer est similaire à ce concept intégré dans les smartphones récents. L'application mobile que nous allons développer permettra à un smartphone de se connecter à un réseau WIFI, de créer un point d'accès mobile sans fil (hotspot) et, en plus de cela, d'échanger des informations de qualité de service entre les machines. Cela permettra à l'utilisateur de baser ses choix sur les critères les plus importants pour choisir le meilleur réseau WIFI disponible pour se connecter à son réseau.

1 Analyse de besoin

1.1 Description des besoins fonctionnels

L'application mobile que nous sommes censés développer doit être capable de lister les réseaux WIFI disponibles et de permettre aux utilisateurs de se connecter à un réseau WIFI en utilisant des protocoles d'authentification tels que WEP et WPA. En outre, notre application doit offrir à l'utilisateur la possibilité de créer un nouveau réseau WIFI et de partager des informations de la machine avec d'autres machines utilisant la même application même si l'application de la machine hôte de WIFI fonctionne en arrière plan, et lister ces informations dans l'application utilisé par celui qui veut se connecter à ce réseau WIFI. Ces informations incluent le débit du réseau WIFI, la force et la puissance du signal ainsi que le niveau de la batterie de la machine hôte du WIFI. Cette fonctionnalité donnera à l'utilisateur une idée de la qualité du réseau WIFI disponible, lui permettant ainsi de choisir le réseau le plus performant auquel se connecter.

1.2 Description des besoins non fonctionnels

Notre application doit être capable d'enregistrer les mots de passe des réseaux WIFI utilisés afin de faciliter la connexion de l'utilisateur à ces réseaux ultérieurement, sans avoir à saisir manuellement les informations de connexion à chaque fois. Il est également important de distinguer clairement les points d'accès générés par l'application de ceux qui sont réels, pour garantir la sécurité des utilisateurs.

1.3 Étude de faisabilité technique

Notre projet consiste à développer une application mobile permettant de générer des réseaux Adhoc de type SPAN en se basant sur les fonctionnalités déjà intégrées dans les smartphones Android. Pour répondre à ce besoin, nous avons identifié plusieurs exigences techniques. Tout d'abord, nous devons concevoir une interface mobile simple et conviviale qui permettra à l'utilisateur de lister les réseaux WIFI disponibles, en faisant la distinction entre ceux qui sont lancés par

l'application à partir d'une autre machine et ceux qui sont lancés par un point d'accès réel avec un accès direct à Internet.

Ensuite, notre application devra permettre à l'utilisateur d'afficher les informations relatives à n'importe quel réseau WIFI, de stocker le mot de passe associé et de se connecter à ce réseau. Nous devons également mettre en place la fonctionnalité permettant à l'utilisateur de lancer un réseau WIFI et de démarrer ou arrêter un serveur pour partager les informations de qualité de service avec d'autres smartphones utilisant la même application. Ce serveur devra fonctionner en arrière-plan et continuer à fonctionner même si l'utilisateur quitte l'application. Enfin, nous devons inclure un système de notification pour informer l'utilisateur que le serveur est en cours d'exécution.

Pour répondre à ces exigences, nous utiliserons les fonctionnalités intégrées dans Android, notamment la gestion des réseaux WIFI, les services en arrière-plan et les notifications système.

Le réseau SPAN généré par l'application est représenté dans la figure suivante

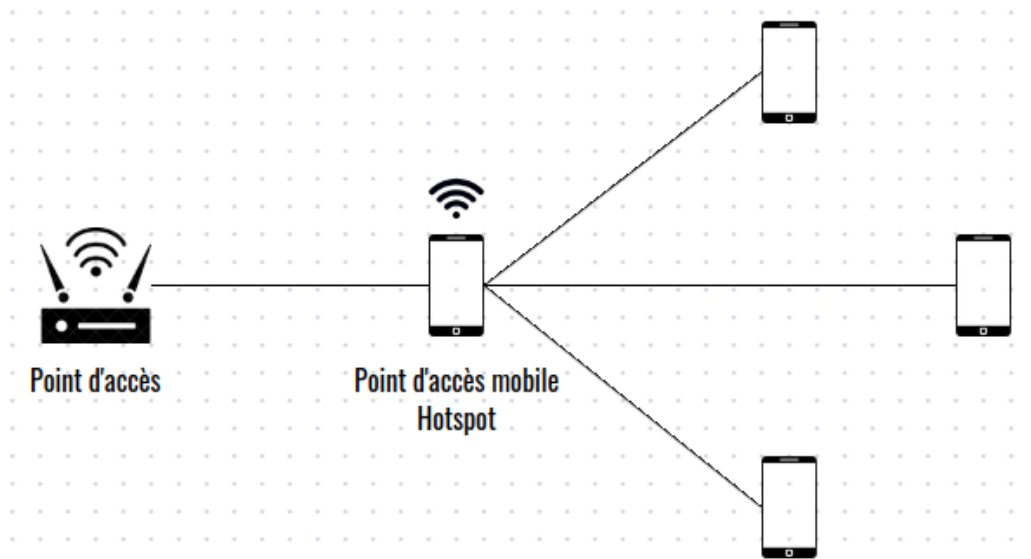


FIGURE 1 – Réseau SPAN généré par l'application

2 Conception de l'application

2.1 Description des fonctionnalités de l'application

L'application que nous sommes censés la développer, doit respecter le diagramme de cas d'utilisation suivant.

