UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

TWDM

Anul IV, Tehnologia Informației

Autor: Borza Andrei Gabriel
Grupa: 30644

FACULTATEA DE AUTOMATICĂ
ȘI CALCULATOARE

12 Mai 2025

Cuprins

1 Introducere

În cadrul acestui laborator vom învăța cum să dezvoltăm o aplicație Android folosind Flutter, un framework modern și cross-platform pentru dezvoltarea aplicatiilor mobile. Aplicația va avea ca scop scanarea rețelelor WiFi disponibile, vizualizarea detaliilor fiecărei rețele, conectarea la o rețea prin parolă, și salvarea acestora pentru utilizare ulterioară.

La finalul acestui laborator, studentul va fi capabil să:

- înțeleagă structura de bază a unei aplicații Flutter
- utilizeze pachetele **wifi_scan** și **wifi_iot**;
- scaneze rețele WiFi și afișeze detalii relevante
- conecteze aplicația la o rețea WiFi
- gestioneze o listă de rețele salvate
- aplice concepte de stateless și stateful widgets
- managementul permisiunilor, accesul la internet
- configurarea unei aplicatii production-ready

2 Arhitectura tehnica

2.1 Mobile

Pentru dezvoltarea aplicatiei mobile am ales ca tehnologie Flutter, care:

- permite dezvoltarea rapidă și interactivă (Hot Reload)
- este cross-platform (același cod funcționează pe Android, iOS și web)
- are o comunitate activă și pachete disponibile pentru interacțiuni hardware, inclusiv rețele WiFi
- este potrivit pentru începători datorită documentației excelente și a simplității limbajului Dart
- true 60fps performance

2.2 Backend

Pentru modulul de backend am ales Spring Boot si Hibernate, impreuna cu PostgreSQL:

- **Spring Boot**: unul dintre cele mai populare si dezvoltate ecosisteme pentru backend
- **Hibernate**: ORM consacrat pentru Spring & Spring Boot
- **PostgreSQL**: database system modern, rapid, scalabil si cu multe functionalitati out of the box

2.3 Comunicare

Pentru comunicarea intre cele doua componente, am ales un model REST, prin care vom putea salva, sterge si accesa retelele scanate pentru procesari/statistici ulterioare, cat si informatii relevante despre acestea, cum ar fi:

- ssid
- bssid(mac)

- security
- signal strength
- caller ip

3 Mobile

3.1 Structura teoretica

O aplicatie flutter, de regula, are urmatoarele componente de baza:

- **folderul android**: aici se afla codul sursa Kotlin pentru conectarea sa cu modulul Flutter, si de unde gestionam permisiunile aplicatiei cat si putem scrie cod nativ, pentru functionalitati cum ar fi procentajul baterier.
- **folderul ios**: asemanatorul cu cel android, diferenta este ca aici codul sursa este Swift.
- mai avem 4 pachete cu functionalitati similare, acestea fiind **linux**, **macos**, **windows** si **web**.
- **folder tests**: aici vom scrie unit tests cat si integration tests, de obicei folosind Cucumber si Mockito.

In continuare, vom discuta de fisierele specifice Flutter, care sunt:

- **pubspec.yaml**: aici avem toate detaliile aplicatiei, sdk, dependinte cat si declararea asset-urilor, cum ar fi fonturi custom, fisiere de configurare etc.
- **lib/main.dart**: punctul de intrare al aplicatiei, de aici vom incepe development-ul.

In general, vom folosi conceptul de Widget, care se refera la componenta UI care se randeaza. Aceste widget-uri pot fi:

- **StatelessWidget**: nu isi schimba starea, deci poate fi randata ca si constanta si nu va participa la re-build
- **StatefulWidget**: componenta se modifica in functie de stare, are la dizpozitie o metoda de ***setState***

3.2 Crearea unei aplicatii Flutter

Inainte sa putem crea o aplicatie Flutter, trebuie sa ne asiguram ca avem instalat SDK-ul de Dart si Flutter (similar cu ceea ce inseamna JDK pentru Java). Apoi, recomandat ar fi instalarea Android Studio (dar se poate configura si IntelliJ), impreuna cu sdk-ul de Android si pregatirea unui emulator (sau conectarea unui dizpozitiv real).

Pentru a crea un proiect nou, putem rula in terminal: ***flutter create my_wifi_app*** (de retinut, standardul Dart este snake case). Varianta de preferat totusi este din Android Studio: Create new Flutter Project.

Dupa ce avem proiectul pornit, vom avea nevoie sa adaugam pachetele necesare lucrului cu WiFi si permisiuni (vom modifica fisierul *pubspec.yaml*, apoi vom rula *flutter pub get*):

```
dependencies:
  flutter:
    sdk: flutter
  wifi_scan: ^0.4.1+2
```

```
  wifi_iot: ^0.3.19+2
  permission_handler: ^11.0.0
  app_settings: ^5.0.0
  network_info_plus: ^6.1.4
  http: ^1.4.0
  flutter_dotenv: ^5.2.1
```

Urmatorul pas este sa ne asiguram ca avem permisiunile necesare pentru WiFi. Vom modifica fisierul *AndroidManifest.xml*, care se afla sub android/app/src/main:

```
<uses-permission
 ↪  android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission
 ↪  android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
 ↪  android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission
 ↪  android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
 ↪  android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- Optional: Needed for newer Androids to initiate
 ↪  connections -->
<uses-permission
 ↪  android:name="android.permission.NEARBY_WIFI_DEVICES"
 ↪  android:usesPermissionFlags="neverForLocation"/>
```

Inainte de a trece la partea de cod, trebuie sa mai configuram un singur lucru: un fisier **loca.env**, care sa contina doua intrari:

```
SERVER_HOST=<your-server-ip>
SERVER_PORT=<your-server-port>
```

Pentru structura aplicatiei, in general sub folderul lib, vom folosi urmatoarele conventii:

- **lib/config**: orice configurari de care avem nevoie
- **lib/data**: modele de date
- **lib/screens**: paginile pe care le vom folosi, cu terminatie in *_screen*
- **lib/widgets**: custom widgets
- **lib/util**: aici vom scrie utilitare si helpere pentru aplicatie, cum ar fi lucrul cu URL-uri etc.

