



Mémoire de fin d'études en vue de l'obtention du
diplôme de **Master**

Filière

Electronique

Spécialité

Systèmes Embarqués

- Thème -

Développement d'une Application Mobile pour Communiquer avec le Tableau de Bord d'un Scooter

Réalisé par:

BOUCHIBANE Amine

Soutenu le 25/06/2025

Membres du Jury :

NOM et Prénom	Établissement	Qualité
Mme. BOUCHAMA Samira	ENSTA	Présidente
Mme. CHOUAF Seloua	ENSTA	Encadrante
M. MAOUCHE Idris	FIBONOVA	Co-Encadrant
Mme. BELHACHAT Faiza	ENSTA	Examinatrice
M. BENZAOUI Mohamed Lassaad	USTHB	Examineur

Année Universitaire: 2024/2025

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

*Ce travail est humblement dédié à la mémoire de ma sœur
qui a toujours cru en moi.*

*Je dédie ce travail également à mes parents pour leurs
sacrifices et leur soutien sans faille. Leur confiance et leurs
prières m'ont accompagné à chaque étape, et c'est grâce à
eux que j'ai pu atteindre ce point dans ma vie.*

*À mon frère et mes sœurs, merci pour vos encouragements
constants et votre compréhension durant les périodes intenses
de ce travail. Votre soutien était d'une grande aide pour moi.
Je souhaite également exprimer ma profonde gratitude à tous
mes amis, pour leur accompagnement et leur encouragements
tout au long de ce parcours. Leur présence m'a offert la force
à dépasser les moments les plus exigeants.*

*Ce travail représente l'aboutissement de plusieurs années
d'études et l'achèvement de mon parcours académique. Il
symbolise la somme des connaissances acquises et la
transition vers une nouvelle étape de ma vie professionnelle.*

Amine

Remerciements

Je tiens à remercier toute personne m'ayant aidé durant cette période de réalisation de ce projet.

Tout d'abord, je tiens à remercier mon encadrante au niveau de l'école **Mme CHOUAF Seloua**, enseignante à l'ENSTA pour ses orientations méthodologiques, ses remarques constructives et son accompagnement tout au long de ce travail.

Je tiens également à remercier mon deuxième encadrant **M. MAOUCHE Idris**, ingénieur chez FIBONOVA, pour la confiance qu'il m'a accordée, sa disponibilité et les moyens qu'il a mis à ma disposition tout au long de mon stage.

Je remercie aussi l'équipe de l'entreprise FIBONOVA pour son accueil chaleureux, son soutien et ses conseils qui ont été d'une grande aide pour moi pour m'améliorer et atteindre des résultats satisfaisants pour ce projet.

Résumé

Ce projet porte sur le développement d'application Android qui sert de passerelle de communication avec un tableau de bord de scooter connecté. Elle établit une liaison Bluetooth robuste et effectue un échange de données bidirectionnel, permettant à la fois de collecter des informations du tableau de bord et de lui transmettre des données provenant du smartphone.

Mots clés: Android Studio, Kotlin, Bluetooth, notifications

Abstract

This project focuses on the development of an Android application that serves as a communication gateway with a connected scooter dashboard. It establishes a robust Bluetooth connection and performs a bidirectional data exchange, allowing it to both collect information from the dashboard and transmit data from the smartphone to it.

Keywords: Android Studio, Kotlin, Bluetooth, notifications

ملخص

يركز هذا المشروع على تطوير تطبيق أندرويد يعمل كجوة اتصال مع لوحة قيادة سكوتر متصلة. يقوم التطبيق بإنشاء اتصال بلوتوث ويجري تبادلاً ثنائياً للاتجاه للبيانات، مما يسمح له بجمع المعلومات من لوحة القيادة وإرسال البيانات من الهاتف الذكي إليها.

كلمات مفتاحية : أندرويد ستوديو، لغة كوتلن، بلوتوث، إشعارات

Table de matières

Dédicace	I
Remerciements	II
Résumé	III
Liste des Acronymes	VII
Liste des Figures	VIII
Liste des Tableaux	VIII
Introduction générale	1
Chapitre 1 Les applications mobiles pour systèmes connectés	2
1.1 Introduction	2
1.2 Applications Android pour la communication sans fil	2
1.3 Etapes de développement d’une application connectée	3
1.4 Architecture d’une application Android pour systèmes IoT	4
1.4.1 Module d’interface utilisateur	4
1.4.2 Module de gestion de la connectivité	4
1.4.3 Module de traitement et de stockage des données	5
1.4.4 Module de services en arrière-plan	5

1.5	Contraintes et défis	5
1.6	Conclusion	6
Chapitre 2 Application Android connectée au tableau de bord		7
2.1	Introduction	7
2.2	Description de l'application	7
2.2.1	Objectifs de l'application développée	7
2.2.2	Contraintes techniques	8
2.3	Android studio	8
2.4	Développement de l'application	9
2.4.1	Initialisation du projet sur Android studio	10
2.4.2	Création de l'interface utilisateur	11
2.4.3	Gestion du Bluetooth	11
2.4.4	Données associées au smartphone	14
2.4.5	Données de navigation	15
2.4.6	Gestion des appels	15
2.5	Échange de données	16
2.5.1	Méthode de file d'attente	16
2.5.2	Lecture et écriture de données	17
2.6	Test et validation de l'application	18
2.7	Conclusion	20
Conclusion générale		21

Références bibliographiques	IX
-----------------------------	----

Annexes	X
---------	---

Liste des Acronymes

API: Application Programming Interface.

BLE: Bluetooth Low Energy.

CPU: Central Processing Unit

FIFO: First-In First-Out

GATT: Generic Attribute Profile.

