Projet Crowdsourcing

UE Domaine d'application de l'IoT Critique

2024 - 2025, Dépt. SN, Parcours E
Katia Jaffrès-Runser, Pr. - Toulouse INP ENSEEIHT
Credits: Quentin Bailleul et Raphaël Rouseyrol.

Contexte du Projet

Ce projet a pour objectif de vous donner les moyens de développer une application Android qui permette de mesurer régulièrement des données issues des capteurs d'un smartphone, de les enregistrer dans un base de données, et de les afficher à l'utilisateur.

Objectifs d'apprentissage

- Comprendre les nouveaux framework de développement Android basés sur :
  - Architecture Modèle-Vue-Contrôleur d'Android Jetpack
  - Le langage Kotlin
  - Les co-routines et les flux
- Savoir développer une application avec Jetpack Compose, et les ViewModel pour la gestion des données dans l'interface utilisateur.
- Savoir accéder aux données de localisation du smartphone.
- Savoir enregistrer les données dans une base de données et les afficher.

Une brève description du contexte

L'objectif principal de ce projet est de vous donner les outils pour construire une application Android capable de mesurer à intervalles réguliers l'activité de leur utilisateur ou de son environnement, via les différents capteurs et sondes offertes part un smartphone et accessibles via une API Android moderne. Ces bases vous permettront de savoir comment développer différentes applications très utiles actuellement dans différents contextes :

- Device monitoring : mesure de l'utilisation des resources du smartphone pour en connaitre les per- formances au cours du temps (monitoring réseau, monitoring système, ..)
- Quantified self : mesure de l'activité de l'utilisateur pour qu'il puisse mesurer son comportement
- Crowdsourcing : mesure de l'activité ou de données qui sont poussées vers un serveur de façon à améliorer la compréhension d'un phénomène, contribuer à une tâche.

Déroulement

Ce module se compose de 5 séances décomposées en 4 séances de travail et une séance de recette. Il est organisé sous la forme d'une classe inversée qui se décompose en deux parties :

1. Prise en main des bases du développement moderne en Android
2. Approfondissement d'une des fonctionnalités nécessaire au développement d'une application de Crowdsourcing.

Pour la première partie, la suite de ce document vous fournira des exercices à réaliser individuellement pour prendre en main les bases du développement Android actuel :

- La création de l'interface utilisateur minimale de votre application, avec `Android Jetpack` (outils Compose et ViewModel).
- La mesure des données de localisation d'un smarphone soit à la demande, soit en tâche de fond.

Quand votre apprentissage sera terminé, il vous est demandé pour la phase d'approfondissement, de travailler **en groupe de 3 étudiants** pour préparer, pour la dernière séance, une présentation des concepts et la démonstration d'une application qui permette de réaliser une des fonctionnalités nécessaires à une application de crowdsourcing.

> Attention, il ne vous est pas démandé dans le temps impartit de réaliser une application complète de crowdsourcing.

Voici quelques actions que vous pourriez choisir d'approfondir :

- La mesure et l'enregistrement d'une donnée dans une base de données (avec les outils `Room` ou `MySql` par exemple)
- L'affichage dynamique de données avec un flux (`Flow`)
- L'automatisation de la mesure de données en tâche de fonds avec les classes [`Worker` & `WorkManager`]
- La production de graphiques à partir de données (pas forcément enregistrées dans un base de données).

Organisation des séances :

- Séance 1 et 2 : apprentissage via les exercices et tutoriels mis à disposition
- Séance 3 : définition des binômes et choix du sujet à approfondir pour l'évaluation.
- Séance 4 : travail sur le sujet.
- Séance 5 : Présentation et démonstration par binôme du travail réalisé

Evaluation

La principale séance d'évaluation aura lieu en séance 5. Chaque groupe de projet devra présenter son travail sous la forme suivante :

[Présentation - 10 minutes max]

Il vous est demandé de préparer une présentation du sujet d'approfondissement qui inclut une démonstration de votre projet Android. Cette présentation devra permettre aux étudiants du cours de comprendre les objets que vous avez manipulés et comment ils s'intègrent à une application de Crowdsourcing.