Codul sursa se poate descarca de pe github: AndreiZ0R/wifi-mobile-app.

3.3 Interfata aplicatiei

- Ecran principal, in care putem scana pentru retelele din apropiere cat si filtra
- Ecran pentru detaliile retelei
- Ecran pentru managementul retelelor salvate

4 Backend

4.1 Prerequisites

Pentru partea de backend, vom avea nevoie de:

- jdk 21 (openjdk sau alternative)
- maven 3.9
- IntelliJ
- Docker

4.2 Arhitectura si configurare

Vom folosi o arhitectura layered, plecand de la modelul de date catre repository - service - rest controller.

Pentru a ne asigura ca totul ruleaza as expected, avem nevoie de urmatoarele configurari:

- fisier **local.env** sub folderul database
- modificarea fisierului properties pentru a expune serviciul
- adaugarea variabilelor de mediu la Running Configuration-ul serviciului

Atat in fisierul local.env cat si in running-confing, avem nevoie de doua intrari, pentru a ne conecta la baza de date:

- **DB_USERNAME**
- **DB_PASSWORD**

Pentru a porni DB-ul, vom rula sub folderul database comanda:

```
docker-compose --env-file=local.env up
```

care se va asigura ca totul porneste as expected.

Avand DB-ul up, trebuie sa modificam fisierul properties pentru a ne conecta (*application.yml*):

```
spring:
  application:
    name: wifi-be
  datasource:
    url: jdbc:postgresql://localhost:5435/wifi_db
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
        default_schema: public
    show-sql: true
```

```
server:
  port: 8080
  address: 0.0.0.0
```

Ultimul pas este sa rulam ***mvn clean install***, si apoi putem porni serviciul.