IDE: Integrated Development Environment.

IoT: Internet of Things.

NFC: Near Field Communication

RAM: Random Access Memory

RGPD: Règlement Général sur la Protection des Données.

UI: User Interface.

UUID: Universally Unique Identifier

WiFi: Wireless Fidelity

XML: eXtensible Markup Language.

Liste des Figures

Fig. 2.1	Interface de démarrage et interface principale de l'application	10
Fig. 2.2	Permission pour activer le Bluetooth et la localisation	18
Fig. 2.3	Réception d'un appel	19
Fig. 2.4	Communication avec un tableau de bord	20

Liste des Tableaux

Tab. 2.1	Services et caractéristiques BLE pour le tableau de bord . . .	13
----------	--	----

Introduction générale

L'émergence de l'Internet of Things (IoT) a changé la façon d'interagir avec le monde physique, en incluant un nombre important d'objets dans un environnement rendu intelligent par des interconnexions généralisées à tous ses composants. Au cœur de ces objets connectés, les applications Android sont désormais perçues comme des interfaces où l'utilisateur interagit sur une grande diversité de dispositifs de l'IoT, qu'il s'agisse des systèmes domotiques ou des objets portables connectés. Le smartphone Android, en tant que plateforme mobile omniprésente, est utilisé comme passerelle intuitive pour prendre le contrôle d'une multitude de dispositifs IoT, tout en permettant de visualiser les différentes données en temps réel.

Le présent projet vise la réalisation d'une application Android capable de communiquer avec un tableau de bord connecté qu'elle ne se contente pas de relier, mais sur les données de laquelle elle navigue pour enrichir la conduite, le smartphone se transformant ainsi, au fil du temps, en centre de commande et d'information du pilote lui-même. La technologie Bluetooth est utilisée pour établir une connexion fluide et constante entre ces deux dispositifs. Ce document s'articule autour de deux chapitres essentiels :

Un chapitre porte sur l'intégration des applications Android dans le domaine de l'IoT, ainsi que sur leur architecture et les contraintes liées au développement de telles applications.

Un autre chapitre qui détaille les fonctionnalités à intégrer dans notre application et explique le processus de son développement, de l'établissement d'une connexion Bluetooth Low Energy (BLE) jusqu'à effectuer un échange de données avec un tableau de bord connecté. Il porte également sur la phase de test de cette application pour vérifier ses performances et son bon fonctionnement.

Chapitre 1

Les applications mobiles pour systèmes connectés

1.1 Introduction

De nos jours, l'interface utilisateur évolue du simple affichage analogique vers l'intégration de fonctionnalités intelligentes et connectées. Les systèmes embarqués sur les tableaux de bord offrent un potentiel bien au-delà de l'analogique. L'IoT permet désormais aux dispositifs d'échanger des données en temps réel, ouvrant la voie à des interactions dynamiques et automatisées. Dans ce contexte, les applications mobiles, notamment sur Android, jouent un rôle essentiel en facilitant l'interfaçage et le contrôle des systèmes connectés.

1.2 Applications Android pour la communication sans fil

Avec l'essor de l'IoT, le smartphone devient un hub central, collectant, stockant et analysant des données issues de multiples objets connectés. Cette position privilégiée est particulièrement mise en avant sur la plateforme Android, qui s'impose comme un choix incontournable pour le développement d'applications IoT. Grâce à sa nature ouverte et flexible, sa vaste communauté de développeurs et son expérience approfondie dans le domaine, Android offre un cadre idéal pour la création et l'optimisation de ces applications.

Pour connecter le smartphone aux objets intelligents, les applications Android exploitent plusieurs technologies de communication sans fil. Parmi elles, le BLE joue un rôle central dans l'écosystème IoT grâce à sa faible consommation énergétique et sa capacité à établir des connexions sur de courtes distances, garantissant ainsi une autonomie prolongée pour les dispositifs connectés. En complément du BLE, d'autres technologies comme le Wireless Fidelity (Wi-Fi) et le Near Field Communication (NFC) sont utilisées en fonction des besoins spécifiques en connectivité et en bande passante des différents scénarios IoT.

1.3 Etapes de développement d'une application connectée

Concevoir et développer une application Android pour les systèmes connectés nécessite une démarche méthodique dans le but de répondre efficacement aux exigences posées. Ce processus se décompose en plusieurs étapes clés [1], allant de la compréhension approfondie des besoins jusqu'à l'écriture du code, en passant par la définition de l'architecture.

a. Analyse des besoins

Cette phase est la base de tout développement logiciel réussi. Il s'agit d'identifier précisément les besoins fonctionnels et doit intégrer les contraintes utilisateur. Cette démarche implique souvent des échanges avec les parties prenantes pour s'assurer que toutes les attentes sont bien comprises et documentées avant de passer aux étapes de conception et de développement.

b. Conception architecturale

Une fois les besoins sont définis, la phase de conception architecturale agit pour structurer l'application de manière logique et évolutive. Il s'agit de décomposer l'application en modules principaux, chacun ayant une responsabilité claire et bien définie.

c. Implémentation

L'étape d'implémentation met en œuvre la conception architecturale, en écrivant le code de l'application. Pour le développement Android, les langages de programmation privilégiés sont Kotlin et Java. Kotlin est de plus en plus populaire et recommandé par Google¹ pour plusieurs facteurs, vu qu'il offre une syntaxe plus expressive que Java, ce qui réduit la quantité de code à écrire et diminue les risques d'erreurs. Il est également entièrement interopérable avec Java, permettant aux développeurs d'utiliser des bibliothèques Java existantes et de migrer leurs projets progressivement. Le choix entre les deux dépendra souvent de la familiarité de l'équipe et des spécificités du projet.