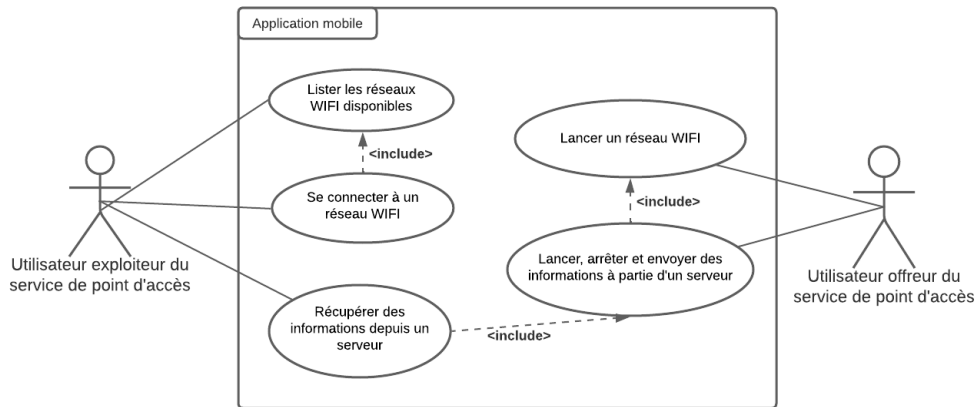


FIGURE 2 – Diagramme de cas d'utilisation de notre application

On distingue deux types d'utilisateurs de l'application : ceux qui souhaitent se connecter à un réseau WiFi et ceux qui souhaitent lancer un réseau WiFi. Les premiers doivent être en mesure d'afficher la liste des réseaux WiFi disponibles, qu'ils proviennent d'un point d'accès réel ou qu'ils soient lancés par l'application à partir d'un autre smartphone. Ils doivent également pouvoir sélectionner un réseau dans cette liste et afficher ses informations, ainsi que se connecter à ce réseau. Les seconds, quant à eux, peuvent créer un réseau WIFI, lancer et arrêter un serveur qui fonctionne en arrière-plan. Ce serveur est utilisé pour transférer les informations de chaque réseau entre la machine hôte du réseau et la machine souhaitant s'y connecter.

2.2 L'architecture de l'application

En ce qui concerne l'architecture de l'application, nous avons envisagé d'utiliser une architecture simple, telle que présentée dans la figure ci-dessous.

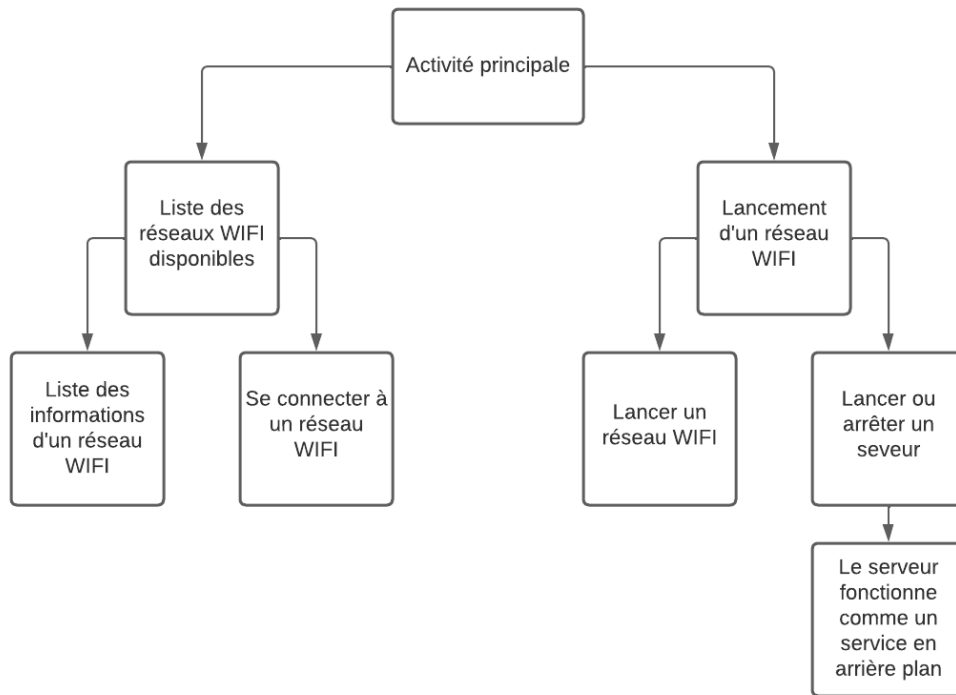


FIGURE 3 – Architecture de l'application

Dans la suite de ce rapport on va voir chaque fonctionnalité, comment on a fait à l'implémenter et le résultat de cette implémentation.

