

Noé LINDENLAUB
Xavier HUEBER
ISC 3 IE-a

Rapport de projet

Développement

Mobile :

RGB WeatherKit



Table des matières

Table des figures	3
1. <i>Introduction</i>	4
Table des abréviations.....	4
2. <i>Cahier des charges</i>	5
Modalités	5
Matériel Utilisé	6
Description des fonctionnalités.....	6
3. <i>Bluetooth Low Energy</i>	7
 Bluetooth Low Energy ?	7
 Raspberry et Matrice.....	7
Structure du programme	8
 Documentation officielle	9
 Bibliothèque BLE Library	9
4. <i>Structure de l'application.....</i>	10
 BLEInterface.....	10
Initialisation.....	11
Méthodes publiques	11
Méthodes privées	11
Problèmes perçus	12
 Activités et fragments	12
MenuPrincipal	12
FragmentBTSettings	13
FragmentHome	13
FragmentManual	14
FragmentWeather.....	15
InfoPopUp.....	15
NoBLEAuthorization	15

Fonctionnement final	16
Traduction	19
CI/CD et Gitlab.....	19
5. Annexe	20
Changelog	20
Application iOS	21
Bibliographie	22

Table des figures

Figure 1 - Adafruit RGB Matrix Bonnet for Raspberry Pi	7
Figure 2 - Fichier devices.txt.....	8
Figure 3 - Arborescence de l'application sur Android Studio	10
Figure 4 Vue dans Android Studio de menu principal	12
Figure 5 Fragment BT Settings, coupé en hauteur	13
Figure 6 Fragment Home.....	13
Figure 7 FragmentManual : envoi de pixel	14
Figure 8 FragmentManual : envoi de texte.....	14
Figure 9 Fragment Send Weather, coupé en hauteur	15
Figure 10 - Capture d'écran du fragment Home	16
Figure 11 - Capture d'écran de FragmentBTSettings	16
Figure 12 - Crédits	17
Figure 13 - Capture d'écran du fragment Google Maps	17
Figure 14 - Capture d'écran du fragment Weather.....	17
Figure 15 - Bouton d'information	17
Figure 16 - Photo du résultat d'envoi de texte avec les paramètres ci-contre.....	18
Figure 17 - Capture d'écran du Fragment Manual	18
Figure 18 - Exemple de pipeline obtenu automatiquement après un push sur git ...	19

1. Introduction

Dans ce rapport, nous présenterons un projet sur la création d'une application Android qui permet de contrôler une matrice LED via Bluetooth Low Energy (BLE). Le but de ce projet est de développer une application facile à utiliser qui permet à l'utilisateur de configurer et de contrôler la matrice LED à distance via un smartphone ou une tablette. Nous expliquerons les différentes étapes de développement, les défis rencontrés et les solutions mises en place pour résoudre ces problèmes. Nous décrirons également les fonctionnalités de l'application et les tests réalisés pour s'assurer de son bon fonctionnement.

Ce projet permet de connecter des dispositifs physiques à des appareils mobiles, ouvrant ainsi la porte à de nombreuses applications pratiques, comme les systèmes d'affichage dynamique, les systèmes de notification, etc. Il s'agit ici de pouvoir avoir une information en temps réel sur la météo, sur une surface visible, et facilement accessible, il est donc un exemple d'utilisation pratique de l'IoT. C'est un sujet passionnant qui nous a permis de mettre en pratique nos connaissances en développement d'application mobile, en Bluetooth Low Energy, ainsi qu'en Java & Kotlin Android.

Table des abréviations

Terme	Abréviation
Bluetooth Low Energy	BLE
Application Programming Interface	API
Internet of Things	IoT
Red Green Blue (matrice)	RGB

2. Cahier des charges

Selon le cahier des charges, nous devions réaliser une application dont le principe fondamental consiste en la communication via Bluetooth Low Energy entre un téléphone et une carte Raspberry pour contrôler une matrice de LED.

