



Programmation mobile

Compte-rendu TP2

Les capteurs

Louis Jean
Master 1 IMAGINE
Université de Montpellier
N° étudiant : 21914083
<https://github.com/louis-jean0/HAI811I-mobile>

17 mars 2024

Table des matières

1	Introduction	2
2	Liste de capteurs	3
3	Détection de présence/absence de capteurs	5
4	Accéléromètre	7
5	Direction	9
6	Secouer un appareil	11
7	Proximité	14
8	Géolocalisation	16
9	Conclusion	18

1 Introduction

Dans le cadre de ce deuxième TP, nous avons exploré divers aspects du développement d'applications mobiles Android, en mettant l'accent sur l'utilisation des capteurs. À travers une série d'exercices progressifs, nous avons abordé la manipulation de données issues de capteurs variés, comme l'accéléromètre, le capteur de proximité, ainsi que l'exploitation des services de localisation. En intégrant ces fonctionnalités, nous avons cherché à concevoir des applications qui non seulement répondent aux besoins fonctionnels mais offrent également une expérience utilisateur intuitive et engageante. Ce compte-rendu détaille le processus de développement étape par étape de chaque application, mettant en lumière les défis rencontrés et les solutions adoptées pour les surmonter.

N.B : tout au long de ce TP, j'ai travaillé sur Android Studio, en émulant le téléphone Pixel 3a, à défaut d'avoir un appareil Android à disposition.

2 Liste de capteurs

Le but de cet exercice était de découvrir comment interagir avec les capteurs disponibles sur un appareil Android à l'aide de l'API **Sensor**. J'ai appris à utiliser le **SensorManager** pour accéder à la liste des capteurs et à les afficher dans une interface utilisateur simple grâce à une **ListView**.

```
1 public class MainActivity extends Activity {
2     private SensorManager sensorManager;
3     private ListView listView;
4     private List<String> listSensorNames;
5     @Override
6     protected void onCreate(Bundle savedInstanceState) {
7         super.onCreate(savedInstanceState);
8         setContentView(R.layout.activity_main);
9         listView = (ListView) findViewById(R.id.←
10            ← sensorsListView);
11         sensorManager = (SensorManager) getSystemService(←
12            ← SENSOR_SERVICE);
13         List<Sensor> sensorList = sensorManager.←
14             ← getSensorList(Sensor.TYPE_ALL);
15         listSensorNames = new ArrayList<>();
16         for (Sensor sensor : sensorList) {
17             listSensorNames.add(sensor.getName() + " - " + ←
18                ← sensor.getType());
19         }
20     }
21 }
```

Ce code initialise le SensorManager, récupère la liste de tous les capteurs disponibles (Sensor.TYPE_ALL), extrait leurs noms et types, puis affiche ces informations dans une ListView via un ArrayAdapter.

Cette activité m'a permis de comprendre l'importance de lister les capteurs disponibles sur un appareil, car cela peut être crucial pour développer des applications qui s'adaptent aux capacités matérielles spécifiques de l'appareil sur lequel elles sont installées.