3 Implémentation de l'application

Notre application permet de générer et d'adhérer à un réseau Adhoc de type SPAN. Par conséquent, l'application doit inclure les fonctionnalités suivantes :

3.1 Lister les réseaux WIFI disponibles

Pour des raisons d'organisation nous avons créer une classe nommée **reseau-WIFI**, chaque objet de cette classe présente un WIFI disponible. Chaque objet de type `reseauWIFI` possède des attributes nommée SSID, BSSID, Debit, Force, Puissance, niveauBatterie, necessitPassword et APReel, ces attributs correspond respectivement au nom du réseau, l'adresse MAC du point d'accès, débit du réseau, Force du signal, puissance du signal, niveau de batterie de la machine hôte

du réseau dans le cas d'un réseau lancé par l'application dans un autre smart-phone, un booléen qui permet de spécifier si un réseau est sécurisé ou ouvert et finalement un booléen qui spécifie si un réseau est issu d'un vrai point d'accès ou lancé par l'application.

```
public class reseauWifi {  
    protected String SSID, BSSID;  
    protected float Debit, Force, Puissance;  
    protected int niveauBatterie;  
    protected boolean necessitePassword, APReel;
```

FIGURE 4 – Attributs de la classe reseauWIFI

Cette classe inclut des fonctions pour récupérer et modifier chaque attribut, ainsi que deux fonctions supplémentaires, une pour se connecter à un réseau WIFI et l'autre pour récupérer les données échangées entre deux machines à l'aide des sockets.

Pour lister les réseaux WIFI disponibles nous avons utilisé la classe **WifiManager**, qui permet de démarrer une recherche des réseaux WIFI disponibles et retourner le résultat sous forme d'une liste d'objets de type **ScanResult**, qu'on va utiliser pour récupérer le nom et l'adresse MAC du réseau WIFI, en plus de cela nous avons utilisé un objet de type **BroadcastReceiver** qui va notifier l'application une fois le scan est terminé pour qu'on puisse afficher la liste des réseaux WIFI disponibles.

Pour faire la distinction entre les réseaux WIFI issus d'un vrai point d'accès et ceux lancés par l'application, nous avons ajouté la contrainte que le nom des réseaux WIFI lancés par l'application **doivent commencer par "_"**, en utilisant le fait que c'est rare de trouver un réseau WIFI dont le nom commence par "_", et donc l'application va fonctionner sans aucun problème.

```

for (ScanResult wifi : resultat) {
    if (!Lancee.contains(wifi.SSID) && !Reel.contains(wifi.SSID)) { //pour éviter le fait d'écrire le même nom de WIFI plusieurs fois
        String ssid = wifi.SSID;
        String capabilities = wifi.capabilities;
        if (ssid.charAt(0) == '_'){
            Lancee.add(ssid);
            WifisLancee.add(new reseauWifi(wifi.SSID,wifi.BSSID, capabilities.contains("WPA")||capabilities.contains("WEP"), false));
        }
        else{
            Reel.add(ssid);
            WifisReels.add(new reseauWifi(wifi.SSID,wifi.BSSID, capabilities.contains("WPA")||capabilities.contains("WEP"), true));
        }
    }
}
}

```

FIGURE 5 – Distinction entre les deux types de WIFI

Pour récupérer la liste des réseaux WIFI disponibles, notre application doit avoir la permission d'accéder à la localisation. pour garantir cela on va vérifier si la permission n'est pas garantie on va proposer à l'utilisateur de donner à l'application cette permission.

```

public void checkLocationPermission(Context context){
    if (ContextCompat.checkSelfPermission(context, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED){
        requestPermissions(new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
    }
}

```

FIGURE 6 – Proposer à l'utilisateur de donner la permission de localisation

Le client va avoir un menu indiquant s'il voulait donner la permission d'accéder à la localisation pour cette application, comme vous voyez dans la figure suivante.

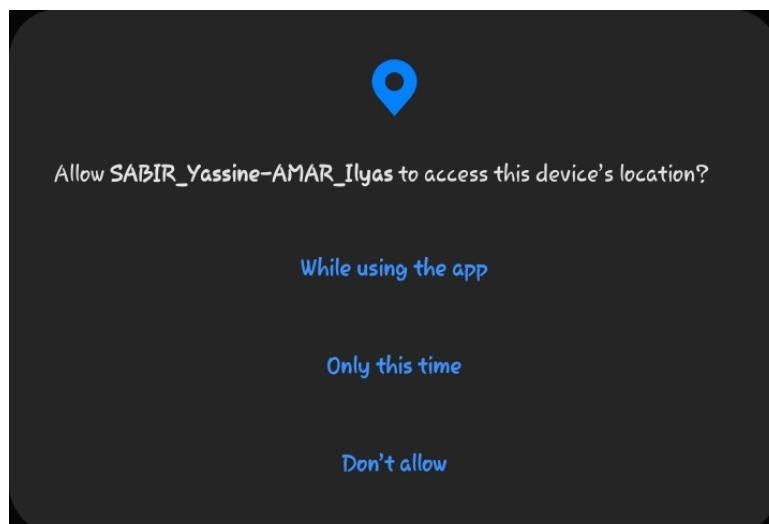


FIGURE 7 – Menu donnant à l'utilisateur la possibilité de donner l'accès à la localisation à l'application

Si le WIFI ou la localisation est désactiver, l'utilisateur aura une notification textuel indiquant qu'il doit les activer pour que le scan soit réalisé.

```
if (wifiMan.getWifiState() == WifiManager.WIFI_STATE_DISABLED) { //verifier si le wifi est desactivé
    Toast.makeText(ConnecteWIFI.this, "SVP activer le WIFI et actualiser la page", Toast.LENGTH_SHORT).show();
} else {
    if (!locationMan.isProviderEnabled(LocationManager.GPS_PROVIDER) && !locationMan.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
        Toast.makeText(ConnecteWIFI.this, "SVP activer la localisation et actualiser la page", Toast.LENGTH_SHORT).show();
    } else {
        rechercheWifi(ConnecteWIFI.this);
    }
}
}
```