1.4 Architecture d'une application Android pour systèmes IoT

Afin de construire une bonne application Android, une architecture modulaire est essentielle car elle permet de distinguer les responsabilités pour faciliter le développement, les tests, et la maintenance. Typiquement, une telle application est structurée autour de plusieurs modules clés, chacun gère un aspect spécifique de la communication et de l'interaction.

1.4.1 Module d'interface utilisateur

La conception du module d'interface utilisateur représente le point de contact direct entre l'utilisateur et le système connecté. L'expérience utilisateur doit être conçue pour le contexte d'utilisation, notamment la simplicité et la lisibilité en mouvement.

1.4.2 Module de gestion de la connectivité

Le module de gestion de la connectivité sert à gérer l'intégralité du cycle de vie des connexions sans fil, depuis la phase de "scanning" pour découvrir les périphériques disponibles à proximité, jusqu'à l'établissement d'une connexion. Ce module doit également anticiper et gérer les scénarios de déconnexion et

¹Google, identifié via : <https://about.google>

mettre en œuvre des stratégies de reconnexion efficaces pour assurer la continuité du service.

1.4.3 Module de traitement et de stockage des données

Ce module transforme les informations reçues des dispositifs IoT en un format exploitable par l'application. Il inclut le parsing des données, un processus permettant d'interpréter les octets reçus afin de les convertir en types de données compréhensibles (nombres, chaînes de caractères, structures complexes). Une fois analysées, ces données peuvent nécessiter une agrégation ou une transformation avant d'être prêtes pour l'affichage, une analyse approfondie ou leur stockage permanent.

1.4.4 Module de services en arrière-plan

Ce module gère les opérations indépendantes de l'interface utilisateur, telles que la maintenance d'une connexion Bluetooth, la réception de notifications critiques ou la collecte continue de données. Sur Android, la gestion de ces services doit être optimisée en tenant compte des restrictions de plus en plus strictes du système d'exploitation, afin de garantir une consommation d'énergie réduite et une stabilité accrue du système.

1.5 Contraintes et défis

Le développement d'applications Android pour les systèmes IoT présente de nombreux défis et contraintes qui, s'ils ne sont pas gérés avec rigueur, peuvent compromettre l'expérience utilisateur de l'application finale. Ces difficultés découlent principalement de la nature distribuée de l'IoT et de la diversité de l'écosystème Android. Les principaux défis et limitations liés au développement d'applications Android pour l'IoT sont :

- La gestion de la compatibilité avec diverses versions d'Android (API levels).
- L'adaptation à la multitude de modèles de smartphones, résolutions d'écran et chipsets Bluetooth variés.
- L'optimisation de la consommation de la batterie du smartphone (Blue-

tooth, localisation).

- La gestion efficace de l'utilisation de la mémoire RAM et du CPU.
- La protection des données utilisateur (localisation, préférences) échangées et stockées.
- La conformité aux réglementations sur la protection des données personnelles (tels que le Règlement Général sur la Protection des Données (RGPD)).
- La fiabilité et stabilité de la connexion BLE, y compris la gestion des interférences et des reconnections.
- La conformité aux restrictions d'Android sur les opérations en arrière-plan.

1.6 Conclusion

En conclusion, Ce chapitre a pris sur les applications mobiles Android au sein du domaine de l'IoT, En explorant les principes fondamentaux de leur fonctionnement, ainsi que les mécanismes de communication sans fil intégrés. La décomposition du développement en plusieurs étapes a mis en évidence la complexité de la création de ces applications.

Tous les principes et défis abordés peuvent être une base solide pour la conception et la mise en œuvre d'une application Android spécifique, destinée à un système connecté.

Chapitre 2

Application Android connectée au tableau de bord

2.1 Introduction

Les applications mobiles pour les systèmes connectés permettent de commander et de surveiller un environnement, qu'il soit de domotique ou de véhicules. Par exemple, une application dédiée à un tableau de bord connecté offre une meilleure interaction avec le véhicule à l'aide de fonctionnalités comme la gestion de la navigation. La conception d'une telle application exige une rigueur particulière pour garantir une fiabilité de la communication ainsi qu'une continuité fluide entre les mondes physique et numérique sans jamais compromettre l'expérience de l'utilisateur.

2.2 Description de l'application

L'application Android est le compagnon numérique d'un tableau de bord connecté pour scooter. Elle est conçue pour apporter une réponse concrète et efficace aux besoins formulés.

2.2.1 Objectifs de l'application développée

Cette application est conçue pour améliorer l'expérience de conduite et établir une communication Bluetooth avec le tableau de bord. En effectuant un échange de données spécifiques qui comprennent :

- Les données de navigation (distance et temps restants avant l'arrivée, indications de direction, etc.).

- Données associées au smartphone (nom de l'appareil et niveau de batterie).
- Données liées à la réception des appels (Nom de l'appelant, durée de l'appel, etc.).
- Des informations provenant du tableau de bord (vitesse de déplacement du véhicule, kilométrage parcouru, etc.).

L'application doit également être capable d'activer et de désactiver le Bluetooth du smartphone et la localisation, ainsi que de surveiller le smartphone pour un appel entrant ou un changement dans son niveau de batterie ou encore attendre des notifications d'itinéraire provenant de l'application Google Maps pour garantir la mise à jour des données envoyées au tableau de bord. Alors que cette application est conçue pour des tâches bien précises, elle doit satisfaire des contraintes logicielles liées à l'expérience utilisateur sur le smartphone.