5 Rezultate

Figura 1: Scan retele WiFi

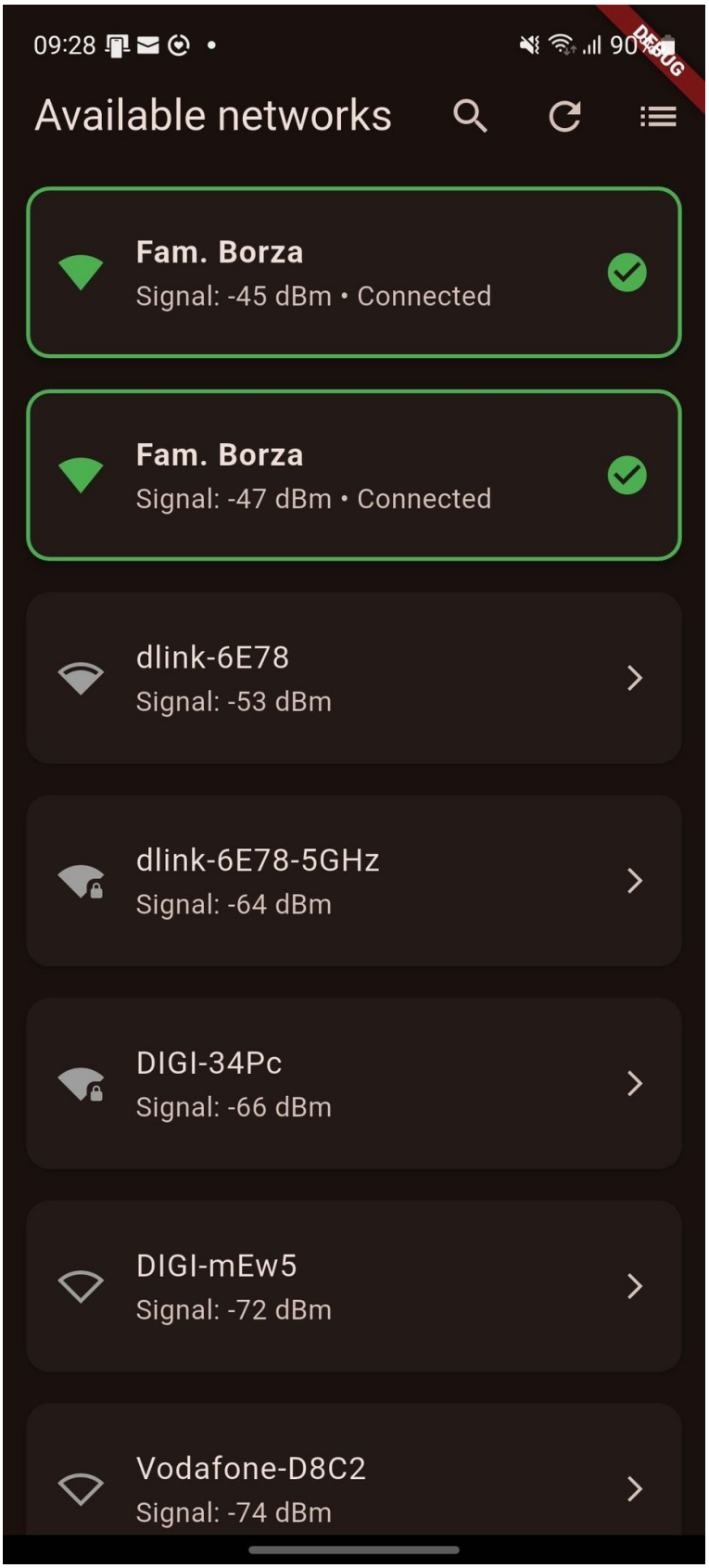

Figura 2: Retele disponibile