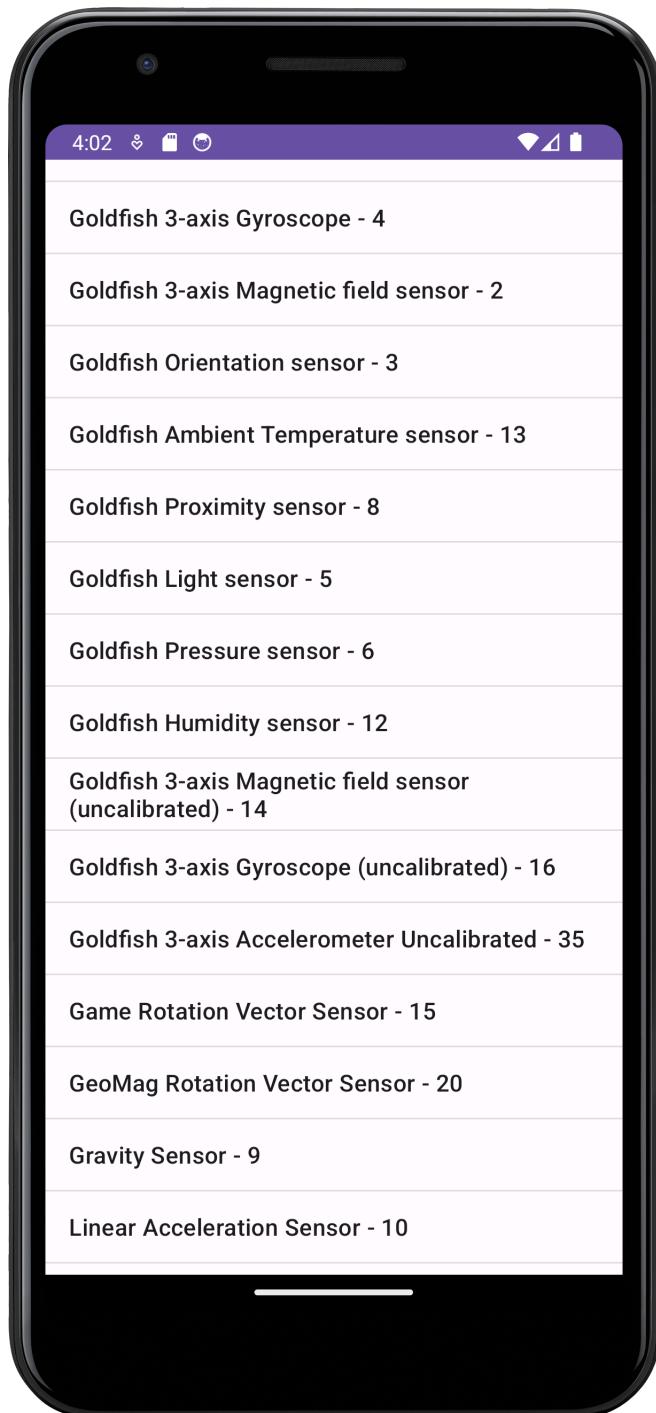


Figure 1: Liste de capteurs

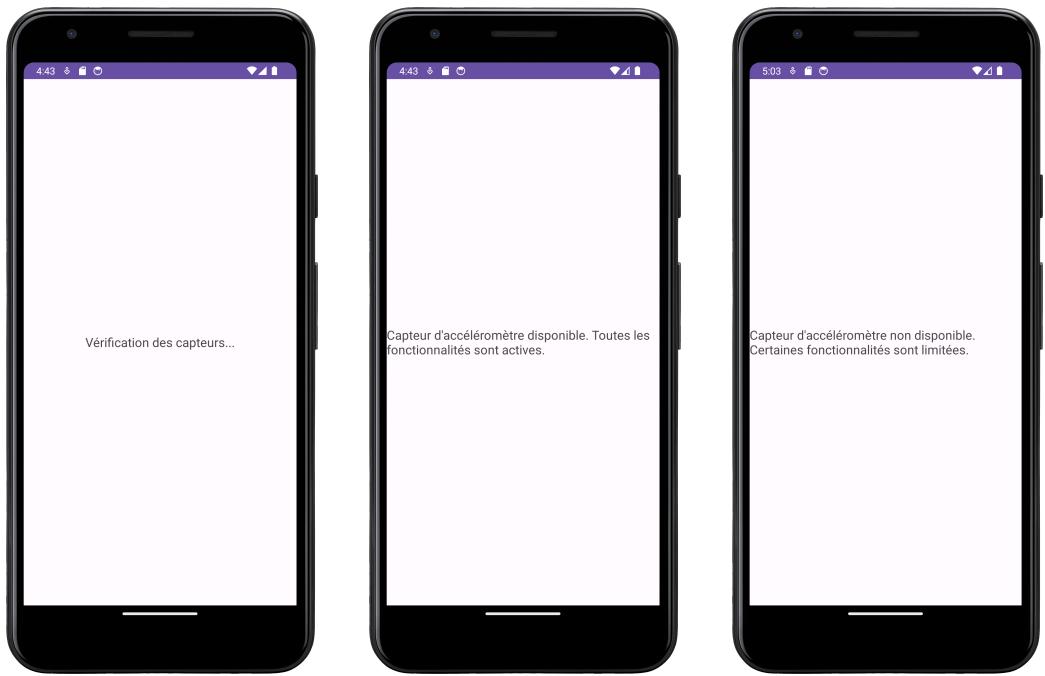
3 Détection de présence/absence de capteurs

Dans cet exercice, j'ai utilisé le **SensorManager** pour vérifier la disponibilité d'un capteur spécifique sur l'appareil.