[Questions des autres étudiants - 10 minutes max]

Une séance de questions ouverte aux autres étudiants fait suite à la présentation.

[Questions de l'enseignant - 10 minutes max]

Une séance de questions de l'enseignant sur le projet produit.

[Rendus]

Il vous est demandé de rendre :

- votre présentation et les éventuelles ressources que vous avez utilisées (à référencer correctement) au format PDF,

- un projet AndroidStudio,

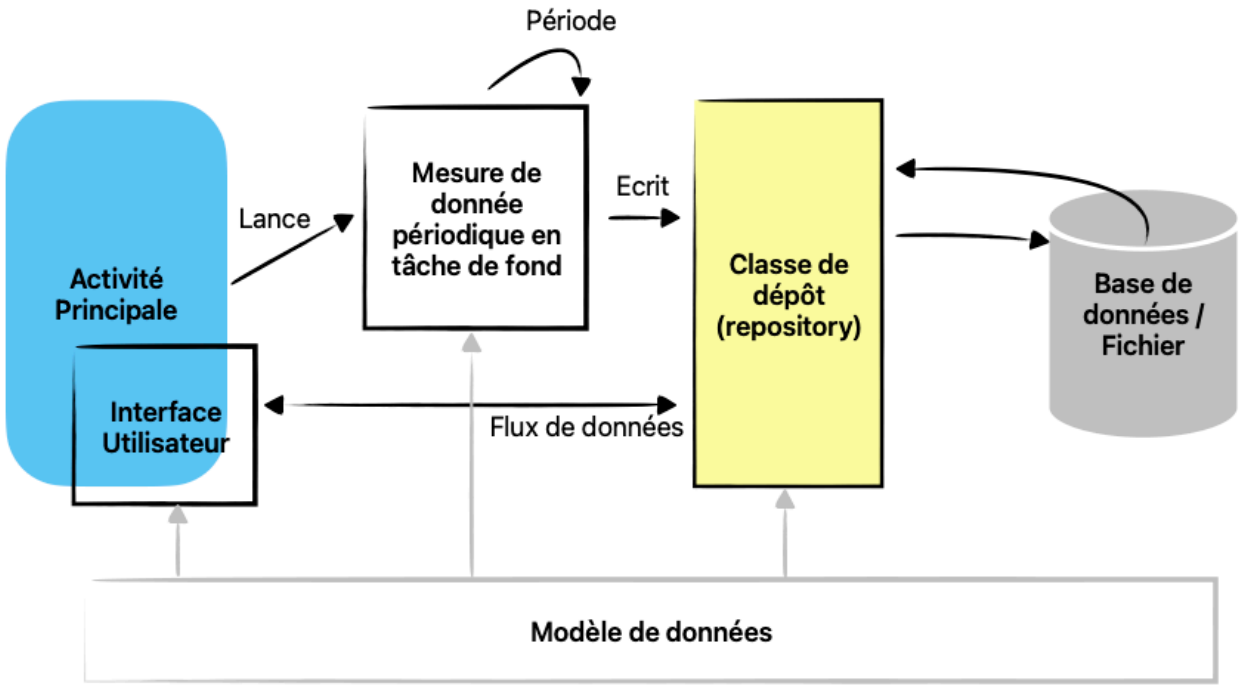
en déposant le tout sur un projet du gitlab de l'INP (https://hudson.inp-toulouse.fr/). A vous d'y créer un nouveau projet et d'y ajouter l'enseignant en tant que développeur pour qu'il puisse accéder aux ressources.

[Barème de notation]

- Présentation : 7 points
  - 4 points pour le contexte / description des objets utilisés,
  - 3 points pour la démo.
- Réponse aux questions des étudiants : 3 points
- Réponse aux questions de l'enseignant : 4 points
- Evaluation des rendus : 6 points (projet Android fonctionnel, qualité du code, documentation / commenté, référence/qualité des ressources).

Architecture générale d'une application de Crowdsourcing.

Voici l'architecture classique d'une application de crowdsourcing.



Architecture Android `Jetpack`

Il est proposé de travailler avec une architecture d'application moderne et recommandée par Android : https://developer.android.com/topic/architecture?hl=fr.