FIGURE 8 – Assurer que le WIFI et la localisation sont activés

La notification contextuel que les utilisateurs vont avoir si un des services est désactivé

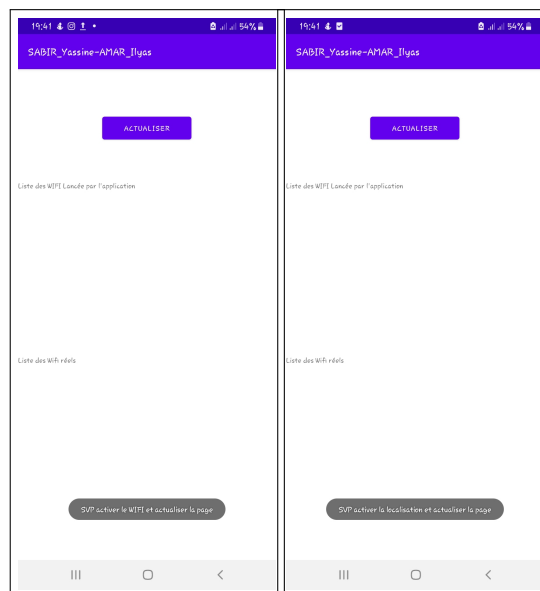


FIGURE 9 – Notification contextuel affiché

Si tout les services sont activés, l'utilisateur va avoir la liste de toutes les réseaux WIFI, disponibles séparés comme dans la figure suivante.

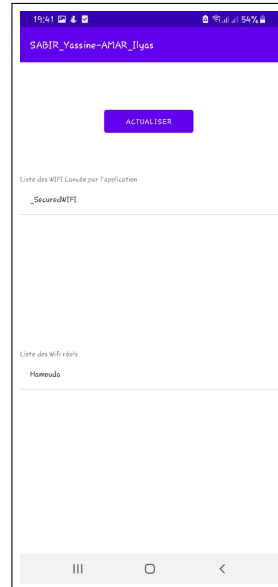


FIGURE 10 – Liste des réseau WIFI disponibles

3.2 Avoir les informations et se connecter à un réseau WIFI

Une fois l'utilisateur aura la liste des réseau WIFI disponibles, il va avoir la possibilité de choisir un réseau parmi cette liste, une fois il clique sur un des réseau on va afficher les informations de ce réseau, on va distinguer deux cas, les réseau WIFI issuent d'un vraie point d'accès, pour ce type de réseau on va afficher que le nom du réseau WIFI et l'adresse MAC du réseau, et pour l'autre type de réseau WIFI lancé par cette application dans un autre smartphone, va affiché les autres informations tels que le niveau de batterie, debit force et puissance du signal, comme indiquer dans la figure suivante.

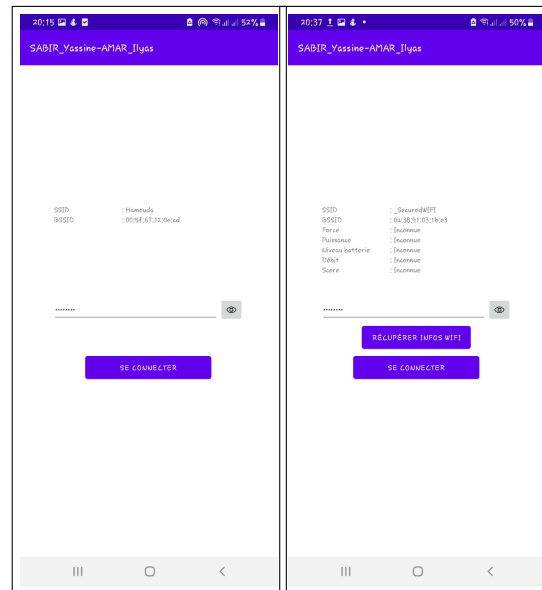


FIGURE 11 – Informations réseau

Le attributs de type booléen que nous avons inclues dans la classe `reseauWIFI` vont nous aider à spécifier les éléments à afficher dans l'activité des informations du réseau WIFI. En effet, le champ **necessitePassword** on va utiliser cela pour décider si le champ de mot de passe est affiché ou pas comme indiqué dans la figure ci-dessous.

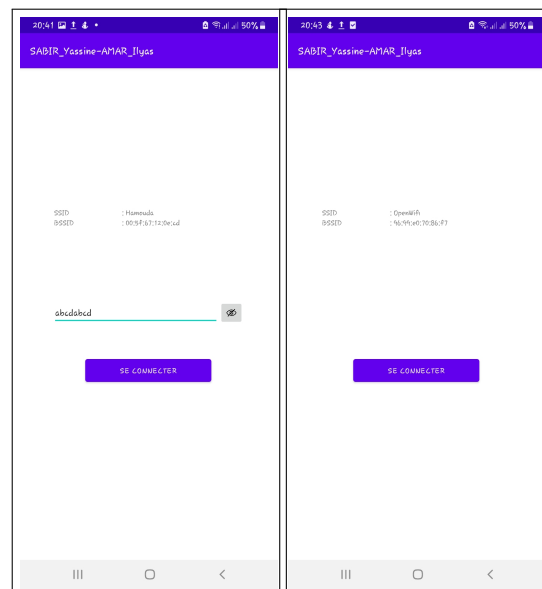


FIGURE 12 – Réseau sécurisé et réseau ouvert

L'attribut **APreel** va nous permettre de spécifier les informations à afficher,

comme montré dans la figure 11.