2.2.2 Contraintes techniques

Pour garantir que cette application atteigne pleinement son potentiel, son développement doit prendre en compte plusieurs contraintes techniques essentielles :

- Compatibilité avec les différentes versions d'Android.
- Adaptabilité à la diversité des appareils (tailles d'écran, résolutions, chipsets Bluetooth variés).
- Conception d'une interface utilisateur intuitive, fluide et facile à utiliser.
- Maintien d'une connexion BLE stable et fiable, avec une gestion efficace des reconnections.

D'autres contraintes doivent également être intégrées pour assurer une application performante et optimisée, adaptée aux exigences du système et aux attentes des utilisateurs.

2.3 Android studio

Android Studio est l'environnement de développement intégré (IDE) officiel pour le développement des applications Android, approuvé par Google. Il offre

plusieurs outils et fonctionnalités conçus pour le processus de développement. Avec un éditeur de code puissant, Android Studio offre une expérience fluide aux développeurs, en prenant en charge les dernières Application Programming Interfaces (APIs) et bibliothèques Android. Ces capacités font de lui le meilleur IDE pour créer des applications Android de haute qualité et efficaces[2].

Pour cette application, on a utilisé une version stable d'Android Studio, nommée "Hedgehog", lancée en novembre 2023[3]. Cette version apporte des performances améliorées, de nouvelles fonctionnalités et une stabilité accrue.

2.4 Développement de l'application

Pour développer notre application Android, nous devons d'abord définir les fonctionnalités à intégrer, afin de garantir une expérience utilisateur fluide. L'objectif principal est de permettre l'échange de données avec un tableau de bord pour scooter en utilisant le BLE à l'aide d'un appareil Android. Les fonctionnalités clés identifiées sont les suivantes :

- Un bouton pour le scan et la connexion au tableau de bord.
- Un bouton pour se déconnecter.
- Une zone de texte pour l'état du smartphone (connecté, déconnecté, en attente).
- Deux zones de texte superposées une pour la vitesse et une autre pour le kilométrage.

La figure 2.1 se trouvant sur la page suivante illustre les deux interfaces développées pour cette application. L'une de ces interfaces contient les fonctionnalités citées (boutons, zones de texte, etc.).

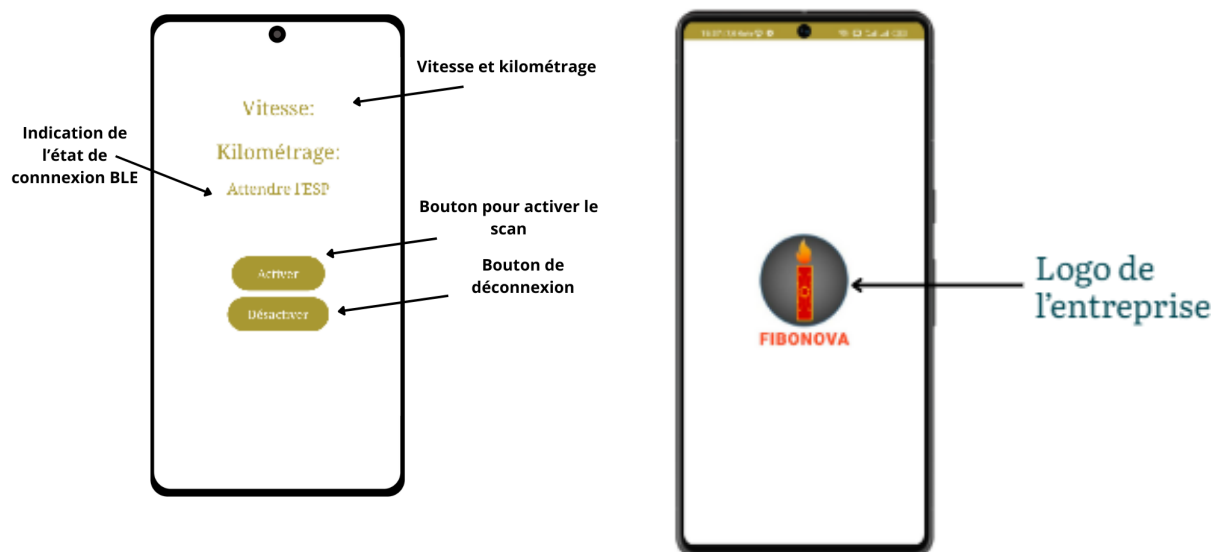


Figure 2.1: Interface de démarrage et interface principale de l'application

2.4.1 Initialisation du projet sur Android studio

Lors de la création d'un projet sur Android Studio Hedgehog, on commence par sélectionner "Empty Views Activity" pour obtenir un modèle vierge. L'application est développée en Kotlin, avec une compatibilité définie entre l'API 24 (Android Nougat) et l'API 34 (Android 14).

Après l'initialisation du projet, on retrouve trois dossiers essentiels, structurés comme suit :

a. manifests

Contient le fichier "AndroidManifest.xml", qui déclare les composants de l'application ainsi que les permissions requises (comme l'accès au Bluetooth et à la localisation). Ce fichier informe le système Android sur la structure et les besoins de l'application avant son lancement.

b. kotlin+java

Ce dossier regroupe le code source de l'application, écrit en Kotlin ou Java. Chaque dossier structure les classes en groupes logiques. Le choix entre les deux langages dépend généralement de l'héritage du projet ou des compétences de l'équipe de développement.

c. res

Ce dossier rassemble toutes les ressources non liées au code nécessaires à l'interface utilisateur et au bon fonctionnement de l'application. On y trouve :

- Les layouts (fichiers XML définissant l'interface utilisateur).
- Les drawables (images, icônes).
- Les valeurs (textes, couleurs, dimensions)[1].