J'ai configuré mon application pour afficher le message "Vérification des capteurs..." pendant 3 secondes avant de procéder à la vérification de la disponibilité d'un capteur spécifique, améliorant ainsi l'expérience utilisateur en fournissant un feedback visuel pendant le processus de vérification.

```
1 public class MainActivity extends Activity {
2     private SensorManager sensorManager;
3     private TextView sensorStatusTextView;
4     @Override
5     protected void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         setContentView(R.layout.activity_main);
8         sensorStatusTextView = findViewById(R.id.←
9             → tvSensorStatus);
10        sensorManager = (SensorManager) getSystemService(←
11            → SENSOR_SERVICE);
12        new Handler().postDelayed(new Runnable() {
13            @Override
14            public void run() {
15                checkSensorAvailability();
16            }
17        }, 3000);
18    }
19
20    private void checkSensorAvailability() {
21        Sensor sensor = sensorManager.getDefaultSensor(←
22            → Sensor.TYPE_ACCELEROMETER);
23        if (sensor == null) {
24            sensorStatusTextView.setText("Capteur d'accélé←
25            → romètre non disponible. Certaines fonctionnalités sont←
26            → limitées.");
27        } else {
28            sensorStatusTextView.setText("Capteur d'accélé←
29            → romètre disponible. Toutes les fonctionnalités sont ←
30            → actives.");
31        }
32    }
33}
```

Ce code vérifie si l'accéléromètre est présent sur l'appareil, et affiche un message en conséquence.



(a) Vérification des capteurs

(b) Accéléromètre détecté

(c) Accéléromètre absent

Figure 2: Détection de présence/absence de l'accéléromètre

4 Accéléromètre

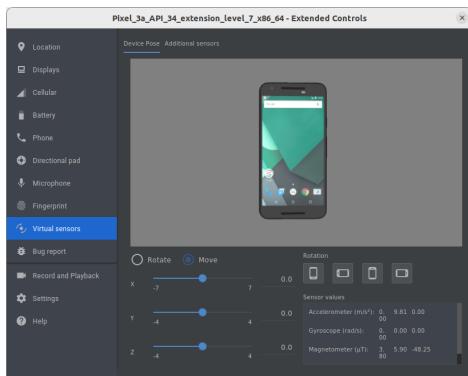
Dans cet exercice, j'ai intégré l'accéléromètre dans une application pour changer dynamiquement la couleur de fond de l'écran en fonction de la magnitude de l'accélération. Cela m'a permis de comprendre comment interagir avec les capteurs matériels et de réaliser une application réactive qui s'adapte aux mouvements de l'appareil. La magnitude de l'accélération a est calculée à l'aide de la formule :

$$a = \sqrt{x^2 + y^2 + z^2}$$

où x , y , et z sont les accélérations mesurées sur les axes X, Y, et Z, respectivement.

```
1 @Override
2 public void onSensorChanged(SensorEvent event) {
3     float x = event.values[0];
4     float y = event.values[1];
5     float z = event.values[2];
6     double magnitude = Math.sqrt(x*x + y*y + z*z);
7     final double SEUIL_BAS = 9.81;
8     final double SEUIL_HAUT = 12;
9     if(magnitude < SEUIL_BAS) {
10         layoutBackground.setBackgroundColor(getResources().←
11             getColor(android.R.color.holo_green_dark));
12     } else if (magnitude < SEUIL_HAUT) {
13         layoutBackground.setBackgroundColor(getResources().←
14             getColor(android.R.color.black));
15     } else {
16         layoutBackground.setBackgroundColor(getResources().←
17             getColor(android.R.color.holo_red_dark));
18     }
19 }
```