Une fois l'utilisateur aura les informations correspondantes à un réseau WIFI, il peut y connecter en insérant le mot de passe dans le cas des réseaux sécurisé, et en cliquant sur le bouton se connecter, ce bouton va nous permettre de se connecter à ce réseau en exécutant le code ci-dessous.

```
public static void connect2Wifi(Context context, String ssid, String bssid, boolean necessitePassword, String passWord){
    if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.Q) {
        NetworkSpecifier specifier;
        if (necessitePassword) {
            specifier = new WifiNetworkSpecifier.Builder()
                .setSsidPattern(new PatternMatcher(ssid, PatternMatcher.PATTERN_PREFIX))
                .setBssid(MacAddress.fromString(bssid))
                .setWpa2Passphrase(passWord)
                .build();
        }
        else{
            specifier = new WifiNetworkSpecifier.Builder()
                .setSsidPattern(new PatternMatcher(ssid, PatternMatcher.PATTERN_PREFIX))
                .setBssid(MacAddress.fromString(bssid))
                .build();
        }

        NetworkRequest request = new NetworkRequest.Builder()
            .addTransportType(NetworkCapabilities.TRANSPORT_WIFI)
            .setNetworkSpecifier(specifier)
            .build();

        ConnectivityManager connectivityMan = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
        ConnectivityManager.NetworkCallback networkCallback = new ConnectivityManager.NetworkCallback(){};
        connectivityMan.requestNetwork(request, networkCallback);
    }
}
```

FIGURE 13 – Se connecter à un réseau WIFI

Le code va se baser sur le nom et l'adresse MAC du réseau WIFI, pour établir la connexion WIFI, en fait le code va afficher un barre d'information, qui va demander à l'utilisateur s'il veut se connecter à ce réseau.

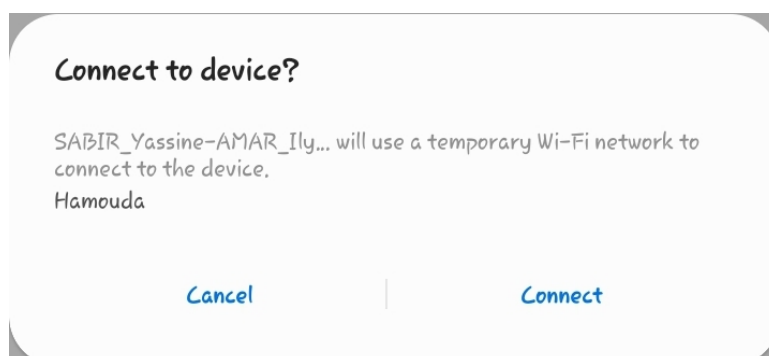


FIGURE 14 – Se connecter à un réseau

Pour améliorer les performances de l'application nous avons ajouter une fonc-

tionnalité qui permet de stocker les mots de passes pour les utilisations ultérieurs. En fait nous avons créé une base de données simple, qui contient deux colonnes, le nom de réseau et le mot de passe. comme indiqué dans la figure suivante.

```
public class DBWIFI extends SQLiteOpenHelper {

    private static final String NomBD = "WifiDB";
    private static final int Version = 1;
    private static final String ReqCreationBD = "Create table wifi("+
        "SSID TEXT primary key,"+
        "PASSWORD TEXT not null);";

    public DBWIFI(Context context) {
        super(context, NomBD, null, Version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(ReqCreationBD);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("drop table if exists wifi");
        onCreate(db);
    }
}
```

FIGURE 15 – Base de données des réseau WIFI

L'utilisateur s'il choisit un réseau WIFI parmi ceux disponibles, l'application va voir est ce que le mot de passe correspondant à ce réseau figure dans la base de donnée, si oui il va initialiser la valeur du mot de passe dans le champs réservé à cela. L'utilisateur en suite il peut insérer un autre mot de passe ou réserver le même ancien mot de passe. Si il s'agit de la première fois l'utilisateur volue se connecter à un réseau, il va saisir le mot de passe qui va être inséré dans la base de données, à l'aide de la fonction suivante.

```
public static void AjouterWifi(SQLiteDatabase db, String SSID, String password){
    ContentValues valeurs = new ContentValues();
    valeurs.put("ssid",SSID);
    valeurs.put("password",password);
    db.insert("wifi", null, valeurs);
}
```

FIGURE 16 – Enregistré le mot de passe d'un réseau à un réseau

Une notification contextuel indiquant que le mot de passe est enregistré va être affiché à l'écran.