2.4.2 Création de l'interface utilisateur

La conception de l'interface utilisateur (UI) repose sur un fichier XML, qui définit les éléments visuels et les interactions, séparément du code source (voir figure 2.2). Cette interface est conçue pour être intuitive, accessible et facile à utiliser, garantissant une expérience utilisateur fluide et compréhensible.

L'application contient deux interfaces utilisateur. La première s'affiche une fois que l'application est lancée et disparaît après 3 secondes. Elle contient un logo de l'entreprise FIBONOVA¹. Tandis que la deuxième est l'interface principale de l'application.

2.4.3 Gestion du Bluetooth

La gestion du Bluetooth est cruciale pour la connectivité avec le tableau de bord. Elle inclue la découverte des périphériques BLE émettant, l'établissement de la connexion et l'interaction avec les services GATT du tableau de bord. L'application agit comme un client BLE qui effectue une lecture des données en temps réel (vitesse, kilométrage) via des requêtes périodiques, ainsi qu'une écriture vers le tableau de bord. Cette communication bidirectionnelle garantit une synchronisation constante entre le smartphone et le tableau de bord.

a. Activation et désactivation

L'application Android offre un contrôle intuitif sur l'activation et la désactivation de la fonctionnalité Bluetooth sur le smartphone. Dès son lancement, elle vérifie si le Bluetooth est actif sur le téléphone. Si ce n'est pas le cas, elle invite l'utilisateur à l'activer, soit directement via une boîte de dialogue système.

¹FIBONOVA, identifiée via : <https://www.fibonovaalgeria.com>

Une fois activé, l'application peut initier le processus de scan BLE pour découvrir le tableau de bord. Inversement, l'utilisateur peut désactiver le Bluetooth via l'application elle-même pour interrompre la connexion active et stopper la recherche de périphériques. L'application utilise également la localisation pour initier le scan BLE ; pour cela, il est nécessaire de pouvoir l'activer aussi.

Une fonction vérifie l'état du Bluetooth du smartphone. S'il est activé, l'application continue son exécution, sinon une boîte de dialogue permet d'activer le Bluetooth.

Listing 2.1: Activation du Bluetooth

```
private fun ensureBluetoothEnabled() {  
    val bluetoothAdapter = BluetoothAdapter.getDefaultAdapter()  
    if (bluetoothAdapter == null) {  
        // l'appareil ne supporte pas le Bluetooth  
        Toast.makeText(this, "Bluetooth n'est pas supporté",  
            Toast.LENGTH_LONG).show()  
        finish()  
    } else if (!bluetoothAdapter.isEnabled) {  
        val enableBtIntent = Intent(BluetoothAdapter.  
            ACTION_REQUEST_ENABLE)  
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT)  
    }  
}
```

C'est avec le même principe qu'une autre fonction vérifie l'état de la localisation. Si elle est activée, l'application continue son exécution, sinon une boîte de dialogue dirige l'utilisateur vers les paramètres de localisation pour l'activer. Ces deux fonctions sont déclarées ensuite dans la fonction **OnCreate** pour être exécutées une fois l'application lancée.

b. Services et caractéristiques utilisés

L'échange des données se fait à partir des services définis par des UUID et qui contiennent des caractéristiques. Un service peut contenir plusieurs caractéristiques à la fois si elles appartiennent au même champ d'informations. Le tableau 2.1 liste les services et caractéristiques utilisés ainsi que leurs UUIDs.

Les UUIDs cités dans le tableau 2.1 sont à 16 bits, mais l'API du smartphone gère des UUIDs à 128 bits. Pour cela, on doit les compléter avec un UUID

Table 2.1: Services et caractéristiques BLE pour le tableau de bord

Service	UUID	Caractéristique	UUID
Infos sur l'appareil	0x180A	Nom de l'Appareil	0x2A00
Batterie	0x180F	Niveau de batterie	0x2A19
Notifications et alertes	0x1811	Appels	0x2A46
Données de Scooter	0x1234	Vitesse	0x5678
		Kilométrage	0xABCD
Données de navigation	0x0011	Instructions	0x0033
		Distance au prochain tour	0x0022
		Distance restante	0x0044
		Temps restant	0x0055
		Heure d'arrivée	0x0066

Bluetooth de base qui est 0x0000XXXX-0000-1000-8000-00805F9B34FB [4] en remplaçant les X par notre UUID 16 bits. Par exemple, si le UUID 16 bits du service de batterie est 0x180F, son UUID à 128 bits est 0x0000180F-0000-1000-8000-00805F9B34FB.

c. Gestion de connexion

Une fois connecté, le smartphone initialise la découverte des services du tableau de bord. Ensuite, il procède à la découverte des caractéristiques associées à chacun de ces services. Cet échange structuré via le profil GATT est fondamental pour le bon fonctionnement et la pertinence du tableau de bord connecté.

d. Gestion de la déconnexion

La gestion de déconnexion couvre à la fois les déconnexions volontaires, initiées par l'utilisateur ou l'application elle-même, et les déconnexions imprévues, causées par une perte de signal ou un redémarrage du périphérique. L'application est capable de détecter ces ruptures grâce aux rappels système. En cas de déconnexion inattendue, une stratégie de reconnexion automatique est mise en œuvre, où l'application tente de rétablir la liaison.

Le deuxième bouton permet de se déconnecter d'une liaison BLE en utilisant la fonction prédéfinie **disconnect**, et aussi ferme la section GATT actuelle pour permettre à une autre de s'établir.

