1 Projektverlauf

1.1 Konfigurations-App auf dem Raspberry-Pi

1.2 Hotspot-App auf dem Android Smartphone

Im Folgenden wird der komplette Projektverlauf der Teilaufgabe “Smartphone-App“ beschrieben. Als Programmiersprache wurde Kotlin gewählt. Die Entwicklungsumgebung ist Android Studio. Für die Codeverwaltung wird GitHub verwendet, für die Projektverwaltung verwenden wir ein KANBAN-Board auf eben dieser Seite.

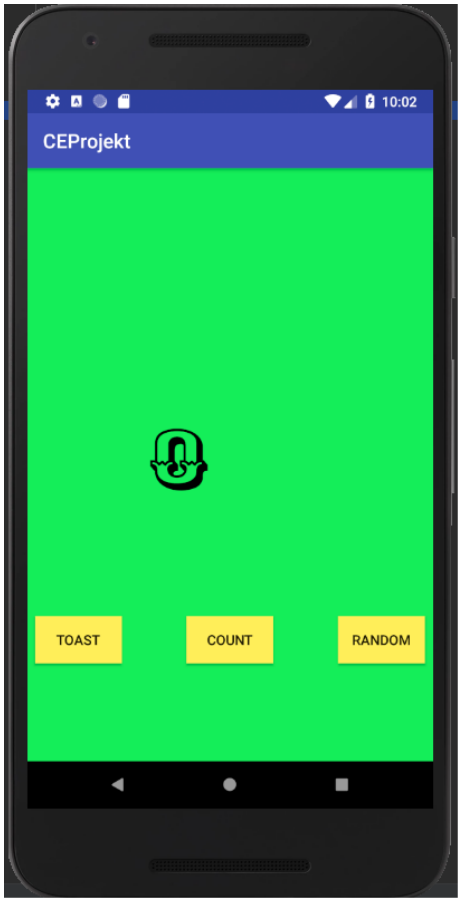

Abb. 1: Erstes Tutorial

1.2.1 Einrichtung der Entwicklungsumgebung

Das Ziel dieser Teilaufgabe ist eine funktionsfähige Entwicklungsumgebung einzurichten.

Zuerst muss Android Studio heruntergeladen und installiert werden. Danach wird das Projekt „CEProjekt“ erstellt. Bei der Erstellung des Projektes muss darauf geachtet werden, dass ein Häckchen bei „Kotlin-Support“ gesetzt ist. Zusätzlich wird hier die Android Version (Hier mindestens Android 5.0). und das Gerät für das entwickelt wird, spezifiziert. Nun wird noch ein virtuelles Gerät hinzugefügt, mit welchem man die Funktionalität testen kann, falls man kein Android Smartphone zur Hand hat.

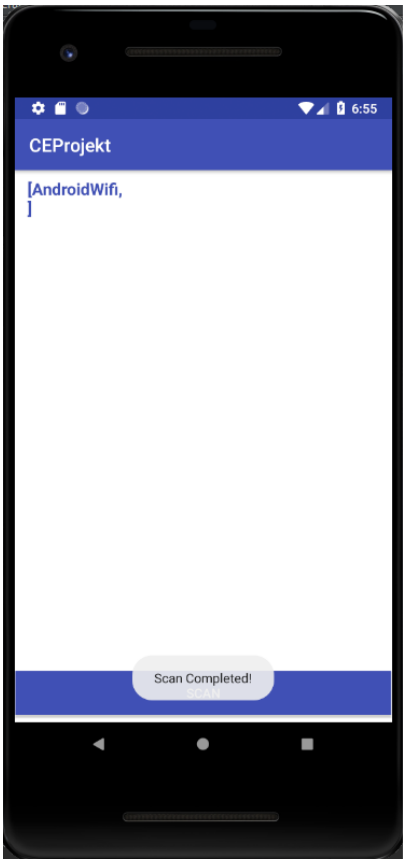

Abb. 2: Liste der WLAN-Netwerke

Um nun auch ein echtes Android-Gerät verwenden zu können, müssen zuerst die Entwickleroptionen des Gerätes sichtbar gemacht werden. Hierfür muss man in den Einstellungen die Geräteinformationen auswählen und mehrmals hintereinander auf die Buildnummer klicken. In diesen muss dann das „USB-Debugging“ angeschaltet werden. Wenn man nun das Smartphone am PC anschließt und versucht die Software auf dem Gerät zu installieren muss man am Gerät dieses noch erlauben. Danach wird die Software auf dem mobilen Endgerät installiert und ist dort verwendbar.