Si l'utilisateur à déjà connecté a ce réseau, le mot de passe stocké dans la base de données va être affiché dans la case de mot de passe, ensuite s'il n'a pas le changer, on va se connecter directement avec ce mot de passe, sinon on doit mettre à jour ce mot de passe dans la base de données, à l'aide de la fonction suivante.

```
public static void update(SQLiteDatabase db, String SSID, String password){
    ContentValues valeurs = new ContentValues();
    valeurs.put("password",password);
    db.update("wifi", valeurs, " ssid = '"+SSID+"' ", null);
}
```

FIGURE 17 – Mise à jour d'un mot de passe

Une notification contextuel va être affiché à l'utilisateur, pour l'informer que le mot de passe est mis à jour. Tout cela on a fait à l'aide du code suivant.

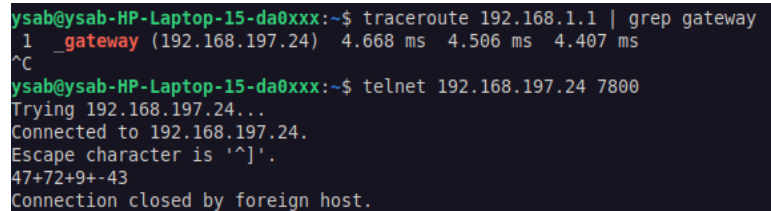
```
seConnecter.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (!necessitePassword){
            reseauWifi.connect2Wifi(getApplicationContext(), sSSID, sBSSID, necessitePassword, (String) null);
        }
        else{
            String password = MotDePasse.getText().toString();

            if (Mpass == null){
                Toast.makeText(InfoWifi.this, "Le mot de passe est enregistré", Toast.LENGTH_SHORT).show();
                DBWIFI.AjouterWifi(dbWrite, sSSID, password);
            }
            else{
                if (!password.equalsIgnoreCase(Mpass)){// l'utilisateur a changer la mot de passe stocké dans la base de données
                    DBWIFI.update(dbWrite, sSSID, password);
                    Toast.makeText(InfoWifi.this, "Le mot de passe est mis à jour", Toast.LENGTH_SHORT).show();
                }
            }
            reseauWifi.connect2Wifi(getApplicationContext(), sSSID, sBSSID, necessitePassword, password);
        }
    }
});
```

FIGURE 18 – Gestion des mots de passe

En plus de cela, dans le cas des réseau WIFI, lancé par l'application dans un autre smartphone, l'utilisateur pour accéder à les informations de la machine hôte du réseau, pour faire cela l'utilisateur doit d'abord **se connecter à ce réseau**, ensuite il va cliquer sur le bouton Récupérer info wifi, l'application va créer une

socket et va se connecter au serveur lancé dans la passerelle dans le port **7800**, il va récupérer une chaîne de caractères, qui contient des nombres séparés par des "+", comme indiqué dans la figure suivante.



```

ysab@ysab-HP-Laptop-15-da0xxx:~$ tracert 192.168.1.1 | grep gateway
 1  _gateway (192.168.197.24)  4.668 ms  4.506 ms  4.407 ms
^C
ysab@ysab-HP-Laptop-15-da0xxx:~$ telnet 192.168.197.24 7800
Trying 192.168.197.24...
Connected to 192.168.197.24.
Escape character is '^]'.
47+72+9+-43
Connection closed by foreign host.

```

FIGURE 19 – Récupération d'informations depuis un autre smartphone

La chaîne de caractères récupéré se compose de 4 chaîne de caractères séparés par +, la première chaîne de caractère correspond au niveau de batterie de la machine hôte du réseau, la deuxième indique le débit du réseau, la troisième indique la force du signal, et la dernière chaîne de caractère indique la puissance du signal. La fonction qui permet de récupérer ces données est indiqué dans la figure suivante.

```

public static String getData(Context context, String ssid, String bssid, boolean necessitePassword, String passWord){
    Executors.newSingleThreadExecutor().execute(new Runnable() {
        @Override
        public void run() {

        }
    });
    WifiManager wifiMan = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
    WifiInfo wifiInfo = wifiMan.getConnectionInfo();
    assert wifiInfo != null;
    while(wifiInfo.getNetworkId() == -1){//On va attendre jusqu'à la connexion au réseau wifi est effectué
    }
    String gateway = null;
    String batterie = null;
    final DhcpInfo[] dhcpInfo = {wifiMan.getDhcpInfo()};
    assert dhcpInfo[0] != null;
    int gatewayIp = dhcpInfo[0].gateway;
    gateway = String.format("%d.%d.%d.%d", (gatewayIp & 0xff), (gatewayIp >> 8 & 0xff), (gatewayIp >> 16 & 0xff), (gatewayIp >> 24 & 0xff));
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);
    try {
        Socket socket = new Socket(gateway,7800);
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        batterie = in.readLine();
        in.close();
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return batterie;
}

```

FIGURE 20 – Récupérer les informations d'une machine à l'aide des sockets

La visualisation de ces informations est dait à l'aide de ce code suivant

```

try{
    String[] Data = reseauWifi.getData(getApplicationContext(), sSSID, sBSSID, necessitePassword, currentpassword).split("\\+");
    niveauBatterie.setText(": " + Data[0]+"%");
    Debit.setText(": "+Data[1]);
    Force.setText(": "+Data[2]+" / 10");
    Puissance.setText(": "+Data[3]);
    Score.setText(": "+Float.toString(Score(Data[2], Data[3], Data[1], Data[0])));
}catch (Exception e){
    e.printStackTrace();
}

```

FIGURE 21 – Code de visualisation des information WIFI

Catch est utiliser ici pour éviter les problème si la fonction getData retourne un objet nulle, dans le cas si le serveur n'est pas lancé.