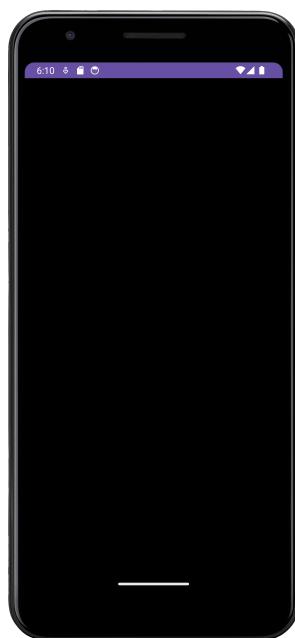
Cette fonction, qui est appelée à chaque fois qu'un changement est détecté au niveau du capteur, calcule la magnitude, et ajuste la couleur de fond de l'écran en fonction de cette dernière. L'écran est rouge si la magnitude est en dessous du seuil bas, vert si elle est comprise entre les deux seuils, et rouge si elle est au dessus du seuil haut.



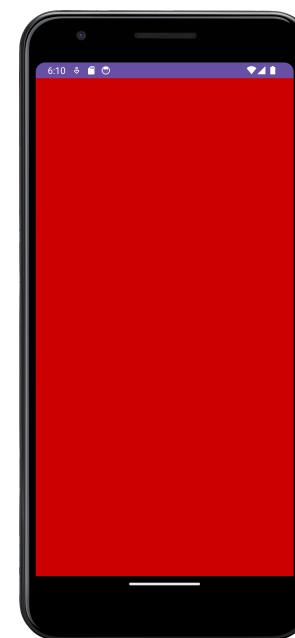
(a) Écran de contrôle du mouvement (via émulateur)



(b) Magnitude inférieure au seuil bas



(c) Magnitude comprise entre les deux seuils



(d) Magnitude supérieure au seuil haut

Figure 3: Contrôle de l'accéléromètre

5 Direction

Dans cet exercice, j'ai développé une application utilisant l'accéléromètre pour détecter et afficher en temps réel la direction du mouvement de l'appareil.

```
1  @Override
2  public void onSensorChanged(SensorEvent event) {
3      float x = event.values[0];
4      float y = event.values[1];
5      final float SEUIL = 2.0f;
6      if (Math.abs(x) > Math.abs(y)) {
7          if(Math.abs(x) > SEUIL) {
8              if (x < 0) {
9                  directionTextView.setText("Droite");
10             } else {
11                 directionTextView.setText("Gauche");
12             }
13         }
14     } else {
15         if(Math.abs(y) > SEUIL) {
16             if (y < 0) {
17                 directionTextView.setText("Bas");
18             } else {
19                 directionTextView.setText("Haut");
20             }
21         }
22     }
23 }
```

La logique de cette fonction évalue d'abord si le mouvement de l'appareil est plus prononcé sur l'axe X (gauche/droite) ou sur l'axe Y (haut/bas) en comparant les valeurs absolues des accélérations sur ces axes. Elle utilise ensuite un seuil défini pour déterminer si le mouvement détecté est significatif. Si le mouvement dépasse ce seuil, la fonction met à jour le texte d'un **TextView** pour indiquer la direction du mouvement.



(a) Haut



(b) Bas



(c) Gauche



(d) Droite

Figure 4: Directions du mouvement de l'appareil

6 Secouer un appareil

Pour allumer/éteindre le flash selon une secousse ressentie par l'appareil, j'ai eu du mal. En effet, il faut d'abord vérifier qu'au moins une des caméras du dos de l'appareil possède un flash, en mettant un garde-fou pour les erreurs possibles. De plus, en étant sous émulateur, ce n'est pas très pratique de s'assurer du bon fonctionnement du flash.

```
1 CameraManager manager = (CameraManager) getSystemService(←
  ↪ Context.CAMERA_SERVICE);
2 try {
3     for (String id : manager.getCameraIdList()) {
4         CameraCharacteristics characteristics = manager.←
  ↪ getCameraCharacteristics(id);
5         Boolean flashAvailable = characteristics.get(←
  ↪ CameraCharacteristics.FLASH_INFO_AVAILABLE);
6         Integer lensFacing = characteristics.get(←
  ↪ CameraCharacteristics.LENS_FACING); // Pour savoir si ←
  ↪ la caméra est une caméra arrière ou avant
7         if(flashAvailable != null && flashAvailable && ←
  ↪ lensFacing != null && lensFacing == ←
  ↪ CameraCharacteristics.LENS_FACING_BACK) {
8             cameraId = id;
9             break;
10        }
11    }
12 } catch (CameraAccessException e) {e.printStackTrace();}
```

