Proiect comunicatii mobile în React Native

Introducere

Acest proiect își propune să ghideze începătorii în crearea unei aplicații mobile folosind React Native, cu scopul specific de a gestiona și afișa informații legate de rețelele WiFi. Aplicația va avea capacitatea de a arăta detalii despre interfața WiFi a dispozitivului, de a lista dispozitivele WiFi disponibile în apropiere și de a facilita comunicarea între telefoane sau între telefon și server. Această inițiativă va oferi utilizatorilor o înțelegere practică a manipulării datelor de rețea prin intermediul unei platforme mobile populare și accesibile.

Motivația Alegerii React Native

React Native este un framework open-source dezvoltat de Facebook, conceput pentru a permite dezvoltatorilor să creeze aplicații mobile native utilizând JavaScript. Acesta oferă posibilitatea de a scrie codul o singură dată și de a-l rula pe multiple platforme, precum iOS și Android, economisind astfel timp și resurse în procesul de dezvoltare.

1. **Eficiență în Dezvoltare:** React Native permite dezvoltatorilor să scrie codul o singură dată și să-l ruleze atât pe iOS cât și Android. Acest lucru reduce timpul și costurile de dezvoltare, deoarece nu este necesar să se scrie cod separat pentru fiecare sistem de operare.

2. **Performanță Apropiată de Aplicațiile Native:** Deși React Native operează pe principiul "learn once, write anywhere", performanța aplicațiilor create este comparabilă cu aceea a aplicațiilor native. Aceasta se datorează utilizării componentelor native, care asigură o integrare fluidă și o interacțiune rapidă cu sistemul de operare al dispozitivului.

3. **Comunitate Vastă și Resurse Abundente:** React Native este susținut de Meta și o comunitate largă de dezvoltatori. Aceasta înseamnă că există o multitudine de resurse de învățare, biblioteci și template-uri de cod disponibile pentru a ajuta dezvoltatorii să rezolve probleme comune și să implementeze funcționalități avansate cu mai puțin efort.

4. **Agilitate în Testare:** Platforma suportă hot reloading și fast refresh, facilitând astfel testarea și iterarea rapidă a codului. Acest lucru permite dezvoltatorilor să vadă imediat

efectele modificărilor făcute în cod, fără a necesita instalarea din nou a aplicației pe un dispozitiv, accelerând ciclul de dezvoltare.

5. **Acces la Funcționalități Hardware Avansate:** React Native facilitează accesul la funcționalități hardware complexe, precum GPS, camera foto și senzorii dispozitivului. Pentru proiectul nostru, aceasta este o caracteristică crucială, deoarece necesită interacțiunea cu interfața de rețea WiFi a dispozitivului.

Implementare aplicație

Configurarea Mediului de Dezvoltare

Pentru a dezvolta o aplicație React Native pe Windows cu target pe Android, este necesar să urmăm câțiva pași specifici pentru configurarea mediului de dezvoltare. Acești pași asigură că avem toate uneltele necesare pentru a începe dezvoltarea. Iată pașii detaliate de configurare:

1. Instalarea Node.js și a unui Manager de Pachete:
Descărcați și instalați Node.js de pe site-ul oficial Node.js. Aceasta va instala și npm (node package manager), esențial pentru gestionarea bibliotecilor și pachetelor React Native.
O alternativă la npm este Yarn, care poate fi instalat prin rularea npm install -g yarn în terminal.
Asigurați-vă că versiunea instalată este Node 18 sau mai nouă.

2. Instalarea Java Development Kit (JDK):
React Native necesită JDK versiunea 11 sau mai nouă. Puteți descărca JDK de pe Java SE Development Kit (JDK). Se recomandă JDK 17 deoarece versiunile mai noi pot întâmpina probleme. Asigurați-vă că setați variabila de mediu JAVA_HOME pentru a indica directorul unde ați instalat JDK.