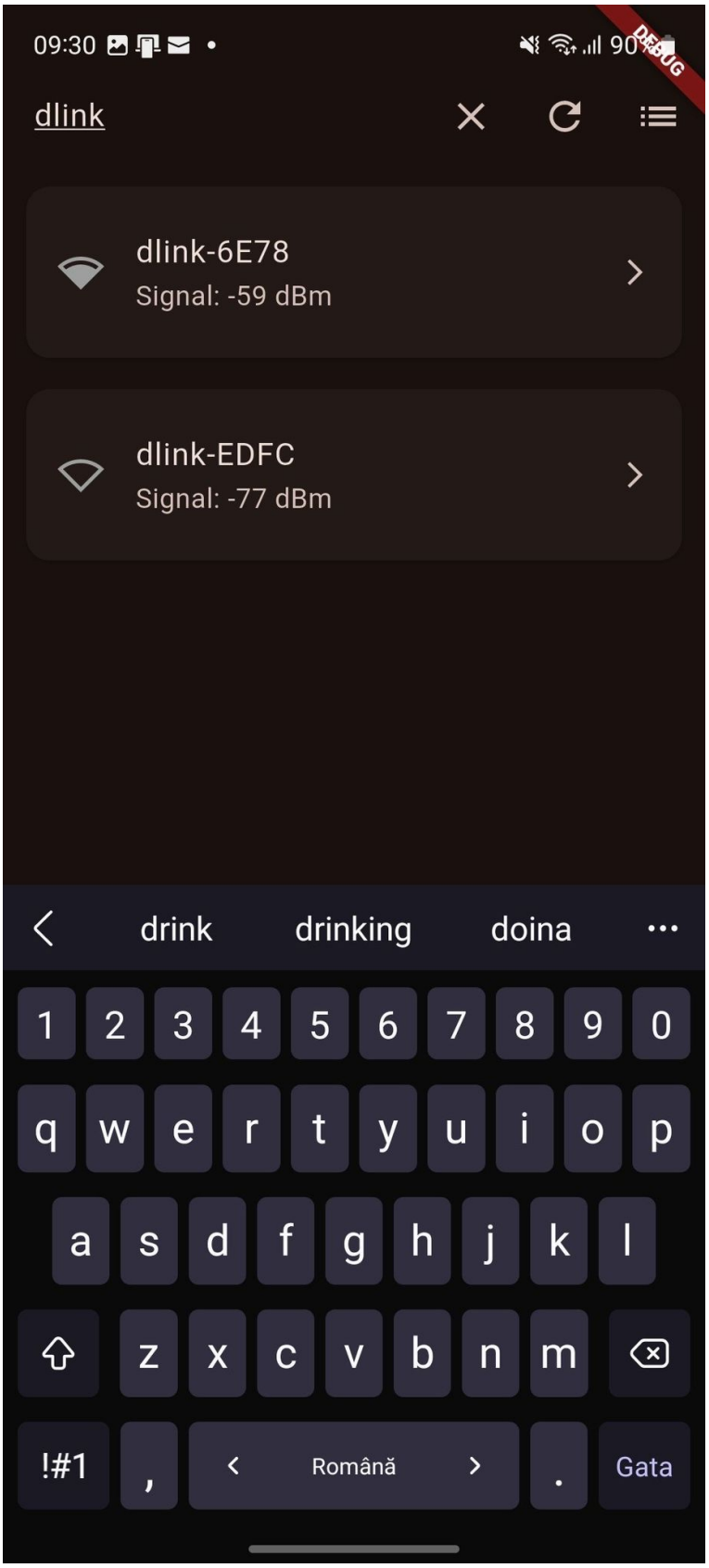

Figura 3: Filtrare

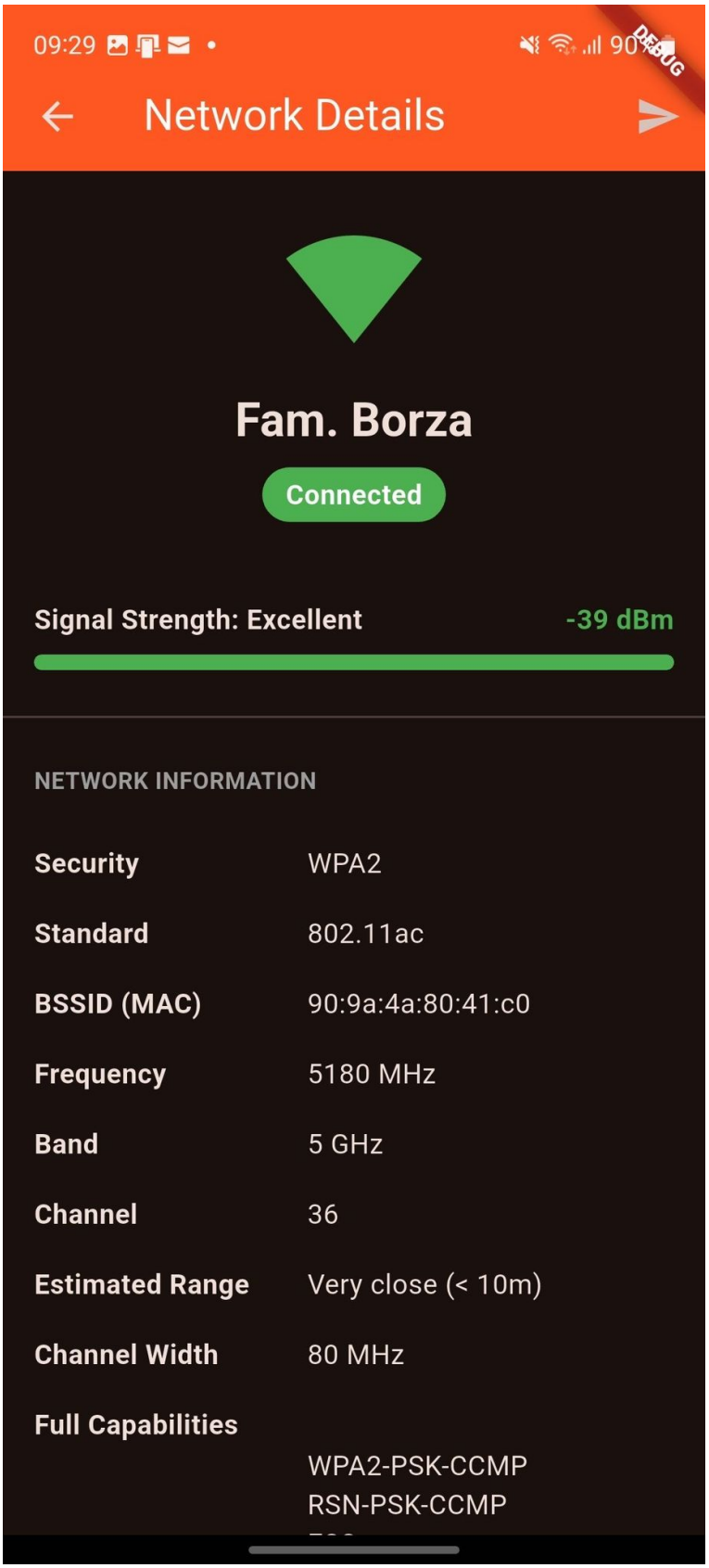

Figura 4: Detalii retea conectata - 1