Voila un exemple de récupération d'information d'un réseau WIFI.

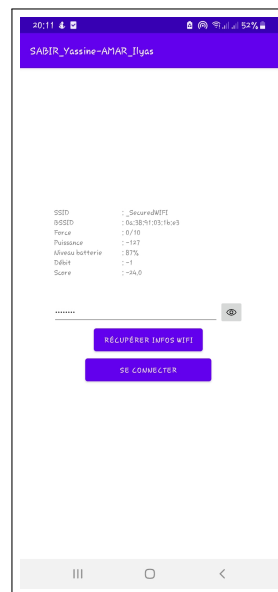


FIGURE 22 – Récupération d'informations d'un réseau

3.3 Lancement d'un réseau WIFI

Cette activité de l'application est désigné pour créer un hotspot ainsi que lancer un serveur ou bien l'arrêter.

Avant d'entamer l'explication de chaque méthode, nous devons tout d'abord expliquer la solution que nous avons juger la plus pratique pour échanger les données de **QoS**, tels que le niveau de batterie, la puissance de cette point d'accès et d'autre informations . entres *le point d'accès* et noter téléphone.

Solution Pour l'échange de données Qos :

Nous avons juger plus pratique d'opter pour l'architecture **Clinet/Server** pour envoyer des données nécessaires tels que le niveau du batterie du téléphone partageant la connexion au téléphone qui souhaite se connecté à cet appareil. Ou le téléphone jouant le rôle d'un point d'accès prend le rôle du Serveur, et accepte et envoie un message contenant le niveau du batterie, débit du réseau , force et puissance du signale de celui-ci au client qui est dans ce cas le téléphone désirant de se connecter, et cherche le meilleur répéteur pour lui.

Pour lancer un serveur nous avons utiliser la classe **ServerSocket**, en spécifiant le port de fonctionnement qu'on a choisis pour qu'il soit le port 7800, en plus nous avons récupérer les critères de qualité de service à l'aide des classes **BatteryManager**, **WifiManager** et **WifiInfo**, comme indiqué dans la figure suivante.

```
Toast.makeText(this, "Le serveur est lancé dans le port "+PORT, Toast.LENGTH_SHORT).show();
bm= (BatteryManager) getApplicationContext().getSystemService(Context.BATTERY_SERVICE);
wm = (WifiManager) getApplicationContext().getSystemService(Context.WIFI_SERVICE);
wi = wm.getConnectionInfo();
Thread_serveur = new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            socket_serveur = new ServerSocket(PORT);
            PrintWriter pw;
            int battery, debit, force, puissance;
            while (true) {
                Socket clientSocket = socket_serveur.accept();
                battery = bm.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY);
                debit = wi.getLinkSpeed();
                puissance = wi.getRssi();
                force = wm.calculateSignalLevel(puissance, 10);
                pw = new PrintWriter(clientSocket.getOutputStream(), true);
                pw.println(Integer.toString(battery) + "+" + Integer.toString(debit) + "+" + Integer.toString(force) + "+" + Integer.toString(puissance));
                pw.flush();
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});
```

FIGURE 23 – Serveur en écoute dans le port 7800

Nous avons choisis que le service capable de lancer le serveur fonctionne en arrière plan, c'est-à-dire même si je quitte l'application le serveur reste en fonctionnement, pour cela nous avons choisit de lancer le service en mode Foreground comme indiqué dans la figure suivante.

```
LancerServeur.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent serveurSocketIntent = new Intent(CreateWifi.this, Socket_Serveur.class);
        startForegroundService(serveurSocketIntent);
    }
});
```

FIGURE 24 – Service du serveur fonctionne en arrière plan

Tant que le service est en train d'exécuter en arrière plan, l'utilisateur va voir une notification indiquant cela, le code de cela est indiqué dans la figure suivante.

```
NotificationChannel channel = new NotificationChannel("Channel_ID", "Channel1", NotificationManager.IMPORTANCE_HIGH);
channel.setDescription("New channel");
NotificationManager nm = (NotificationManager) getApplicationContext().getSystemService(Context.NOTIFICATION_SERVICE);
nm.createNotificationChannel(channel);

Notification notification = new NotificationCompat.Builder(this, "Channel_ID")
    .setContentTitle("Serveur est lancée")
    .setContentText("Le serveur est en écoute dans le port "+PORT)
    .setSmallIcon(R.drawable.ic_notifications_black_24dp)
    .build();

startForeground(1, notification);
```

FIGURE 25 – Code de notification

Voilà la notification affichée.



FIGURE 26 – La notification

Pour arrêter le serveur l'utilisateur doit cliquer sur le bouton arrêter, ensuite le service va s'arrêter immédiatement.

```
ArreterServeur.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(CreateWifi.this, Socket_Serveur.class);
        stopService(intent);
    }
});
```

FIGURE 27 – Arrêt de serveur

Pour le lancement d'un point d'accès, nous avons trouver des difficultés à permettre l'application de lancer un réseau WIFI, en utilisant le ssid et mot de passe, la solution qu'on a utilisée, c'est que dès que l'utilisateur clique sur le bouton de lancer le mot de passe, une notification contextuel va affiché indiquant que l'utilisateur doit accéder au paramètres de point d'accès et lancer un avec la condition que son nom doit commencer par "_", puis il va se ridirigé vers les paramètres de WIFI.

```
LancerAP.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        Toast.makeText(CreateWifi.this, "Lancer un Hotspot avec un nom qui commence par _", Toast.LENGTH_SHORT).show();  
        Intent intent = new Intent(Settings.ACTION_WIFI_SETTINGS);  
        startActivity(intent);  
    }  
});
```

FIGURE 28 – Lancement d'un point d'accès

Voila l'activité qui permet de lancer le WIFI et le serveur.