Cette communication une fois mise en place, nous laisserait la possibilité d'afficher des patterns que nous avons réalisé nous même à l'avance. Ainsi, notre objectif est d'envoyer toutes les informations nécessaires à l'affichage depuis le téléphone directement sur le Raspberry Pi contrôlant la matrice. Le but est dans un premier temps l'affichage de pixels sélectionnés par l'utilisateur avec des coordonnées et la couleur.

Enfin, l'objectif est d'afficher une station météo incluant plusieurs éléments, comme la température et les conditions météorologiques à la localisation.

Modalités

Selon les guidelines de ce travail, le projet a les contraintes suivantes :

- Projet par groupe de 2
- Sujet : Libre
- Application disponible dans 2 langues : français et anglais
- L'application orienté mobile doit être implémenté en Android-Kotlin, composé d'une interface graphique, et doit prendre avoir au moins au moins deux des 4 points suivants :
 - Mesure de performance (temps, délai au démarrage, de réponse, de téléchargement, nombre maximal d'images par seconde, fréquence sonore maximale, etc.)
 - Persistance (stockage de données sur le téléphone)
 - **Gestion de l'utilisation d'un moyen de communication** : Wifi, 4G, 5G, **Bluetooth**. Par exemple, gérer la connexion RF (activer/désactiver, choix de canal de communication, etc.), indiquer à l'utilisateur qu'il est passé de 4G à 5G, etc.
 - **Utilisation d'au moins deux capteurs** : **GPS**, accéléromètre, magnétomètre, camera, microphone, contexte (Wifi/ **BT**/ 3G/ 4G/ Mode Avion/ Time Zone/ Battery Level/ Proximity/ Light/ etc.).

Nous avons donc utilisé le Bluetooth pour la gestion de la communication, étant le moyen de communiquer avec le Raspberry le plus efficient possible pour notre utilisation, et très peu gourmand en ressources et donc en batterie.

La géolocalisation GPS est utilisée avec l'API Google Maps pour détecter la position de l'utilisateur, et obtenir les informations météorologiques pour cet emplacement. La localisation est par ailleurs également transmise à la matrice pour préciser le lieu pour

lequel les informations sont affichées.

Matériel Utilisé

- Matrice LED RGB 32*64 Adafruit
- Raspberry Pi 3b
- Bonnet RGB Matrix pour Raspberry
- Alimentation 5V pour la matrice & Raspberry via le Bonnet
- Téléphone Android 7 ou plus disposant de BLE.

Comparé au cahier des charges initial, nous avons réussi à rendre l'application compatible également aux versions Android 7, initialement étant Android 8 et plus prévu.

Description des fonctionnalités

- Envoi de coordonnées de la matrice RGB pour définir une couleur.
- Envoi d'une ligne de texte sur la matrice.
- Affichage de la météo en fonction de la localisation du téléphone, ou d'un lieu spécifique incluant les conditions météorologiques, d'humidité et de température.

Chacune de ces fonctionnalités fonctionne via des activités différentes, ou bien des fragments. Nous expliquerons le détail dans [Activités et fragments](#).

Il est également à noter que l'évolution de l'application et des fonctionnalités implémentées sont disponibles dans l'annexe [Changelog](#).

Nous avons également imaginé des fonctionnalités additionnelles si le temps nous le permet.

- Sauvegarde des patterns réalisé sus le téléphone
- Mode gravité utilisant l'accéléromètre du téléphone pour afficher des pixels se déplaçant en tombant
- Développement de la même application pour téléphones Apple (iOS)

Nous n'avons eu le temps que de développer une application similaire, bien que très simplifiée pour iOS.

3. Bluetooth Low Energy

La première chose à laquelle nous nous sommes attaqués en octobre était de voir si nos idées étaient viables, nous avons alors commencé par mettre en route la matrice de LEDs et regardé comment l'utiliser à partir du Raspberry Pi.

Bluetooth Low Energy ?

Mais avant tout commençons par regarder comment fonctionne basiquement le BLE.