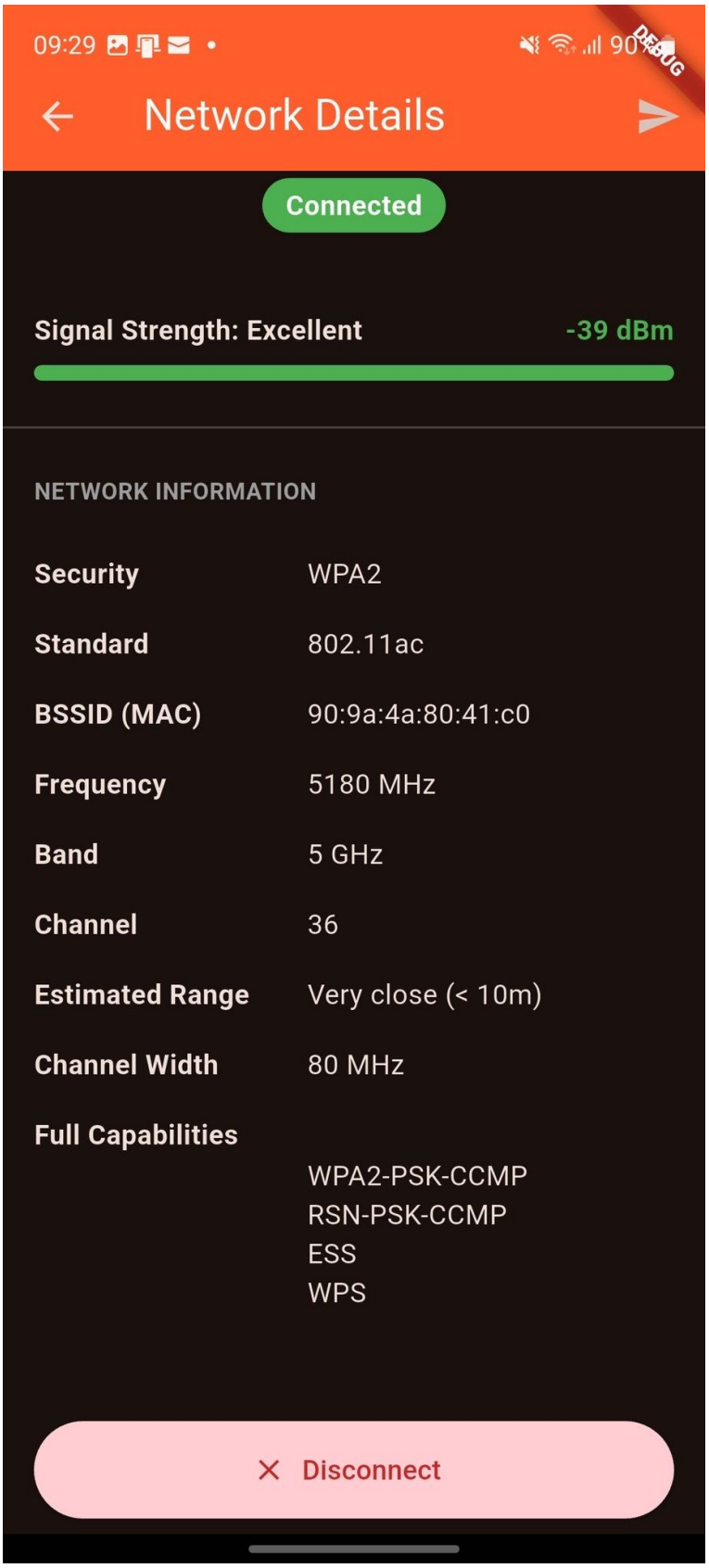

Figura 5: Detalii retea conectata - 2

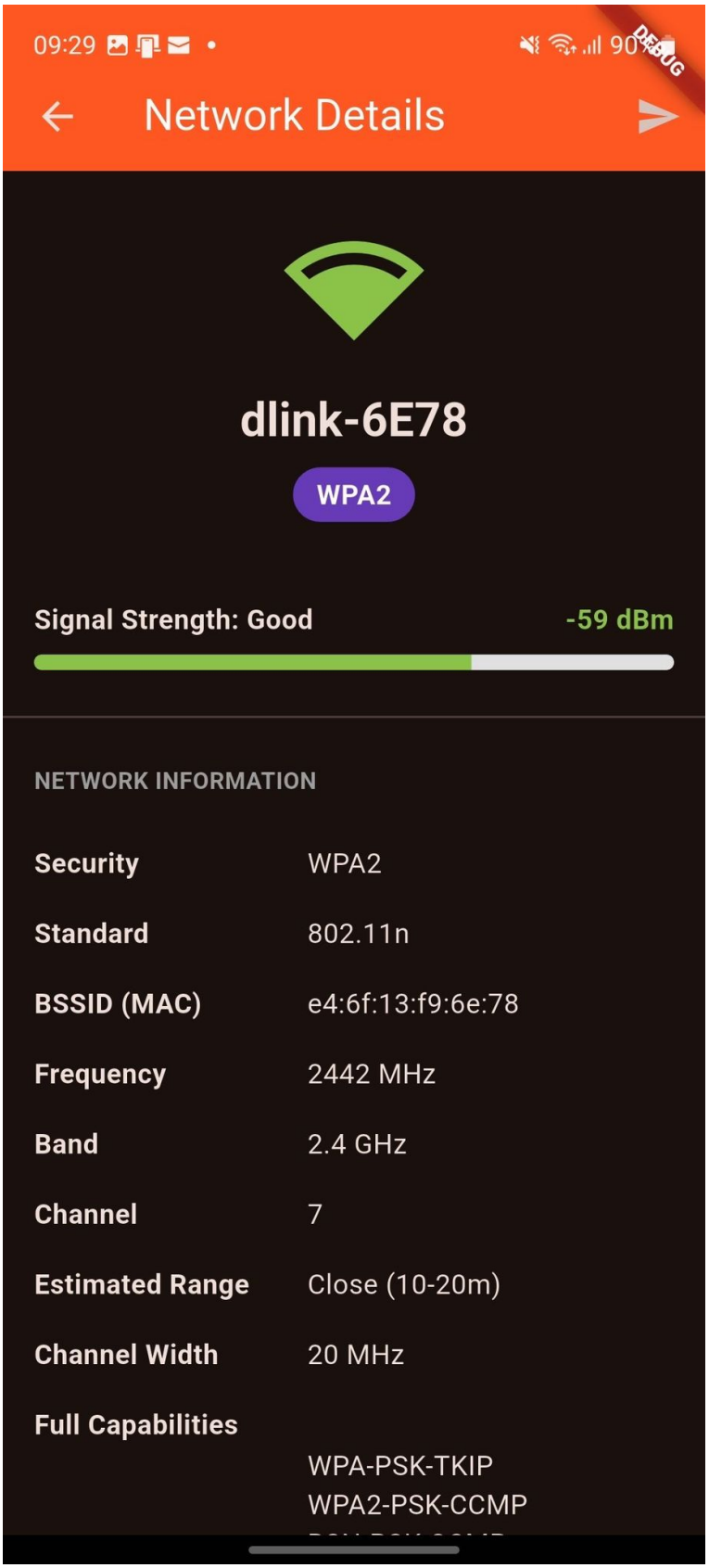