Avec ce code, inséré dans la méthode **onCreate**, on est sûr de trouver une caméra compatible, ou de renvoyer une erreur propre.

```
1 private void toggleFlashLight() {
2     CameraManager camera = (CameraManager) getSystemService(←
  ↪ Context.CAMERA_SERVICE);
3     try {
4         if (isFlashOn) {
5             camera.setTorchMode(cameraId, false);
6             isFlashOn = false;
7             flashStatusTextView.setText("Flash : OFF");
8         } else {
9             camera.setTorchMode(cameraId, true);
10            isFlashOn = true;
11            flashStatusTextView.setText("Flash : ON");
12        }
13    } catch(CameraAccessException e) {e.printStackTrace();}
14 }
```

Ce code permet l'activation/la désactivation du flash, selon le booléen global **isFlashOn** qui indique l'état actuel du flash.

```

1  @Override
2  public void onSensorChanged(SensorEvent event) {
3      float x = event.values[0];
4      float y = event.values[1];
5      float z = event.values[2];
6      double magnitude = Math.sqrt(x*x + y*y + z*z);
7      final double SEUIL = 10.0;
8      if(magnitude > SEUIL) {
9          toggleFlashLight();
10     }
11 }
```

Ici, on fait appel à `toggleFlashLight()` uniquement si la secousse détectée est significative (réglé par le seuil défini).

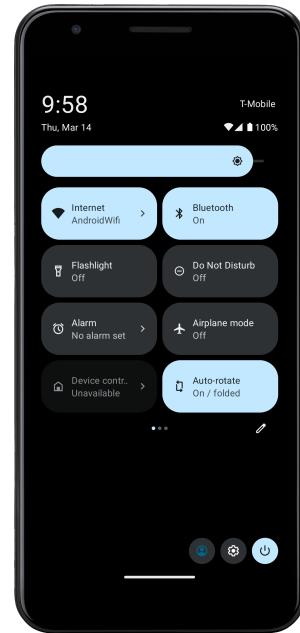
N.B : il ne faut pas oublier de rajouter les permissions dans le manifest, comme ceci.

```

1 <uses-permission android:name="android.permission.CAMERA"/>
2 <uses-feature android:name="android.hardware.camera"/>
3 <uses-permission android:name="android.permission.FLASHLIGHT" ↵
   ↪ "/>
4 <uses-feature android:name="android.hardware.camera.flash" ↵
   ↪ android:required="false"/>
```



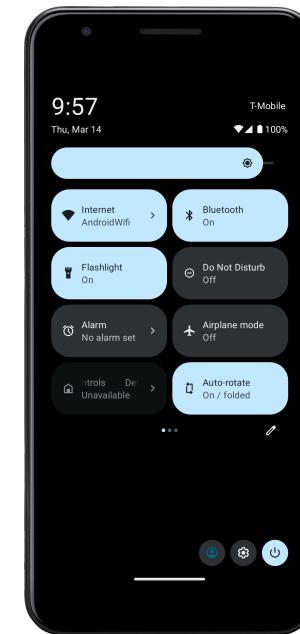
(a) Avant secousse



(b) Flash éteint (avant secousse)



(c) Après secousse



(d) Flash allumé (après secousse)