3. Instalarea Android Studio:
Android Studio va oferi SDK-ul și instrumentele necesare pentru dezvoltarea Android.
Descărcați și instalați Android Studio de pe pagina oficială Android Studio.
În timpul instalării, asigurați-vă că selectați "Android SDK", "Android SDK Platform", și "Android Virtual Device".
După instalare, deschideți Android Studio deschideți "SDK Manager":

- Selectați fila ”SDK Platforms” din SDK Manager, apoi bifați caseta de lângă „Afișați detaliile pachetului” în colțul din dreapta jos.
- Căutați și extindeți secțiunea Android 14 (UpsideDownCake), apoi asigurați-vă că sunt bifate următoarele elemente:
    - Android SDK Platform 34
    - Intel x86 Atom_64 System Image or Google APIs Intel x86 Atom System Image

4. Configurarea Variabilelor de Mediu Android:

Adăugați locația folderului Android/Sdk la variabila de mediu PATH. Aceasta se găsește în general sub C:\Users\<Your-Username>\AppData\Local\Android\Sdk.
Setează variabilele de mediu ANDROID_HOME și ANDROID_SDK_ROOT pentru a indica același director SDK.

5. Configurarea unui Dispozitiv Virtual Android (AVD):
În Android Studio, accesați "AVD Manager" și configurați un Android Virtual Device care să corespundă dispozitivului pe care doriți să testați aplicația.
Asigurați-vă că versiunea de Android instalată pe AVD este compatibilă cu cerințele aplicației React Native.

7. Crearea și Rularea Aplicației:
Creați un nou proiect React Native utilizând comanda:
npx react-native@latest init AwesomeProject
Navigați în directorul proiectului și lansați aplicația pe dispozitivul virtual cu react-native run-android.

Implementarea aplicației

1. **Descărcarea Scheletului Aplicației:**
   1.1. Începeți prin a descărca scheletul pregătit al aplicației din repository-ul de cod sursă sau prin linkul furnizat. Acesta include fișierele de configurare necesare și structura de bază a proiectului.
2. **Instalarea Pachetelor:**
   2.1. Deschideți un terminal în folderul principal al aplicației descărcate.
   2.2. Rulați comanda **npm install** sau **yarn** pentru a instala toate dependențele proiectului specificate în fișierul package.json. Acest pas este crucial pentru a asigura că toate pachetele necesare sunt instalate și pregătite pentru utilizare.
3. **Pornirea aplicației:**
   3.1. In terminal rulați comanda **npm run android** sau **yarn android.** Așteptați până când se deschide un emulator, și aplicația este rulată.

Aplicația include mai multe componente cheie deja implementate, care ajută la afișarea și gestionarea datelor de rețea:

Componenta Button: Această componentă este configurabilă și primește parametri precum un titlu, o funcție “onPress” ce va fi executată la apăsarea butonului, și un parametru de disable pentru dezactivarea butonului dacă este necesar. Acest setup permite adaptabilitatea butonului la diferite contexte de utilizare în cadrul aplicației.

Componenta NetInfo.tsx: Aceasta este o componentă care afișează informații detaliate despre placa de WiFi a dispozitivului. Furnizează date esențiale despre starea conexiunii de rețea, fiind

un instrument valoros pentru utilizatorii care necesită o perspectivă clară asupra performanței și stării lor de conectivitate.

Componenta Network: Prezintă detalii despre o rețea WiFi detectată, cum ar fi numele rețelei (SSID), intensitatea semnalului, și securitatea rețelei, folosind un parametru de tip WifiEntry.

```
export interface WifiEntry {
    SSID: string;
    BSSID: string;
    capabilities: string;
    frequency: number;
    level: number;
    timestamp: number;
}
```

Informații despre placa Wifi

Pentru a afla informații despre placa Wifi o sa folosim libraria [@react-native-community/netinfo](). In fisierul **Home.tsx**:

1. Importam librăria:
   ```
   import NetInfo from '../../components/netinfo/NetInfo';
   ```
2. Instantiem un obiect de tipul **NetInfoState** folosind hook-ul **useNetInfo:**
   ```
   const netInfo = useNetInfo();
   ```
3. Trimitem obiectul către componenta ce afișează datele despre placa Wifi:
   ```
   <ScrollView>
     <NetInfo netInfo={netInfo} />
   ```

Afișarea rețelelor disponibile

Pentru afișarea rețelelor disponibile o sa folosim librăria [react-native-wifi-reborn](). In fisierul **Home.tsx**:

1. Importăm librăria:
   ```
   import WifiManager, {WifiEntry} from 'react-native-wifi-reborn';
   ```
2. Creem un state în care să setăm lista de rețele odată ce le-am obținut:
   ```
   const [wifiList, setWifiList] = useState<WifiEntry[]>([]);
   ```
3. Adăugăm funcția **getNetworks** care primește ca și parametru o funcție care returnează o lista de rețele sub forma unui Promise (call-ul se face asincron) și cere permisiuni

pentru folosirea interfeței Wifi a dispozitivului:

```
const getNetworks = async (
  loadWifiList: () => Promise<WifiManager.WifiEntry[]>,
) => {
  if (Platform.OS === 'android') {
    try {
      const granted = await PermissionsAndroid.request(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
        {
          title: 'Location permission is required for WiFi connections',
          message:
            'This app needs location permission as this is required  ' +
            'to scan for wifi networks.',
          buttonNegative: 'DENY',
          buttonPositive: 'ALLOW',
        },
      );

      if (granted === PermissionsAndroid.RESULTS.GRANTED) {
        // You can now use react-native-wifi-reborn

        const wifiList = await loadWifiList();

        setWifiList(wifiList);

        console.log('wifi list', wifiList);
      } else {
        // Permission denied
        console.log('not granted');
      }      You, 21 hours ago • Network app
    } catch (error) {
      console.log('permerr', error);
    }
  }
};
```

4. Adăugăm un buton pentru a obține rețelele Wifi deja scanate. Trimitem ca si parametru al funcției getNetworks WifiManager.loadWifiList:

```
<Button
  title={'Get Networks'}
  onPress={() => getNetworks(WifiManager.loadWifiList)}
/>
```

5. Adăugăm un buton pentru a scana și încărca rețelele Wifi. Trimitem ca și parametru al funcției getNetworks WifiManager.reScanAndLoadWifiList:

```
<Button
  title={'Re-scan Wifi Networks'}
  onPress={() => getNetworks(WifiManager.reScanAndLoadWifiList)}
/>
```

Comunicare prin HTTP:

Pentru a implementa această funcționalitate deschidem fișierul **HttpRequest.tsx:**

1. Importam librăria **axios**:

```
import axios from 'axios';
```

2. Ca să introducem un url către care să facem un request adăugăm un TextInput:

```
<TextInput
  style={styles.input}
  onChangeText={setUrl}
  value={url}
  placeholder="Enter URL"
  autoCapitalize="none"
/>
```

3. Adăugăm funcția **handleHttpRequest** în interiorul căreia apelăm axios.get pentru a face un request de tipul GET catre url-ul **url** introdus in TextInput:

```
const handleHttpRequest = async () => {
  if (url) {
    setLoading(true);
    try {
      const response = await axios.get(url);
      setResponseText(response.data);
    } catch (error) {
      setResponseText('Failed to fetch data. Error: ' + error);
    } finally {
      setLoading(false);
    }
  }
};
```

4. Adăugăm un buton ca să apelăm funcția definită anterior:

```
<Button
  title={loading ? 'Loading...' : 'Fetch Data'}
  onPress={handleHttpRequest}
  disabled={loading}
/>
```

5. Pentru a putea vedea răspunsul adăugăm o componentă de tip Text:

```
<ScrollView style={styles.responseContainer}>
  <Text>{responseText}</Text>
</ScrollView>
```

Concluzie

Prin implementarea acestei aplicații mobile utilizând React Native, proiectul nostru a reușit să demonstreze modul în care tehnologiile moderne pot fi utilizate pentru a simplifica și eficientiza gestionarea și monitorizarea rețelelor WiFi prin dispozitive mobile. Utilizând componente specializate și librării open-source cum sunt @react-native-community/netinfo și react-native-wifi-reborn, aplicația oferă o interfață intuitivă și funcțională pentru vizualizarea informațiilor detaliate despre starea conexiunilor de rețea și pentru explorarea rețelelor WiFi disponibile.