Figura 6: Detalii retea neconectata - 1

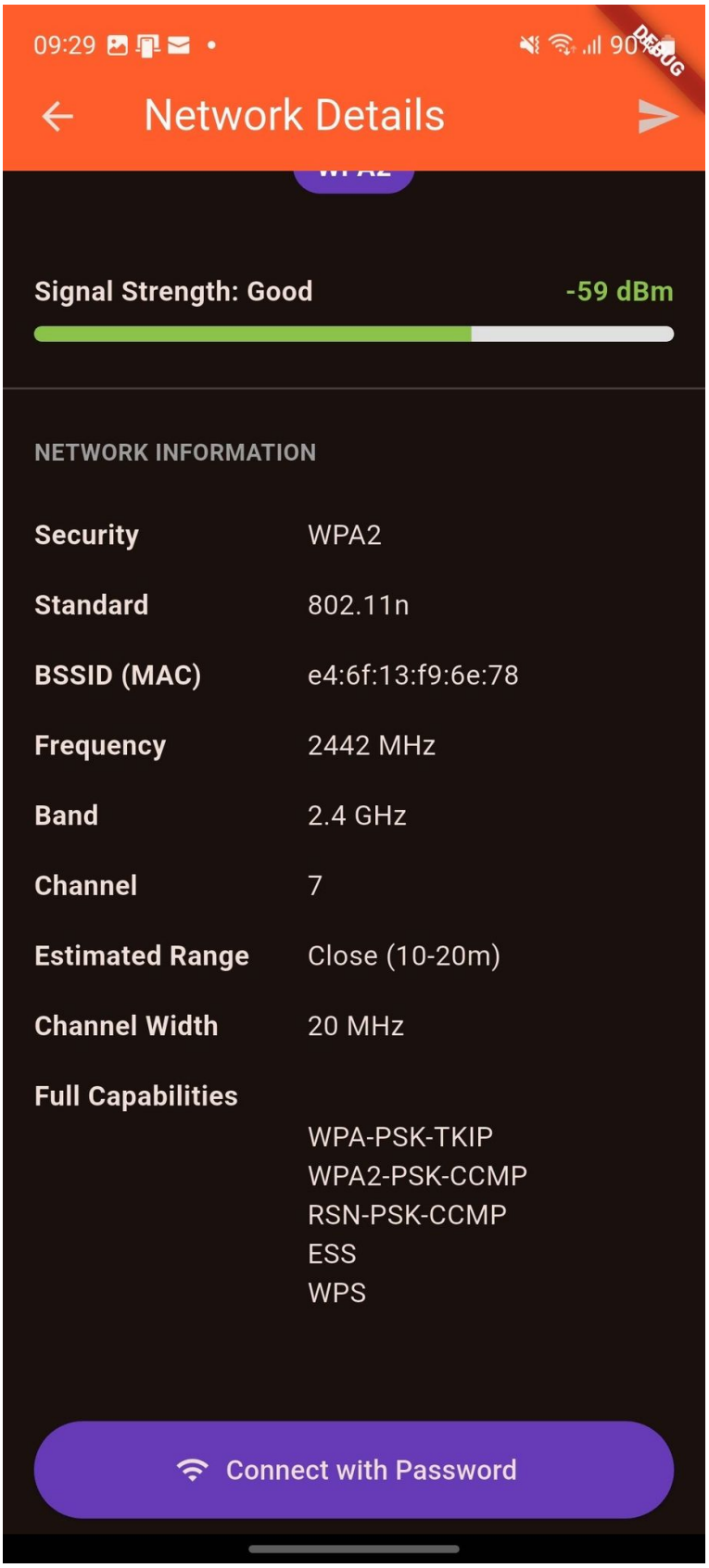

Figura 7: Detalii retea neconectata - 2

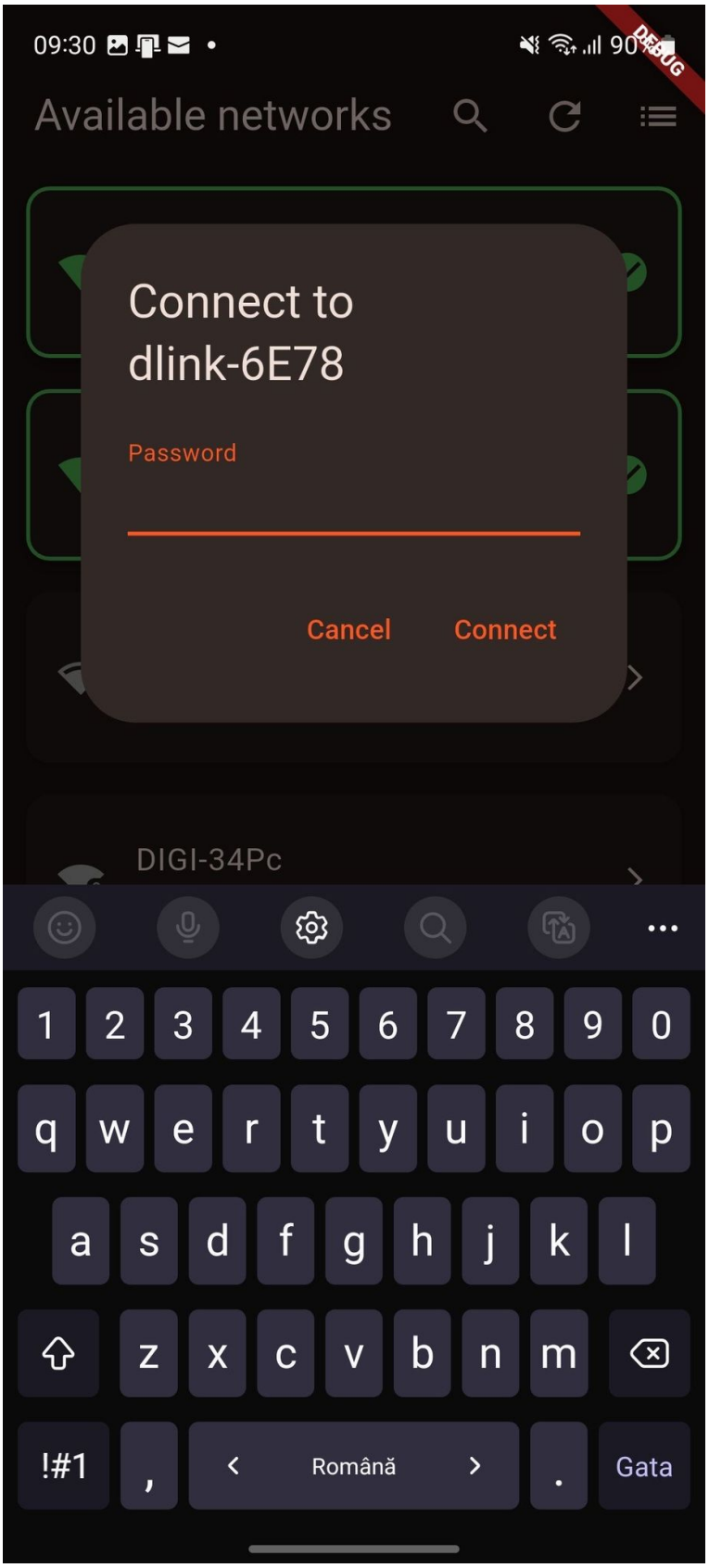