Cette architecture a pour objectif de créer des applications robustes et de haute qualité. Elle se fonde sur :

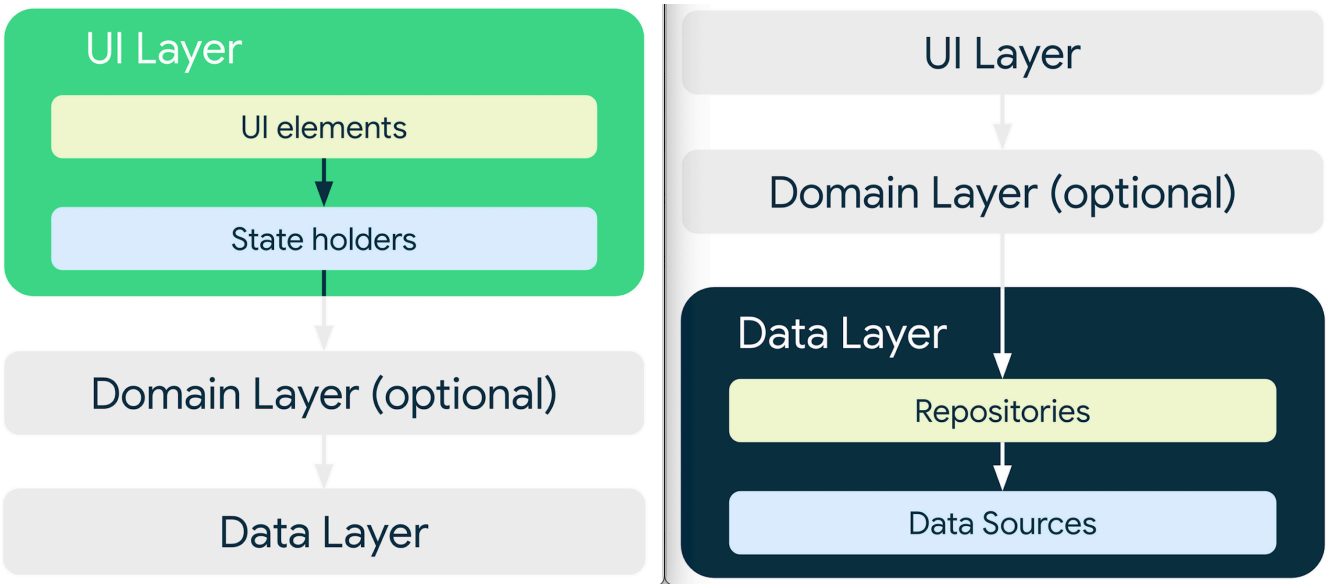
1. un principe modèle - vue - contrôleur. Ainsi, aucun traitement n'est réalisé dans la vue. En d'autres termes, on ne peut lancer de traitement d'une donnée dans une Activité ou un Fragment.
2. sur la définition d'une référence unique (Single source of truth) : une seule classe peut modifier les données et expose des fonctions ou intercepte des événements pour le faire. On parle ici de classe de dépôt (repository).

3. les flux de données sont bien définis : la donnée est lue par la source qui la pousse vers la vue (ici l'UI - User interface) et les événements utilisateurs sont transmis de l'IU vers la source de données.

Je vous invite à lire la partie suivante qui présente l'architecture recommandée par Android :

https://developer.android.com/topic/architecture?hl=fr#recommended-app-arch

Dans ce projet, nous allons principalement travailler avec la couche `UI Layer` et la couche `Data Layer`.



La couche Interface Utilisateur

cf. https://developer.android.com/topic/architecture?hl=fr#ui-layer

La couche IU Layer de décompose en deux parties qui regroupent :

- d'une part les composants graphiques de l'interface. Ces composant graphiques ont besoin de données pour adapter leur affichage.
- d'autre part de conteneurs d'états : ce sont des variables qui comportent les données nécessaires à l'affichage et dont le contenu est soit la conséquence d'une action de l'utilisateur (sauvegarde d'un message / d'un click) soit de données obtenues par la mise à jour de données externes à l'application (typiquement l'enregistrement de nouvelles données dans la base de données).