Listing 2.2: Déconnexion d'une liaison BLE

```

override fun onCreate() { bouton2.setOnClickListener {
    bluetoothGatt?.apply {
        disconnect()
        close()
        bluetoothGatt = null } }
}

```

2.4.4 Données associées au smartphone

Pour récupérer le nom de l'adaptateur Bluetooth local du smartphone dans cette application, on utilise la commande **bluetoothAdapter.name**. Le `bluetoothAdapter` est une instance de la classe `BluetoothAdapter` qui représente l'adaptateur Bluetooth propre au smartphone, et le ".name" est propriété en lecture seule de l'objet `BluetoothAdapter` qui renvoie le nom de l'appareil Android tel qu'il est configuré par l'utilisateur ou par défaut dans les paramètres Bluetooth du système.

Pour récupérer le niveau de batterie en temps réel, on utilise un `BroadcastReceiver`, qui est un mécanisme efficace dans l'Android pour l'obtention de cette information en temps réel. Ici, ce "Receiver" est configuré pour écouter les événements liés à la batterie. On récupère le niveau de batterie et l'échelle (généralement 100) pour calculer le pourcentage en divisant le niveau par l'échelle, puis en multipliant par 100. Une précaution est prise pour éviter une division par zéro si l'échelle est nul.

Listing 2.3: Niveau de batterie

```

var batteryPct = 100 //niveau de batterie initié à 100
private val nBatteryInfoReceiver: BroadcastReceiver = object :
    BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) {
            val level = intent.getIntExtra(BatteryManager.
                EXTRA_LEVEL, -1) //niveau de batterie
            val scale = intent.getIntExtra(BatteryManager.
                EXTRA_SCALE, -1) //échelle

            // Calcul du pourcentage
            batteryPct = if (scale != 0) (level * 100) / scale else
                0 }
}

```

2.4.5 Données de navigation

La récupération des données de navigation provenant de Google Maps est gérée au sein de l'application Android par un **NotificationListenerService**. Ce service spécialisé, s'exécutant en arrière-plan, permet à l'application de capter et d'analyser les notifications générées par Google Maps sur le smartphone, avec une permission spécifique accordée par l'utilisateur. En surveillant le contenu (titre, texte, extras) des notifications émanant d'applications de navigation, le **NotificationListenerService** extrait des informations de navigation. Une fois ces données de navigation obtenues, elles sont ensuite transmises au main principal de l'application par un autre Receiver qui capte les changements dans le Listener des notifications et extrait les informations nécessaires depuis ce Listener. En définissant des variables pour chaque information, on leur assigne les nouvelles instructions liées à l'itinéraire pour pouvoir les envoyer par BLE ou les afficher sur l'application.

2.4.6 Gestion des appels

La gestion des appels entrants sur l'application Android est assurée par un autre **BroadcastReceiver** spécifiquement configuré pour écouter les changements d'état du téléphone. En s'abonnant à l'action système de l'état du téléphone, l'application est informée des différentes phases d'un appel (sonnerie, appel en cours, ou inactivité). Lorsqu'un appel est détecté en phase de sonnerie, le BroadcastReceiver peut extraire des informations essentielles telles que le numéro de téléphone de l'appelant. Avec les permissions à l'application d'accéder à la liste des contacts du téléphone, il est possible de récupérer le nom de l'appelant depuis les contacts du téléphone. Un autre aspect est d'établir un compteur pour la durée de l'appel qui termine si l'appel est terminé et renvoie la durée totale de l'appel en minutes et secondes. Ces informations sont ensuite traitées par l'application pour être transmises via BLE au tableau de bord, permettant ainsi d'afficher instantanément le nom de l'appelant et la durée de l'appel sur l'écran du scooter.

Listing 2.4: Réception des appels

```
private fun onAnswered() {
```

```

callTimerJob?.cancel() //annuler toute tâche de chronomètre
précédente
callStartTime = System.currentTimeMillis() // initialiser le
temps de début de l'appel
isCallActive = true

callTimerJob = coroutineScope.launch { //exécuter un comtage
en arrière plan
    var seconds = 0
    while (isActive) {
        val timeFormatted = String.format("%02d:%02d",
            seconds / 60, seconds % 60) //incrémenter le
compteur chaque seconde
        delay(1000)
        seconds++
    }
}
}

```

2.5 Échange de données

L'objectif principal de cette application est de pouvoir échanger des données avec le tableau de bord à travers les services présentés auparavant. Cet échange est souvent géré de manière asynchrone. Cependant, cette nature asynchrone possède une contrainte majeure dans la plupart des piles BLE qui est de ne permettre l'exécution que d'une seule opération BLE à la fois sur une même connexion. Tenter d'envoyer plusieurs requêtes (comme pour notre cas) simultanément entraîne l'échec de l'opération. C'est pourquoi une gestion séquentielle via une file d'attente est utilisée pour organiser ces opérations et assurer leur traitement ordonné et fiable.

2.5.1 Méthode de file d'attente

Une file d'attente (queuing) est une structure de données linéaire qui fonctionne selon le principe du First-In, First-Out (FIFO) où les nouveaux éléments sont toujours ajoutés à une extrémité et les éléments sont toujours retirés de l'autre extrémité. L'implémentation d'une file d'attente garantit un traitement des opérations BLE de manière ordonnée et non concurrente [5].

Deux fonctions **enqueueOperation** et **ProcessNext** constituent le cœur

du système de gestion des opérations BLE via une file d'attente. Lorsqu'une nouvelle opération BLE doit être exécutée, elle est d'abord ajoutée à la liste d'attente. Immédiatement après l'ajout, une vérification est effectuée pour savoir si le processeur de la file d'attente n'est pas déjà actif, la fonction **processNext**) est appelée pour démarrer le traitement. Cette fonction est chargée de récupérer et d'exécuter la prochaine opération de la file. Si la file d'attente est vide, cela signifie que toutes les opérations en attente ont été traitées ; un drapeau est alors mis à faux pour signaler que le processeur est inactif et n'a plus rien à faire.