Figura 8: Conectare la retea

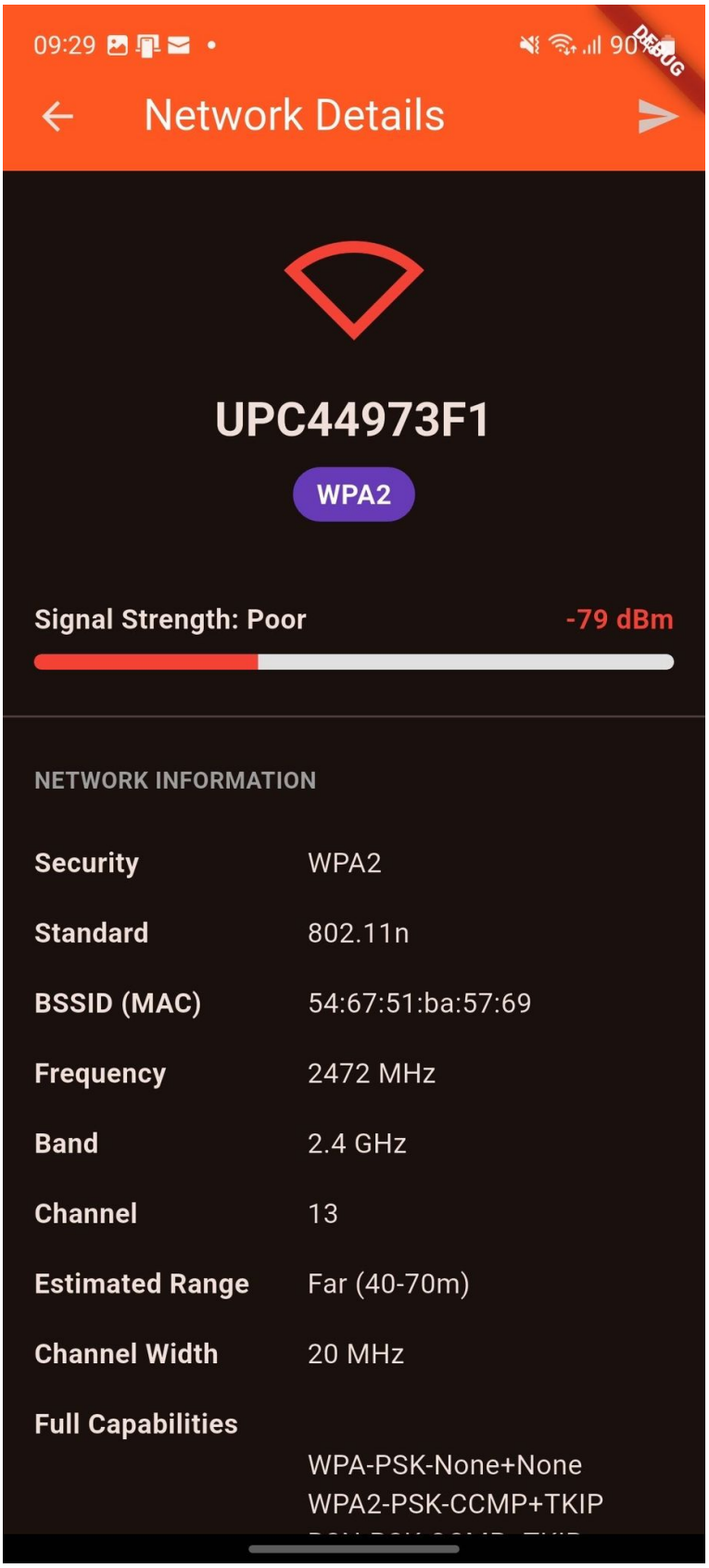

Figura 9: Retea cu semnal slab