1.2.2 Erstellung einer ersten Applikation

Das Ziel dieser Teilaufgabe ist es ein erstes Tutorial (siehe https://codelabs.developers.google.com/codelabs/build-your-first-android-app-kotlin/index.html#0) zu programmieren, um zu verstehen wie die einzelnen Klassen und Dateien miteinander zusammenspielen und um einen ersten Einblick in die Programmiersprache Kotlin zu bekommen. Nach der Projekterstellung hat man folgende wichtige Klassen (nur ein Auszug), deren Zusammenspiel in dem Tutorial ein wenig klarer wird.

- String.xml
- Color.xml
- MainActiviy.xml (Grafisch und xml)
- MainActivity.kt
- AndroidManifest.xml

```
private fun startScanning(){
        Toast.makeText(this, "Starting Scan!",Toast.LENGTH_SHORT).show()
        //1. Start Scanning
        wifiManager.startScan()
        //2. Waiting for the Scan_Result_is_Available
        registerReceiver(broadcastReceiver,
        IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION))
        //wait 10 seconds
        Handler().postDelayed({stopScanning()}, 10000)
}
```

Code 1: startScanning Methode

Die Funktionalität steckt hinter den 3 verschiedenen Buttons. Mit dem Button Toast kann eine „Toast Meldung“ (siehe Abb. 2) ausgegeben werden. Mit dem zweiten Button „Count“ wird die Zahl in der Mitte des Bildschirms hochgezählt. Mit dem dritten Button „random“ wird eine Zufallszahl zwischen 0 und der aktuellen Nummer auf dem Bildschirm ermittelt und auf einem zweiten Screen dargestellt (siehe Abbildung ??).

1.2.3 Anzeigen aller verfügbaren WLAN-Verbindungen

Nun wird der Code der vorherigen Anwendung gelöscht und mit der eigentlichen Aufgabe begonnen. Das Ziel dieser Teilaufgabe ist es, dass man eine Liste aller momentan verfügbaren WLAN-Netzwerke bekommt, wenn man auf den Button „SCAN“ klickt (siehe Abb. 2).

```
resultList = wifiManager.scanResults as ArrayList<ScanResult>
```

Code 2: Speichern des Scans

Um das zu erreichen muss zuerst einmal ein wifiManager erstellt werden. Die „startScan“ Methode (siehe Code 1) führt dann einen Scan aus und speichert die Ergebnisse in einer Liste. Dann wird ein BroadcastReciever auf eine Variable gelegt. In diesem werden die Ergebnisse des Scans in der ArrayList „resultList“ gespeichert (siehe Code 2). Wenn der Scan nun fertig ist, dann geht es nach

einer kurzen Wartezeit in die „stopScanning“ Methode (siehe Code 3). In dieser wird nun der Reciever wieder entfernt und die Liste in einzelne Strings umgewandelt und ausgegeben.

```
private fun stopScanning(){
        unregisterReceiver(broadcastReceiver)
        d("Test: Unregister", "0")
        val stringList = ArrayList<String>()
        //Add the result names to a List
        for(result in resultList){
                stringList.add(result.SSID)
                stringList.add("\n")
                d("Test", result.SSID)
        }
        Toast.makeText(this, "Scan Completed!",Toast.LENGTH_SHORT).show()
        wifiList.text=stringList.toString()
}
```

Code 3: stopScanning Methode

An dieser Stelle ist ein Problem aufgetreten nämlich, dass die Liste stets leer war. Da zu diesem Zeitpunkt kein Android-Endgerät zur Verfügung stand wurde dies mit dem Emulator getestet. Dieser hat ein simuliertes WLAN und es war nicht ganz klar, ob der Emulator sein simuliertes WLAN erkennt. Als dies jedoch mit einem echten Gerät getestet wurde, bestand das Problem weiterhin.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Code 4: statische Berechtigung