La couche Données

cf. https://developer.android.com/topic/architecture?hl=fr#data-layer

La couche Data Layer se décompose aussi en deux parties :

- d'une part les sources de données qui permettent d'enregistrer les données obtenues soit par l'action de l'utilisateur, soit issues de l'environnement (mesure de capteurs,..), soit du réseau.
- d'autre part de classes de dépôt (repositories). Il y a généralement une classe par type de donnée. Elles sont responsables des tâches suivantes :
  - Présenter les données au reste de l'application
  - Centraliser les modifications apportées aux données
  - Résoudre les conflits entre plusieurs sources de données
  - Extraire des sources de données du reste de l'application
  - Contenir la logique métier

Bibliothèques Android

L'application peut se construire autour des bibliothèques d'Android Jetpack, un ensemble récent de bibliothèques permettant de simplifier et de moderniser la création d'une application Android selon l'architecture introduite précédemment. Les principaux outils que l'on peut utiliser dans ce projet sont les suivants :

- La création de l'interface utilisateur avec Jetpack `Compose`, avec la définition du modèle de données associé.
  https://developer.android.com/jetpack/compose/layouts/basics?hl=fr.
- La création de conteneurs d'états pour rendre la mise à jour des composants graphiques dynamique (la mise à jour du conteneur d'état met automatiquement à jour l'affichage du composant graphique qui utilise la variable d'état associée).
  https://developer.android.com/topic/architecture/ui-layer?hl=fr#architecture
- L'automatisation de la mesure de données en tâche de fonds avec les classes [`Worker` & `WorkManager`]
  https://developer.android.com/topic/libraries/architecture/workmanager
- Le gestionnaire de base de données `Room` de type SQLite
  https://developer.android.com/training/data-storage/room
- La possibilité de générer des flux de données `Flow` qui permettent de mettre à jour les variables d'état automatiquement.
  https://developer.android.com/kotlin/flow?hl=fr

Différentes combinaisons de ces outils de développement permettent de générer des codes plus concis.
Il est possible par exemple, à chaque mise à jour d'une table de la base de données Room, de publier un message dans un flux. Ceci permet de mettre à jour de l'interface utilisateur si ce flux est lié à une variable d'état tel que décrit ici : https://medium.com/androiddevelopers/room-flow-273acffe5b57

Ces nouveaux outils sont naturellement développés pour le langage Kotlin (et non Java). Il vous faudra apprendre ce langage (AndroidStudio possède une fonctionnalité qui permet de traduire un code en Java en Kotlin automatiquement - Menu `Code` / `Convert Java File to Kotlin File`)

Documentation

La principale source de documentation, très riche, est le site https://developer.android.com.

Un ensemble d'exemple des code est fournit par Android pour développer des fonctionnalités avec cette nouvelle architecture. Ils sont particulièrement utiles, et je vous recommande de les utiliser dans votre travail.

https://github.com/android/platform-samples/tree/main

Un codelab pour apprendre a utiliser les contrôleurs d'états et les ViewModels
https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state#0

Ces exemples sont souvent utilisés dans les documentations du site https://developer.android.com.

Work Manager + Worker :
https://developer.android.com/topic/libraries/architecture/workmanager?hl=fr

---

Premiers exercices

Exercice 1 : Prise en main de Jetpack Compose.

Dans cet exercice, je vous invite à suivre le tutoriel pour apprendre à utiliser Jetpack Compose est disponible ici :
https://developer.android.com/jetpack/compose/tutorial?hl=fr

Ce tutoriel vous permet d'acquérir les bases de la création d'une interface utilisateur avec le langage de programmation Kotlin. Compose permet aussi la visualisation de l'UI dans AndroidStudio, sans passer par la

compilation et l'exécution sur un téléphone (réel ou virtuel).

Pour réaliser ce tutoriel, il est conseillé de travailler avec la dernière version d'Android Studio. Il faut créer une application en sélectionnant `Nouveau projet`, puis sous `Téléphone et tablette`, sélectionnez `Activité Compose vide`.

Exercice 2 : Une première interface

Cet exercice a pour objectif de vous faire créer une première application qui, à partir d'un bouton, vous fait afficher des données. Les données sont codées "en dur" dans un objet pour que vous puissiez simplement travailler à leur affichage.