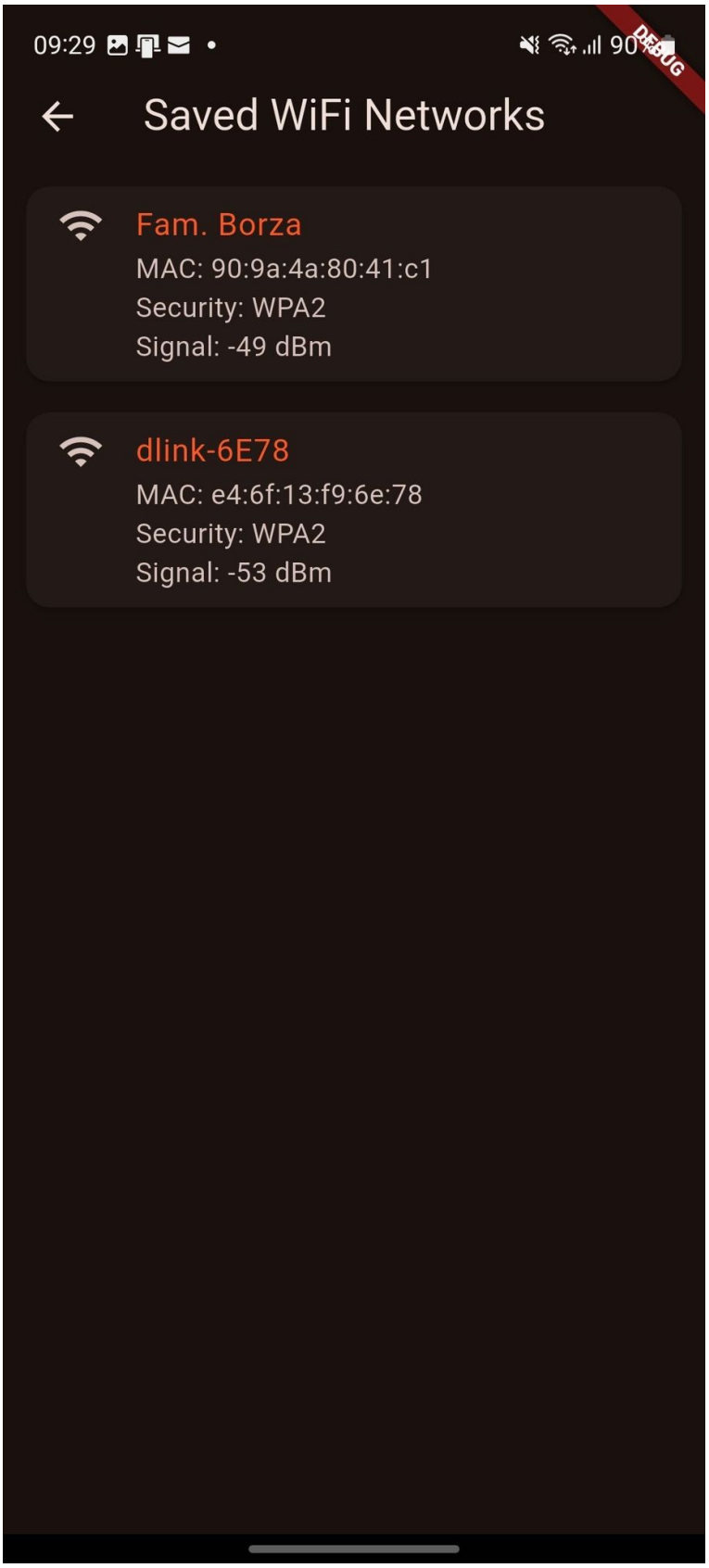

Figura 10: Retele salvate

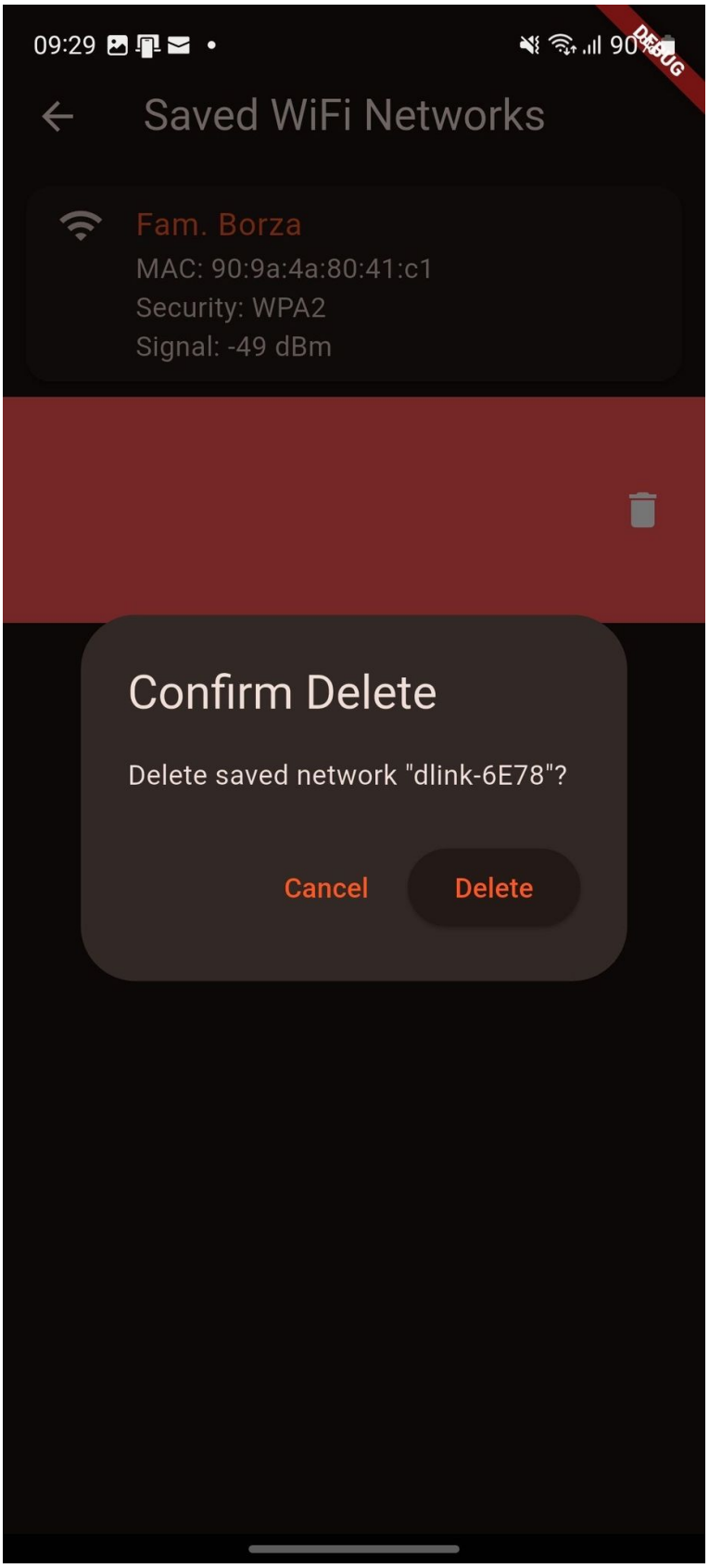

Figura 11: Stergere retea salvata