Das eigentliche Problem ist, dass ab Android 6.0 einige Berechtigungen sehr kritisch sind (hier ACCESS_FINE_LOCATION), und diese zur Laufzeit gesetzt werden müssen. Ich habe diese jedoch bereits zur Compilezeit in AndroidManifest.xml gesetzt (siehe Code 4).

```
// Check if the permission has been granted
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        // Permission is already available, start scanning
        Toast.makeText(this, "Permission was granted",Toast.LENGTH_LONG).show()
        startScanning()
}
else {
        requestPermission()
}
```

Code 5: dynamische Berechtigung

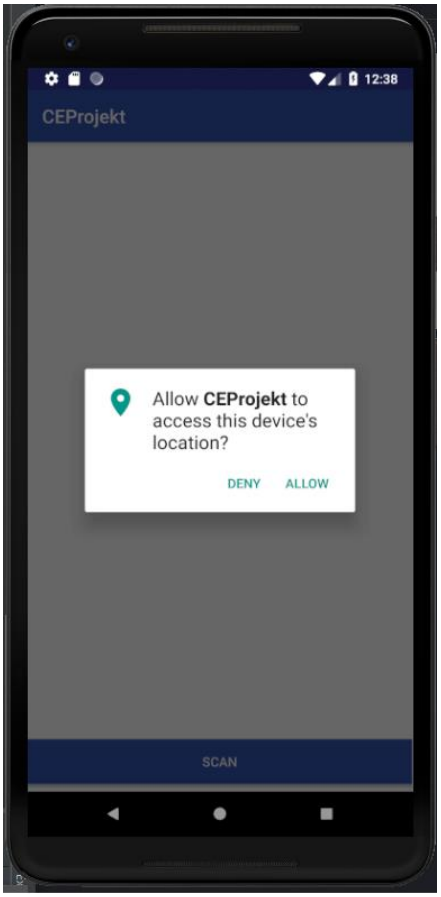

Abb. 3: Berechtigung zur Laufzeit

Die Lösung des Problems ist eine Abfrage und das setzten der Berechtigung zur Laufzeit (siehe Code 5). Wenn man nun die Anwendung startet und auf den Button klickt und zur Überprüfung kommt, dann wird ein Fenster angezeigt, welches man bestätigen muss (siehe Abb. 3). Nun werden auch alle verfügbaren WLAN-Netzwerke angezeigt. Dies funktioniert auch widererwartend auch mit dem AndroidStudio-Emulator.

```
const val WIFI_NAME ="Ax494"
const val WIFI_PASSWORD="Iam1H0tSp0t!?"
```

Code 6: Name und Passwort des gewünschten WLAN (iPhone)

1.2.4 Verbindung mit einem iPhone-Hotspot

Das Ziel dieser Teilaufgabe ist eine automatische Verbindung mit dem Hotspot eines bestimmten iPhones herzustellen, wenn man auf einen Button klickt.

Die Liste der vorherigen Teilaufgabe wird nicht mehr angezeigt, stattdessen wird ein Button „Connect To iPhone“ angezeigt. Nach einem Klick darauf und einer Wartezeit von einigen Sekunden bekommen wir eine Meldung, ob dies erfolgreich war oder nicht. (siehe Abbildung 4)

Da man sich immer mit dem gleichen Gerät verbinden soll, werden zuerst globale Variable für den Netzwerk-Namen und das Passwort initialisiert (siehe Code 6).

```
for(result in resultList){
        if(result.SSID== WIFI_NAME){
                hotspotToConnectTo=result
                hotspotIsAvailable=true
        }
}
```

Code 7: Speichern des gewünschten Hotspots

Es wird nun am Ende der stopScanning-Methode eine neue Funktion „checkForHotspot“ aufgerufen. In dieser suchen wir nach dem gewünschten Namen und speichern diesen in der globalen Variable „hotspotToConnectTo“ (siehe Code 7).