Pour y arriver, il faut définir une variable qui est un conteneur d'état (uiState), initialisé avec ces données "en dur", dans une classe qui hérite de la classe `ViewModel`. Il faut pour cela définir un modèle des données définit dans une classe à part. On pourra considérer les classes de données suivantes qui permettent d'enregistrer une liste de données de localisation :

```
    data class LocationUiState(
        var timestamp: String,
        var latitude: Double,
        var longitude: Double
    )
    data class LocationListUi(
        val listUi: List<LocationUiState> = listOf()
    )
```

On pourra utiliser les données "en dur" sous le format suivant :

```
// Example dataset for initial tests : a list of LocationUiState
    object ExampleDataSet{
        private val ExampleLocationListUi = listOf<LocationUiState>(
            LocationUiState("10:01:02", 43.60229670390851, 1.4557916720712325),
            LocationUiState("10:01:30", 43.60229670342341, 1.4557916720712314),
            LocationUiState("10:02:00",43.60229670391235, 1.4557916720712312),
            LocationUiState("10:04:03", 43.60229670390453, 1.45579167207154351),
        )
        val ExampleList = LocationListUi(ExampleLocationListUi)
    }
```

Un tutoriel pour définir le modèle de données et définir comment interagir avec les données dans un conteneur d'états (`ViewModel`) est disponible ici :
https://developer.android.com/topic/architecture/ui-layer?hl=fr#architecture
Un second ici :
https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state#0

Comportement de l'application attendu

Au démarrage, l'activité (choisir une `Activité Compose vide`) affiche les données "en dur" à l'écran (sous la forme d'un texte). Il y a un bouton au début de l'écran qui permet, au click, de mettre à jour les données affichées au travers du conteneur d'état qui propose une fonction qui s'appelle 'refreshData()' en ajoutant un nouvel échantillon "en dur" à la variable d'état.

Exercice 3 : Collecte de la position du smartphone

La collecte de la position du smartphone est une requête fréquente dans les applications mobiles. Plusieurs méthodes, bibliothèques ont existé en Android. Actuellement, la bibliothèque la simple à utiliser pour la majorité des besoins est la `fused location provider` des Google Play services.

La position géographique est une donnée personnelle et son utilisation nécessite un ensemble de mécanismes de permissions. Ces mécanismes permettent à l'utilisateur d'être informé de leur utilisation et de donner son accord pour qu'elles soient utilisées de façon ponctuelle ou récurrente.
Ces demandes de permission sont documentées ici : permissions localisation(https://developer.android.com/training/location/permissions?hl=fr)

Un tutoriel sur cette bibliothèque est disponible ici :
https://developer.android.com/develop/sensors-and-location/location

Il est possible de demander deux types de positions :

- La dernière position qui a été mesurée par le smartphone (par une autre application) : `getLastLocation()` description(https://developer.android.com/training/location/request-updates?hl=fr#get-last-location)
- La position actuelle en faisant une requête au `fused location provider`. Cette requête a plusieurs paramètres, l'un étant par exemple la précision voulue de la position. Plus la précision est fine, plus la durée de traitement est possiblement longue (et donc énergivore). Cette requête est asynchrone, et il faut définir une méthode de callback pour qu'on puisse récupérer le résultat une fois qu'il est prêt. description(https://developer.android.com/training/location/request-updates?hl=fr#updates)

Comportement de l'application attendue

Cet exercice vous propose de créer une nouvelle application qui suite à l'appui sur un bouton, affiche la position actuelle de l'utilisateur. Vous pouvez la combiner aux fonctionnalités de l'application précédente pour utiliser cette mesure de localisation pour la mise à jour du conteneur d'états.

Voici un example de fonctions capables, dans un composant graphique, de récupérer la position actuelle du smartphone :
Exemple de code Jetpack - mesure de la position actuelle(https://github.com/android/platform-samples/tree/main/samples/location/src/main/java/com/example/platform/location/currentLocation)

Voici un example de fonction qui lance en tâche de fond la mesure de la position et l'écrit dans les logs
Exemple de code Jetpack - mesure de la position en tache de fond(
https://github.com/android/platform-samples/tree/main/samples/location/src/main/java/com/example/platform/location/bglocationaccess)