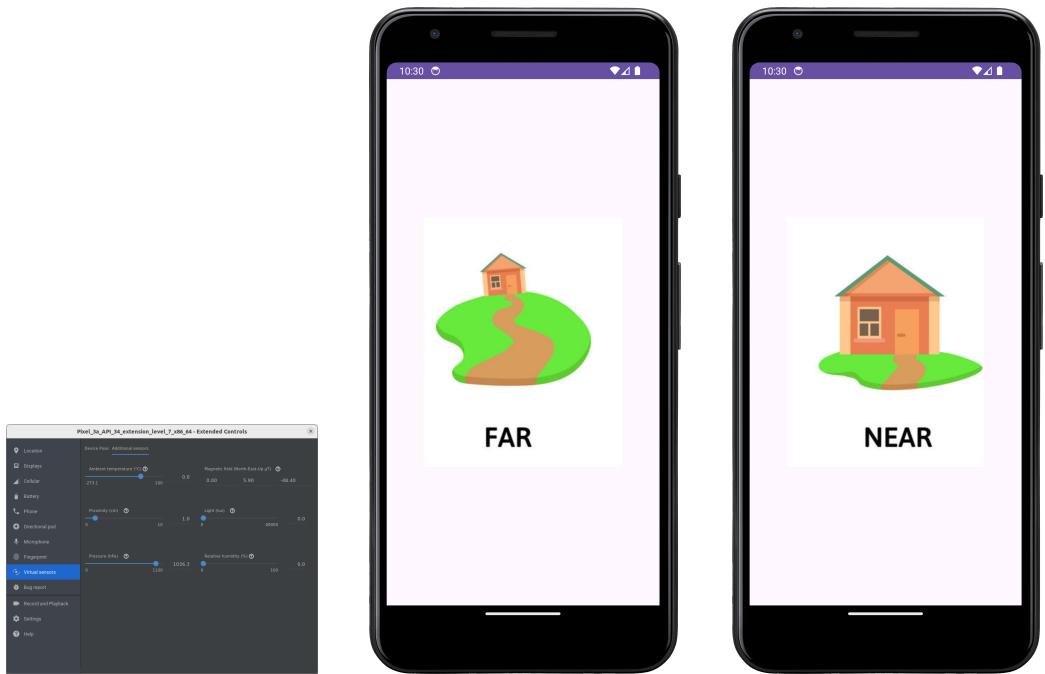
Figure 5: État de l'application et du flash selon le mouvement de l'appareil

7 Proximité

Pour utiliser le capteur de proximité et afficher une image en fonction de la valeur qu'il renvoie, j'ai commencé par ajouter deux images *far.png* et *near.png* dans **res/drawable**.

```
1 @Override
2 public void onSensorChanged(SensorEvent event) {
3     if (event.sensor.getType() == Sensor.TYPE_PROXIMITY) {
4         if (event.values[0] < proximitySensor.↪
5             ↪ getMaximumRange()) {
6             imageView.setImageResource(R.drawable.near);
7         } else {
8             imageView.setImageResource(R.drawable.far);
9         }
10 }
```

Cette fonction, appelée à chaque fois qu'un changement sur le capteur de proximité est détecté, compare la valeur actuelle de ce capteur à sa valeur maximale (en l'occurrence, sa portée), affiche l'image correspondant à proche si cette valeur est plus petite que la portée, et affiche l'image correspondant à loin sinon. **N.B :** il semble que sur l'appareil ici émulé (Pixel 3a), la portée du capteur de proximité soit de 1cm.



(a) Écran de contrôle de la proximité (via émulateur)