2.5.2 Lecture et écriture de données

Pour distinguer entre une opération de lecture et d'écriture, on a conçu une classe intitulée **bleOperation** qui a pour but d'insérer l'UUID du service et de la caractéristique à interroger.

Listing 2.5: Classes de lecture et écriture en BLE

```
sealed class BleOperation {  
    data class Read(val serviceUUID: UUID, val charUUID: UUID):  
        BleOperation()  
    data class Write(val serviceUUID: UUID, val charUUID: UUID, val  
        value : Any):BleOperation()  
}
```

Par exemple, pour envoyer au tableau de bord le niveau de batterie par le service ayant l'UUID 0x180F et la caractéristique 0x2A19, on insère ces UUID avec la valeur du niveau de batterie comme suit:

Listing 2.6: service et caractéristique de niveau de batterie

```
private val BATTERY_SERVICE_UUID = UUID.fromString("0000180F  
-0000-1000-8000-00805F9B34FB")  
private val BATTERY_CHARACTERISTIC_UUID = UUID.fromString("00002  
A19-0000-1000-8000-00805F9B34FB")  
  
enqueueOperation(BleOperation.Write(BATTERY_SERVICE_UUID,  
    BATTERY_CHARACTERISTIC_UUID, batteryPct))
```

Le code ci-dessus définit une opération d'écriture sur le service et la caractéristique de la batterie avec la valeur du niveau de batterie "batteryPct", puis ajoute cette opération à la file d'attente. Afin d'établir un échange périodique

des données, les opérations sur la file d'attente sont définies dans une boucle qui démarre une fois que le smartphone est connecté et se poursuit jusqu'à sa déconnexion, avec un retard d'une seconde entre le traitement de toute la file d'attente et un nouveau lancement.

2.6 Test et validation de l'application

Après les phases de conception et de développement de l'application, la validation est une étape indispensable pour garantir son bon fonctionnement et sa fiabilité en conditions réelles. Cette section détaillera les procédures de test mises en œuvre, visant à vérifier non seulement la connectivité BLE avec le tableau de bord, mais aussi les données échangées et la réactivité de l'interface utilisateur.

Lorsque l'application est lancée, elle demande la permission pour activer le Bluetooth du smartphone et aussi accéder aux paramètres pour activer la localisation (figure 2.2).

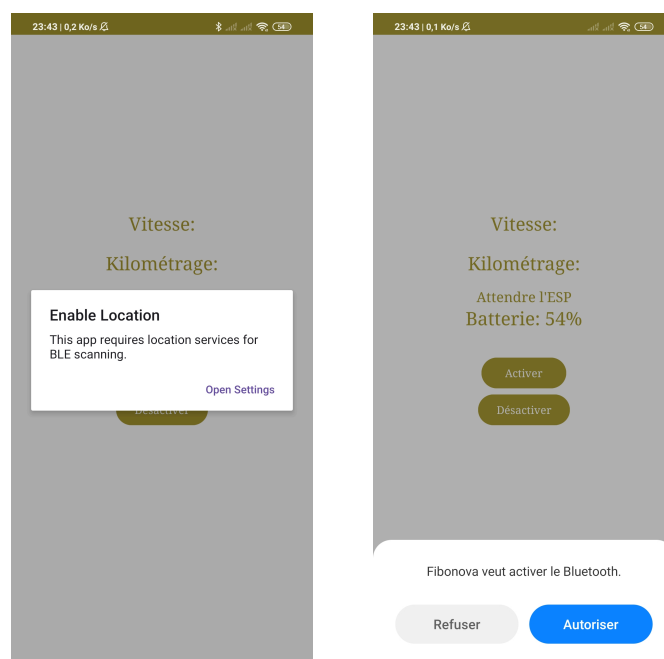


Figure 2.2: Permission pour activer le Bluetooth et la localisation

Le premier test concerne le niveau de batterie ainsi que le récepteur d'appels. On effectue un appel depuis un autre smartphone pour tester le nom du contact récupéré si c'est le nom du contact affiché sur la notification d'appel. La figure 2.3 montre comment l'application a pu récupérer le niveau de batterie du

smartphone et le nom de l'appelant qui apparaît en haut.

Sur la figure 2.3, le nom sur la zone du texte des appels convient au nom de l'appelant sur l'appel qui est "Amine", et le niveau de batterie en haut du téléphone est affiché sur sa zone de texte dans l'application (53 %).

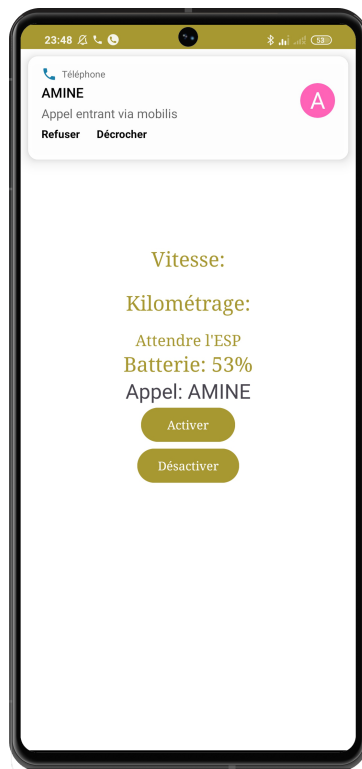


Figure 2.3: Réception d'un appel

Pour la communication avec le tableau de bord, comme illustré sur la figure 2.4 se trouvant sur la page suivante, si on appuie sur le bouton d'activation du scan, le texte affiché sur l'application lorsqu'elle se connecte indique la réussite de la connexion et les données de navigation seront envoyées au tableau de bord.