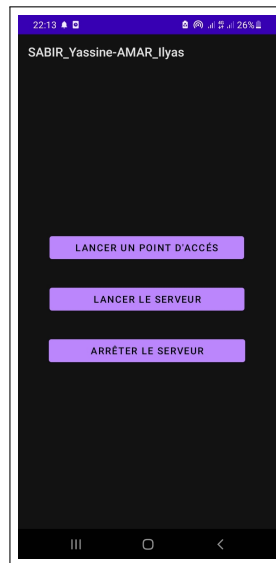


FIGURE 29 – Activité de lancement de WIFI et du serveur

Voici ce qu'on va avoir lorsque on clique sur le bouton lancer un point d'accès

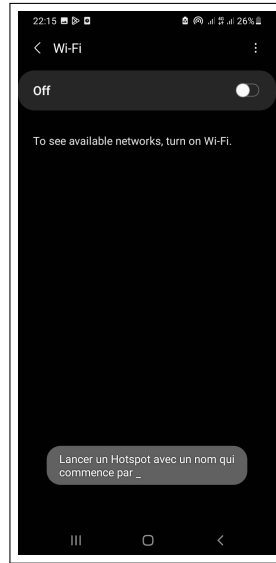


FIGURE 30 – Ridirection vers les paramètres de WIFI

Si on clique sur le bouton de lancement de serveur, on doit avoir ceci.

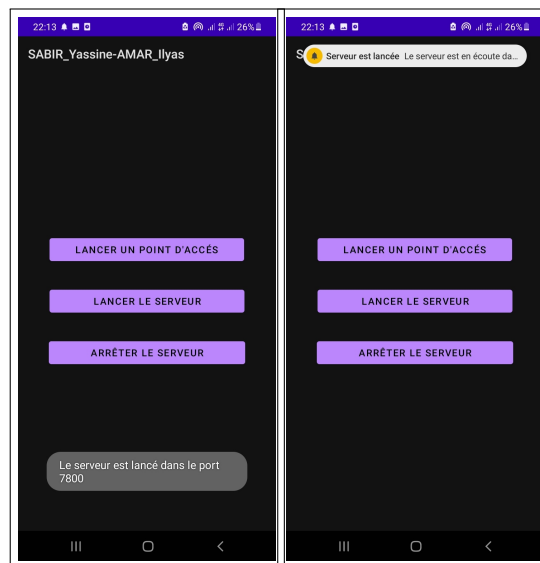


FIGURE 31 – Notification lors du lancement du serveur

Si on arrête le serveur, la notification qui indique qui est en train de fonctionnement va disparaître, et une notification contextuel va s'afficher.



FIGURE 32 – Arrêt du serveur

3.4 Les permissions données à l'application

Comme nous avons vu l'application doit être capable d'accéder à l'information réseau WIFI, les informations de localisation, les gestionnaires de notification, l'état de batterie, l'état de la connexion et d'internet et pouvoir les changer et la possibilité de lancer des services en arrière plan. Alors les permissions que nous avons données à notre application sont les suivantes :

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" tools:ignore="ExtraText"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS" tools:ignore="ProtectedPermissions"/>
<uses-permission android:name="android.permission.BATTERY_STATS" tools:ignore="ProtectedPermissions"/>
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.VIBRATE"/>
```

FIGURE 33 – Les permissions données à l'application

4 Conclusion

En conclusion, nous avons réussi à développer une application mobile permettant aux utilisateurs de se connecter à un réseau SPAN ou de créer un point d'accès, agissant comme un répéteur. Nous avons acquis une précieuse expérience en matière de réseautage et avons relevé le défi de trouver des solutions sécurisées pour choisir le point d'accès en fonction de la qualité de service (Qos), mais finalement nous avons opté pour la solution que nous avons jugé la plus pratique dans notre cas, celle de Client/Serveur.

Références

- [1] <https://developer.android.com/guide/topics/connectivity/wifi-permissions>
- [2] <https://developer.android.com/reference/android/Manifest.permission>
- [3] <https://developer.android.com/reference/android/net/wifi/WifiManager>
- [4] <https://developer.android.com/reference/android/net/wifi/WifiInfo>
- [5] <https://developer.android.com/reference/android/net/wifi/ScanResult>
- [6] <https://developer.android.com/reference/android/net/wifi/WifiConfiguration>
- [7] <https://developer.android.com/reference/android/os/BatteryManager>
- [8] <https://developer.android.com/reference/java/net/ServerSocket>
- [9] <https://developer.android.com/reference/java/net/Socket>
- [10] <https://www.ariase.com/box/dossiers/comment-mesurer-force-signal-wifi>
- [11] <https://stackoverflow.com/tags/android/info>