(b) Image affichée quand proximité $\geq 1\text{cm}$

(c) Image affichée quand proximité $\leq 1\text{cm}$

Figure 6: Contrôle de l'affichage en fonction du capteur de proximité

8 Géolocalisation

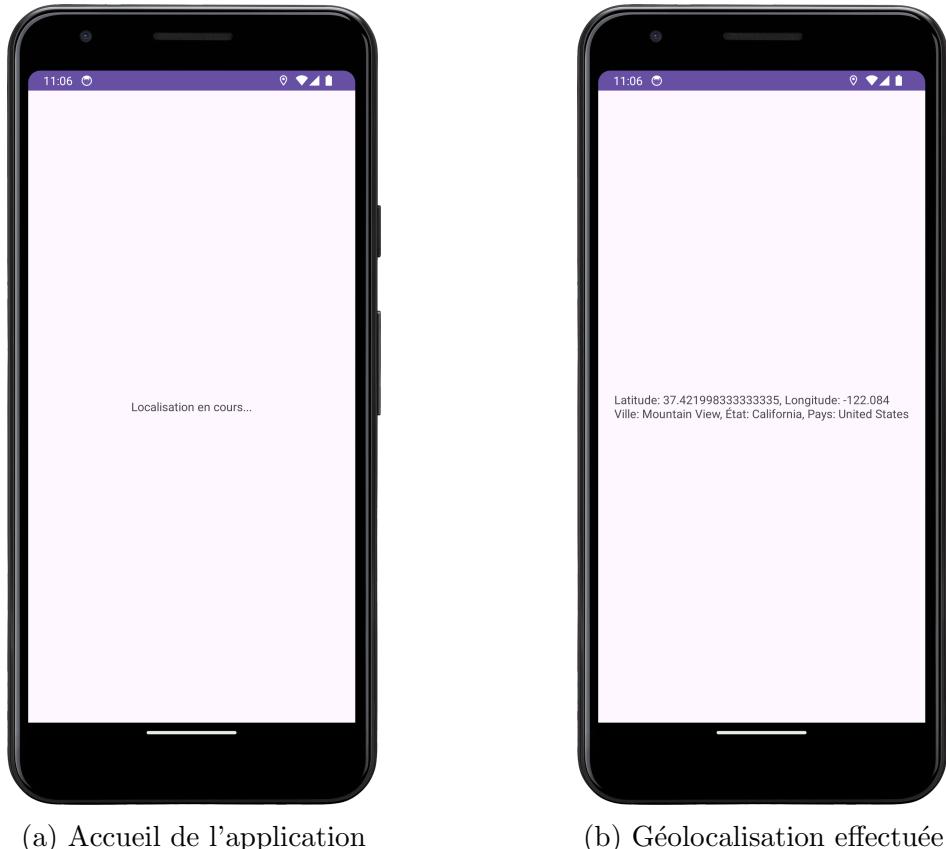
Premièrement, pour réaliser une application qui utilise la géolocalisation, on rajoute les permissions dans le manifest.

```
1 <uses-permission android:name="android.permission.←
  ↪ ACCESS_FINE_LOCATION" />
2 <uses-permission android:name="android.permission.←
  ↪ ACCESS_COARSE_LOCATION" />
```

J'ai ensuite utilisé **Geocoder** pour inverser les latitudes/longitudes obtenues et remonter la localisation sous forme de ville, état et pays.

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_main);
5     locationTextView = findViewById(R.id.locationTextView);
6     locationManager = (LocationManager) getSystemService(←
  ↪ LOCATION_SERVICE);
7     locationListener = new LocationListener() {
8         @Override
9         public void onLocationChanged(Location location) {
10             String text = "Latitude: " + location.←
  ↪ getLatitude() + ", Longitude: " + location.←
  ↪ getLongitude();
11             Geocoder geocoder = new Geocoder(MainActivity.←
  ↪ this, Locale.getDefault());
12             try {
13                 List<Address> addresses = geocoder.←
  ↪ getFromLocation(location.getLatitude(), location.←
  ↪ getLongitude(), 1);
14                 if (!addresses.isEmpty()) {
15                     Address address = addresses.get(0);
16                     String city = address.getLocality();
17                     String state = address.getAdminArea();
18                     String country = address.getCountryName←
  ↪ ();
19                     text += "\nVille: " + city + ", État: " ←
  ↪ + state + ", Pays: " + country;
20                 }
21             } catch (IOException e) {
22                 e.printStackTrace();
23             }
24             locationTextView.setText(text);
25         }
26     }
27 }
```

L'application affiche "Ville : Mountain View, État : California, Pays : United States" car l'émulateur ne dispose pas de vraie géolocalisation et utilise donc la localisation par défaut, à savoir le siège de Google.



(a) Accueil de l'application

(b) Géolocalisation effectuée

Figure 7: Application de géolocalisation

9 Conclusion

En conclusion, j'ai non seulement acquis des compétences techniques spécifiques, mais ai également développé une compréhension plus profonde de la manière dont les applications modernes peuvent tirer parti des fonctionnalités matérielles et logicielles des dispositifs mobiles. Ce TP a été une occasion précieuse de renforcer mes compétences en programmation Android tout en explorant les possibilités qu'offre cette plateforme pour le développement d'applications mobiles.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.