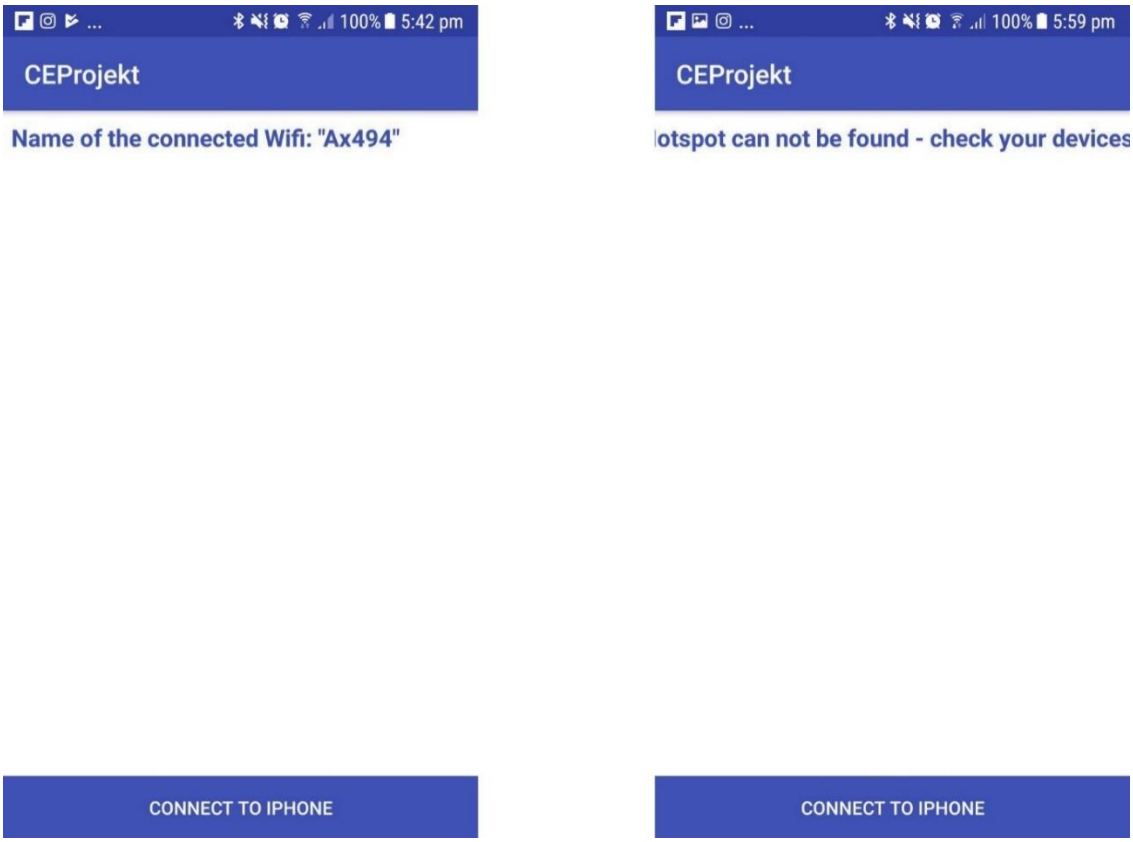

Abb. 4: Verbindung mit dem Hotspot des iPhones

Falls der Hotspot nicht gefunden wird dann überprüfe, ob dieser schon in der WLAN-Liste des Android-Gerätes zu finden ist, falls dies nicht der Fall ist füge ihn hinzu und starte einen erneuten Scan. Wenn dieser bereits konfiguriert wurde, dann Zeige eine Fehlermeldung (siehe Code 8).

```
//check if the hotspot is already configured
for(item in wifiManager.configuredNetworks) {
	if(item.SSID.substring(1,item.SSID.length-1)==(WIFI_NAME)) {
		isConfigured=true
	}
}
//if it is not configured, configure it and scan again
if(!isConfigured){
	val conf = WifiConfiguration()
	conf.SSID = WIFI_NAME
	conf.preSharedKey = WIFI_PASSWORD
	wifiManager.addNetwork(conf)

	Toast.makeText(this, "Hotspot not found - network will be configured",
	Toast.LENGTH_SHORT).show()
	actionInProgress=false //able to start a new scan
	startScanning()
}
else {
	//if it is already configured display a check your device message
	displayedText.append("Hotspot can not be found - check your devices")
	actionInProgress=false
}
```