Le Bluetooth Low Energy est une technologie de communication qui fonctionne en parallèle du Bluetooth et qui a été développé par Nokia en 2006 de manière indépendante. En 2010, le BLE a été ajouté à la norme du Bluetooth 4.0.

Le BLE permet entre autres de consommer 10x moins que le Bluetooth, mais a un débit beaucoup plus limité et un rayon d'action de 10m (soit 10x moins que le Bluetooth).

La particularité cependant du BLE c'est qu'il permet dans la norme de créer des services (que l'on peut comparer à une structure) comprenant des caractéristiques (que l'on peut comparer à des variables). Ces caractéristiques peuvent alors être créées par un serveur et des clients peuvent s'y connecter et lire/écrire ces caractéristiques.

Raspberry et Matrice

La matrice et le Raspberry communiquent au travers d'un header spécial que nous avons dû acheter (voir Figure 1). Une fois celui-ci installé sur le Raspberry Pi, nous avons pu utiliser les librairies fournies par Adafruit pour contrôler la matrice de LEDs.

Vous trouverez ainsi sur le Gitlab de la HE-ARC¹ dans la branch "Raspberry" le code qui est utilisé sur pour communiquer par BLE avec l'appareil Android et communiquer avec la Matrice.

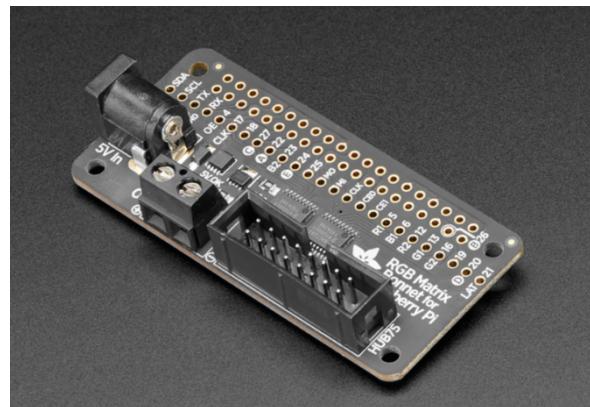


Figure 1 - Adafruit RGB Matrix Bonnet for Raspberry Pi

La thématique du projet se concentrant exclusivement sur le développement Android, je n'effleurerai que la surface du sujet et n'irai pas dans les détails.

¹ <https://gitlab-etu.ing.he-arc.ch/isc/2022-23/niveau-3/3294-4-devemob/devmob-rgbweatherkit-/tree/Raspberry>

Structure du programme

La bibliothèque Bluetooth btferret² que nous utilisons, nous permet d'utiliser les api Bluetooth du kernel linux installé sur le Raspberry Pi. Celle-ci commence au lancement du logiciel par regarder dans le fichier devices.txt et de créer des caractéristiques. On peut ainsi voir dans la ci-dessous, que nous créons un service

```
DEVICE = ledPy  TYPE=MESH  node=1000  ADDRESS = B8:27:EB:CB:92:8A
;LECHAR = NAME  PERMIT=02  SIZE=8  HANDLE=0005 ; index 0
LECHAR = NAME  PERMIT=06  SIZE=8  HANDLE=0005  UUID=BADDCAFE00000000000000000000000000000001 ; index 0
LECHAR = PixelX  PERMIT=06  SIZE=1  HANDLE=0007  UUID=BADDCAFE00000000000000000000000000000002 ; index 1
LECHAR = PixelY  PERMIT=06  SIZE=1  HANDLE=000A  UUID=BADDCAFE00000000000000000000000000000003 ; index 2
LECHAR = Color  PERMIT=06  SIZE=10  HANDLE=000C  UUID=BADDCAFE00000000000000000000000000000004 ; index 3
LECHAR = Send  PERMIT=06  SIZE=1  HANDLE=000E  UUID=BADDCAFE00000000000000000000000000000005 ; index 4
LECHAR = Text  PERMIT=06  SIZE=20  HANDLE=0011  UUID=BADDCAFE00000000000000000000000000000006 ; index 5
;LECHAR = