Application de Journalisation des Positions GPS et des Niveaux de Signal

Introduction

Ce projet consiste en la création d'une application Android qui journalise les positions GPS d'un téléphone et le niveau de signal du réseau mobile (GSM, LTE, 5G) lorsqu'ils changent. Les données sont enregistrées dans un fichier CSV au format suivant :

YYYY-MM-DD HH:mm:ss; Latitude ; Longitude; Altitude; dbm; BatteryLevel

Les objectifs principaux de l'application sont :

- Journaliser les informations de localisation et de signal à intervalles réguliers.
- Gérer plusieurs langues (EN, FR, AR).
- Sauvegarder les données dans un fichier CSV pour une analyse ultérieure.

Structure du Code

Le code de l'application est divisé en plusieurs parties, dont chacune a des responsabilités spécifiques.

1. Importation des Bibliothèques

Le code commence par importer les bibliothèques nécessaires pour gérer la localisation, les permissions, le matériel du téléphone, et l'interface utilisateur :

```
import android.Manifest;

import android.content.Context;

import android.content.pm.PackageManager;

import android.location.Location;

import android.os.BatteryManager;

import android.os.Bundle;

import android.os.Looper;

import android.telephony.PhoneStateListener;

import android.telephony.SignalStrength;

import android.telephony.TelephonyManager;

import android.widget.Button;

import android.widget.TextView;

import android.widget.Toast;
```

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.core.app.ActivityCompat;

import com.google.android.gms.location.FusedLocationProviderClient;

import com.google.android.gms.location.LocationCallback;

import com.google.android.gms.location.LocationRequest;

import com.google.android.gms.location.LocationResult;

import com.google.android.gms.location.LocationServices;

2. Classe MainActivity

La classe MainActivity étend AppCompatActivity, ce qui permet de créer une interface utilisateur et de gérer les interactions de l'utilisateur.

- **Attributs** :
  - FusedLocationProviderClient fusedLocationProviderClient : pour obtenir des mises à jour de localisation.
  - LocationCallback locationCallback : pour recevoir les mises à jour de localisation.
  - TelephonyManager telephonyManager : pour écouter les changements de signal.
  - int dbm : pour stocker le niveau de signal.
  - boolean isTracking : pour savoir si le suivi est en cours.

```
public class MainActivity extends AppCompatActivity {

    private FusedLocationProviderClient fusedLocationProviderClient;  3 usages
    private LocationCallback locationCallback;  4 usages
    private TelephonyManager telephonyManager;  2 usages
    private int dbm = 0;  2 usages
    private boolean isTracking = false;  3 usages
```

3. Méthode onCreate

La méthode onCreate est appelée lorsque l'activité est créée. Elle initialise l'interface utilisateur et les services nécessaires :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button btnStartStop = findViewById(R.id.btnStartStop);
    TextView tvLastLog = findViewById(R.id.tvLastLog);

    fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);
    telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
```

- **Écoute des changements de signal** : La méthode onSignalStrengthsChanged est surchargée pour mettre à jour le niveau de signal dbm à chaque changement

```
// Écoute des changements de signal
telephonyManager.listen(new PhoneStateListener() {
    @Override 1 usage
    public void onSignalStrengthsChanged(SignalStrength signalStrength) {
        super.onSignalStrengthsChanged(signalStrength);
        dbm = signalStrength.getLevel();
    }
}, PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
```

4. Méthode startTracking

La méthode startTracking initie le suivi de la localisation :

```
private void startTracking(TextView tvLastLog) { 1 usage
    LocationRequest locationRequest = LocationRequest.create();
    locationRequest.setInterval(10000); // 10 secondes
    locationRequest.setFastestInterval(5000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

- **Gestion des permissions** : La méthode vérifie si les permissions de localisation sont accordées. Si ce n'est pas le cas, elle doit appeler ActivityCompat.requestPermissions.

- **Mise à jour de la localisation** : La méthode utilise FusedLocationProviderClient pour demander des mises à jour de localisation. À chaque nouvelle position, elle appelle logData pour journaliser les informations.

5. Méthode stopTracking

La méthode stopTracking arrête la mise à jour de la localisation :

```
    private void stopTracking() {  1 usage
        if (locationCallback != null) {
            fusedLocationProviderClient.removeLocationUpdates(locationCallback);
        }
    }
```

6. Méthode getBatteryLevel

Cette méthode récupère le niveau de la batterie du téléphone :

```
private int getBatteryLevel() {  1 usage
    BatteryManager batteryManager = (BatteryManager) getSystemService(BATTERY_SERVICE);
    return batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY);
}
```

7. Méthode logData

La méthode logData construit une entrée de journalisation et l'enregistre dans un fichier

```
private String logData(double latitude, double longitude, double altitude, int dbm, int batteryLevel) {  1 usage
    String timestamp = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.getDefault()).format(new Date());
    String logEntry = timestamp + "; " + latitude + "; " + longitude + "; " + altitude + "; " + dbm + "; " + batteryLevel;
```

```
    // Écriture dans un fichier CSV
    try {
        File file = new File(getExternalFilesDir( type: null),  child: "LogTracking.csv");
        FileWriter writer = new FileWriter(file,  append: true);
        writer.append(logEntry).append("\n");
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return logEntry;
}
```

Format des données : Les données sont formatées avec un timestamp et les valeurs de latitude, longitude, altitude, dbm, et niveau de batterie.

Internationalisation

Pour supporter plusieurs langues, vous devez ajouter des fichiers de ressources pour chaque langue dans le dossier res/values. Par exemple :

- res/values/strings.xml (anglais)

```
<resources>
    <string name="app_name">Internalisation</string>
    <string name="start_tracking">Start Tracking</string>
    <string name="stop_tracking">Stop Tracking</string>
    <string name="tracking_started">Tracking started</string>
    <string name="tracking_stopped">Tracking stopped</string>
    <string name="last_log">Last Log: None</string>
</resources>
```

- res/values-fr/strings.xml (français)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Internalisation</string>
    <string name="start_tracking">Démarrer le suivi</string>
    <string name="stop_tracking">Arrêter le suivi</string>
    <string name="tracking_started">Suivi démarré</string>
    <string name="tracking_stopped">Suivi arrêté</string>
    <string name="last_log">Dernier enregistrement : Aucun</string>
</resources>
```

res/values-ar/strings.xml (arabe)

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">التدوين الداخلي</string>
    <string name="start_tracking">بدء التتبع</string>
    <string name="stop_tracking">إيقاف التتبع</string>
    <string name="tracking_started">تم بدء التتبع</string>
    <string name="tracking_stopped">تم إيقاف التتبع</string>
    <string name="last_log">آخر سجل: لا شيء</string>
</resources>
```

```
    // Méthode pour changer la langue
    private void changeLanguage() {  1 usage
        if (currentLanguage.equals("en")) {
            currentLanguage = "fr"; // Changer à anglais
        } else {if (currentLanguage.equals("fr")) {
            currentLanguage = "fr"; // Changer à français
        } else
            currentLanguage = "ar"; // Changer à arabe
        }

        // Appliquer la langue
        Locale locale = new Locale(currentLanguage);
        Locale.setDefault(locale);
        Configuration config = new Configuration();
        config.setLocale(locale);
        getResources().updateConfiguration(config, getResources().getDisplayMetrics());
    }

    // Mettre à jour le texte selon la langue actuelle
    private void updateText() {  1 usage
        Button btnStartStop = findViewById(R.id.btnStartStop);
        TextView tvLastLog = findViewById(R.id.tvLastLog);
        Button btnChangeLanguage = findViewById(R.id.btnChangeLanguage);

        btnStartStop.setText(R.string.start_tracking);
        tvLastLog.setText(R.string.last_log);
        btnChangeLanguage.setText(R.string.change_language);
    }
}
```

Chaque fichier doit contenir des traductions des chaînes utilisées dans l'application, comme les textes des boutons et les messages Toast.

Détails de la méthode changeLanguage()

1. **Changement de langue** :
    - La méthode vérifie d'abord la langue actuelle en utilisant la variable currentLanguage, qui est supposée contenir le code de langue en cours (par exemple, "en" pour l'anglais, "fr" pour le français, et "ar" pour l'arabe).
    - Si la langue actuelle est l'anglais ("en"), elle est changée en français ("fr").
    - Si la langue actuelle est le français ("fr"), il n'y a pas de changement (c'est une redondance qui pourrait être simplifiée).
    - Si la langue actuelle n'est ni l'anglais ni le français, elle est changée en arabe ("ar").
2. **Application de la langue** :
    - Un objet Locale est créé avec la langue sélectionnée (par exemple, new Locale(currentLanguage)).
    - Locale.setDefault(locale) définit la locale par défaut de l'application.
    - Un nouvel objet Configuration est créé, et la méthode setLocale(locale) est appelée pour définir la nouvelle locale.
    - getResources().updateConfiguration(config, getResources().getDisplayMetrics()) met à jour la configuration des ressources de l'application avec la nouvelle locale, ce qui permet aux chaînes de ressources d'être correctement chargées en fonction de la langue sélectionnée.

Détails de la méthode updateText()

1. **Récupération des éléments de l'interface** :
    - Les éléments de l'interface utilisateur, comme les boutons et le texte, sont récupérés à l'aide de findViewById(). Cela permet de faire référence aux éléments définis dans le fichier de mise en page XML.
2. **Mise à jour des textes** :
    - btnStartStop.setText(R.string.start_tracking) : Cette ligne met à jour le texte du bouton pour démarrer ou arrêter le suivi. Le texte affiché est chargé à partir des ressources de chaîne, ce qui signifie qu'il changera en fonction de la langue actuelle.

- tvLastLog.setText(R.string.last_log) : Cette ligne met à jour le texte d'affichage pour le dernier enregistrement de localisation.
- btnChangeLanguage.setText(R.string.change_language) : Cette ligne met à jour le texte du bouton permettant de changer la langue.

Conclusion

Cette application permet de suivre et de journaliser la position GPS et le niveau de signal d'un appareil mobile de manière efficace. Elle est extensible pour prendre en charge plusieurs langues et offre une interface utilisateur simple.