Figure 2.4: Communication avec un tableau de bord

2.7 Conclusion

En conclusion, ce chapitre a détaillé le processus de développement de l'application Android dédiée à la communication avec un tableau de bord connecté. En explorant les choix technologiques, notamment l'adoption de Kotlin et de l'environnement Android Studio, la mise en œuvre de la gestion BLE pour la découverte, la connexion, et l'échange de données a été l'objectif principal à atteindre, en insistant sur l'importance de l'approche asynchrone et l'utilisation d'une file d'attente pour sérialiser les opérations. La récupération et le traitement des données essentielles comme le niveau de batterie, les informations de navigation, et la gestion des appels, via des mécanismes dédiés comme les `BroadcastReceiver` et le `NotificationListenerService`, ont été présentés pour montrer la capacité de l'application à servir comme passerelle intelligente pour le tableau de bord connecté.

Conclusion générale

En conclusion, ce projet a mené à la conception et au développement d'une application Android pour la connectivité avec un tableau de bord de motorcycle. Le choix s'est porté sur Kotlin et l'environnement robuste d'Android Studio, des outils qui facilitent le développement et garantissent une fiabilité et une optimisation du code de l'application.

A l'intérieur de cette application réside une gestion du BLE comme technologie de communication choisie. Nous avons détaillé chaque étape, de la découverte des périphériques et l'établissement de la connexion à l'échange de données bidirectionnel. La nature asynchrone des opérations BLE a permis l'implémentation d'une file d'attente. Cette stratégie a été essentielle pour assurer une communication stable sous un flux continu de données, tout en maintenant la réactivité de l'interface utilisateur.

La prise en charge multilingue qui permet à l'utilisateur de choisir la langue de l'interface rendrait l'application accessible à un public mondial. Aussi, l'amélioration de l'interface utilisateur est essentielle. Cela pourrait se traduire par une refonte visuelle plus moderne, en intégrant des animations fluides lorsqu'on appuie sur les boutons ou lors de la réception de notifications. L'ajout d'options de personnalisation permettrait également aux utilisateurs de modifier les thèmes de couleurs, l'agencement des informations ou les types d'alertes, adaptant ainsi l'application à leurs préférences individuelles. Enfin, des vibrations ou des alertes sonores configurables au sein de l'application elle-même pourraient enrichir l'expérience utilisateur et renforcer la perception des notifications importantes au sein de l'application. Prévoir une version équivalente sous un autre système d'exploitation mobile serait également envisageable pour un aspect adaptatif qui permet de viser une communauté d'utilisateurs plus large.

Références bibliographiques

- [1] Android Developers. Documentation officielle. Disponible sur <https://developer.android.com/>. Consulté le 3 juin 2025.
- [2] Android studio. Disponible sur <https://developer.android.com/studio?hl=fr>. Consulté le 05 juin 2025.
- [3] JetBrains Marketplace. Android Studio releases list. Disponible sur <https://plugins.jetbrains.com/docs/intellij/android-studio-releases-list.html#2023>. Consulté le 06 juin 2025.
- [4] novelbits.Bluetooth Low Energy. Disponible sur <https://novelbits.io/uuid-for-custom-services-and-characteristics/>. Consulté le 15 juin 2025.
- [5] Log Rocket.Kotlin queue guide. Disponible sur <https://blog.logrocket.com/>. Consulté le 15 juin 2025.

Annexe

1. Permissions pour l'Application

L'application aura accès à plusieurs fonctionnalités du smartphone comme le Bluetooth, la localisation, la liste des contacts, etc. Pour cela, on lui donne des permissions citées dans le fichier **AndroidManifest.xml**.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
    tools:ignore="CoarseFineLocation" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE"
    tools:ignore="ProtectedPermissions" />
```

2. Fonction de Scan BLE

Le scan permet de détecter la présence du tableau de bord. Ce dernier, une fois trouvé, l'application se connecte d'une manière automatique avec lui.

```
private val scanCallback = object : ScanCallback() {

    @SuppressWarnings("MissingPermission")
    override fun onScanResult(callbackType: Int, result:
        ScanResult) {
        val device = result.device
        if (device.name != null && device.name.contains("
            ESP32_S3")) { //si l'ESP est trouvé
            statusTextView.text = "ESP32 trouvé: ${device.name},
                connexion..."
            bluetoothLeScanner.stopScan(this)
            connectToDevice(device) //on se connecte au tableau
                de bord
        }
    }

    @SuppressWarnings("SetTextI18n")
    override fun onScanFailed(errorCode: Int) { //si le scan a
        échoué
        statusTextView.text = "Scan échoué: $errorCode"}
}
```

3. Fonction pour activer la localisation

La localisation est importante dans les opérations BLE. On doit vérifier si elle est activée et demander à l'utilisateur de l'activer, sinon en le dirigeant vers les paramètres dans son smartphone.

```
private fun ensureLocationEnabled() {
    val locationManager = getSystemService(Context.
        LOCATION_SERVICE) as LocationManager
    val isLocationEnabled = locationManager.isProviderEnabled(
        LocationManager.GPS_PROVIDER)
        || locationManager.isProviderEnabled(LocationManager
            .NETWORK_PROVIDER)

    if (!isLocationEnabled) { //si la localisation est
        désactivée afficher la boîte de dialogue
        AlertDialog.Builder(this)
            .setTitle("Enable Location")
            .setMessage("This app requires location services for
                BLE scanning.")
            .setPositiveButton("Open Settings") { _, _ →
                val intent = Intent(Settings.
                    ACTION_LOCATION_SOURCE_SETTINGS) //ouvrir les
                    paramètres de localisation en appuyant sur l
                    'option "open settings"
                startActivity(intent)
            }
            .setCancelable(false)
            .show()
    }
}
```