Code 8: Prüfung der Verfügbarkeit des Hotspots

Falls der Hotspot gefunden wurde, wird die Funktion „connectToHotspot“ aufgerufen. In dieser wird die momentane WLAN-Verbindung abgebrochen, das gewünschte WLAN in der Liste der konfigurierten WLAN-Netzwerke herausgesucht und das Android-Gerät verbindet sich damit. Zusätzlich wird noch er angezeigt, mit welchem Netzwerk man sich verbunden hat (siehe Code 9).

```
private fun connectToHotspot(){
        //disconnect the active wifi connection
        wifiManager.disconnect()
        val stringList = ArrayList<String>()
        var networkId =0
        //search all configured wifi connections
        for(item in wifiManager.configuredNetworks) {
        /*if the netwpork is already configured, find the id and display the desired
        Information*/
                if(item.SSID.substring(1,item.SSID.length-1)==(WIFI_NAME)) {
                        networkId = item.networkId
                        displayedText.append("    Name of the connected Wifi: ")
                        displayedText.append(item.SSID)
                        Toast.makeText(this, "network is already configured",
                        Toast.LENGTH_SHORT).show()
                }
        }
        //connect to the wifi
        wifiManager.enableNetwork(networkId,true)
        wifiManager.reconnect()
        Toast.makeText(this, "Connected to iPhone hotspot",Toast.LENGTH_LONG).show()
        actionInProgress=false //able to start a new scan
        }
```

Code 9: connectToHotspot Funktion

1.2.5 Verbindung mit dem Raspberry Pi Hotspot

Das Ziel dieser Teilaufgabe ist eine automatische Verbindung mit dem Hotspot des Raspberry Pi herzustellen, wenn man auf einen Button klickt.

Um sich nun mit dem Raspberry Pi verbinden zu können, wird ein zweiter Button eingefügt. Jetzt wird bei einem Klick auf einen der beiden Button die globalen Variablen „WIFI_NAME“ und „WIFI_PASSWORD“ angepasst (siehe Code 10).. Es werden zusätzlich noch einige Änderungen zwecks „Toast Mitteilungen“ gemacht.

```
fun buttonOneClicked(view: View){
        WIFI_NAME ="Ax494"
        WIFI_PASSWORD="Iam1H0tSp0t!?"
        buttonClicked()
}

fun buttonTwoClicked(view: View){
        WIFI_NAME ="Rpi"
        WIFI_PASSWORD="123456789!?"
        buttonClicked()
}
```

Code 10: Hinzufügen der Pi Verbindung

1.2.6 Einrichten der Umgebung des Raspberry Pi

Das Ziel dieser Teilaufgabe ist die Einrichtung der Umgebung des Raspberry Pi, sodass auch die Kommunikation mit dem Pi getestet werden kann. Es wird eine Tastatur, eine Maus und ein Bildschirm an den Raspberry angeschlossen. Nun wird noch eine SD-Karte benötigt, auf welcher sich das Betriebssystem „Raspbian Jessie“ befindet. Nun wird der Code über git auf den Pi geladen und es kann weitergemacht werden.

```
val secondPart = Intent(this, WifiListActivity::class.java)
//using the result list from the smatphone scan
secondPart.putExtra("resultList", resultList)
startActivity(secondPart)
```

Code 11: Übergabe der Netwerk-Liste und starten der Zweiten Aktivität

1.2.7 Kleine Anpassung der Gestaltung

Das Ziel dieser Teilaufgabe ist eine kleine Anpassung der Benutzeroberfläche, sodass das Design ansprechender wirkt. Das Ergebnis ist in Abb. 5 zu finden. Zusätzlich wird noch ein Button eingefügt, um die Netzwerk-Liste des Raspberry Pi zu erhalten. Dieser ist nur sichtbar, wenn man mit dem Hotspot verbunden ist.

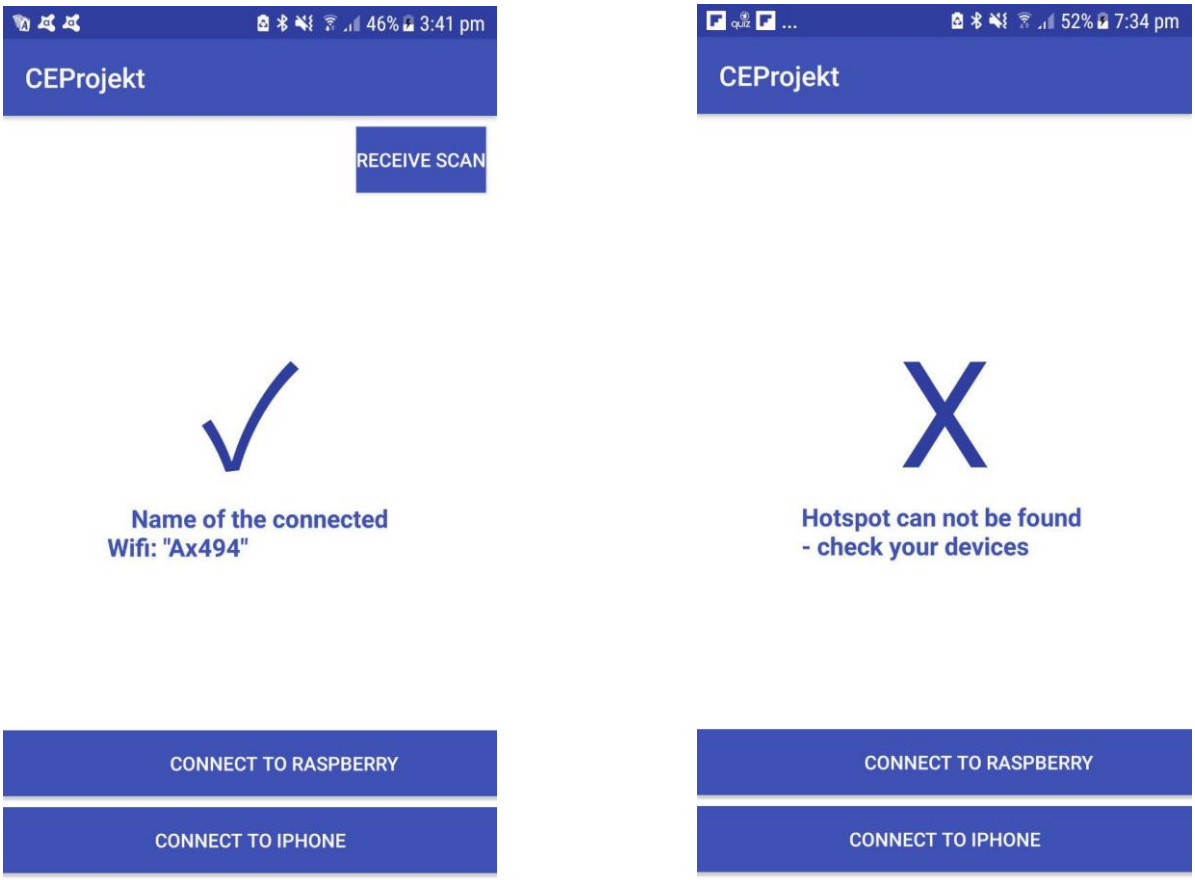

Abb. 5: Angepasste Oberfläche

1.2.8 Anzeigen der Netzwerk Liste des Smartphones

Das Ziel dieser Teilaufgabe ist eine Liste aller verfügbaren WLAN-Netzwerke des Smartphones anzuzeigen.Da der Scann des Raspberry Pi noch nicht implementiert ist, wird im Folgenden die Liste aus 2.2.3 verwendet. Hierzu wird ein zweiter Screen mit einer zweiten Aktivität namens „WifiListActivity“ erstellt. Zuerst muss man die Liste der Aktivität übergeben, um sie auch dort verwenden zu können (siehe Code 11). Die zweite Aktivität muss hingegen die übergebene Liste auch speichern (siehe Code 12) um diese anschließend ausgeben zu können. Dies alles geschieht, wenn man auf den Button „Recieve Scan“ (siehe Abb. 5) drückt. Anschließend kommt man auf einen zweiten Screen, auf welchem alle verfügbaren WLAN-Netwerke in der Umgebung aufgelistet sind (siehe Abb. 6).

```
fun getList(){
        resultList= intent.getSerializableExtra("resultList") as ArrayList<ScanResult>
        displayList()
}
```

Code 12: Speichern und Anzeigen der Netwerk-Liste in der zweiten Aktivität

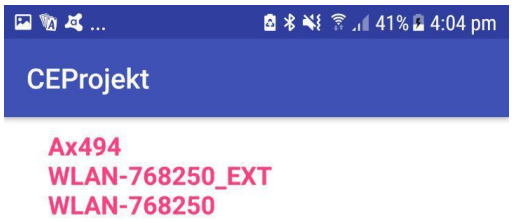

Abb. 6: Netzwerk-Liste des Smartphones

1.2.9 Auswählen einer Verbindung und Eingabe des Passwortes

Das Ziel dieser Teilaufgabe ist es eine Verbindung auszuwählen und anschließend das passende Passwort dazu einzugeben, um dieses später an den Raspberry Pi schicken zu können. Zuerst muss die Liste aus 2.2.8 überabreitet werden, sodass die einzelnen Einträge klickbar sind (siehe Code 13). Sobald nun auf ein Netzwerk geklickt wird, wird man auf einen weiteren Screen weitergeleitet. Auf diesem ist man in der Lage das zugehörige Passwort zu dem ausgewählten WLAN-Netzwerkes einzugeben (siehe Abb. 7).

```
fun displayList(){
        val thirdPart = Intent(this, WifiConfigurationActivity::class.java)
        thirdPart.putExtra("resultList",resultList)
        for(result in resultList){
        /*Creating a  clickable String - on click: start the third activity*/
                val ss = SpannableString(result.SSID)
                val clickableSpan = object : ClickableSpan(){
                        override fun onClick(widget: View?){
                                //create a new activity passing the name of the wifi
                                thirdPart.putExtra("wifiName", result.SSID)
                                startActivity(thirdPart)
                        }
                        override fun updateDrawState(ds: TextPaint?) {
                                super.updateDrawState(ds)
                                ds!!.isUnderlineText = false
                        }
                }
                ss.setSpan(clickableSpan,0,result.SSID.length,
                Spannable.SPAN_EXCLUSIVE_EXCLUSIVE)
                wifiList.movementMethod = LinkMovementMethod.getInstance()
                wifiList.highlightColor = Color.TRANSPARENT
                wifiList.append(ss)
                wifiList.append("\n")
        }
}
```

Code 13: Netzwerk-Liste mit klickbaren Einträgen

Nun hat man die Möglichkeit mit einem Klick auf den Button „Cancel“ wieder zum Anfang der Anwendung zu kommen und einen erneuten Versuch zu starten, sich mit dem gewünschten Hotspot zu verbinden. Durch einen Klick auf den Button „Back“ kommt man wieder zurück zur Netzwerk Liste, und kann nun ein anderes Netzwerk, mit dem sich der Pi dann verbinden soll, auswählen (siehe Code 14). Durch den Button „Send to Raspberry Pi“ soll es möglich sein, den Namen des gewünschten Netzwerkes und das zugehörige Passwort an den Pi zu senden (siehe Abschnitt 2.??).

```
fun buttonBackClicked(view: View){
        val secondPart = Intent(this, WifiListActivity::class.java)
        resultList= intent.getSerializableExtra("resultList") as
        java.util.ArrayList<ScanResult>
        secondPart.putExtra("resultList",resultList)
        startActivity(secondPart)
}
```

Code 14: Funktion hinter dem „Back" Button

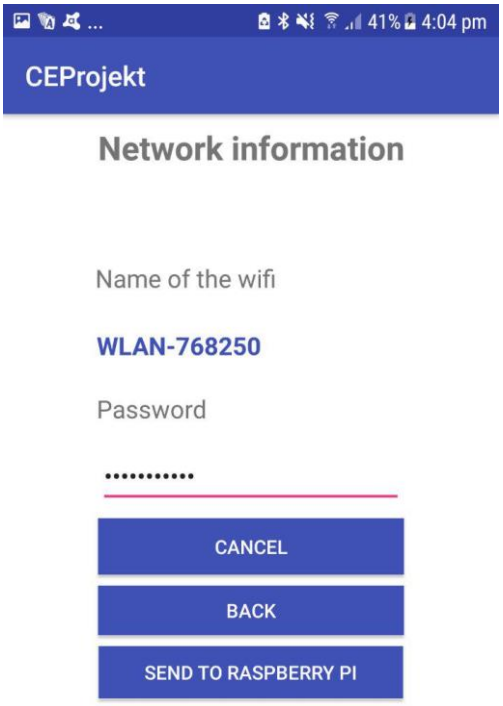

Abb. 7: Eingabe des Passwortes zu einer ausgewählten Verbindung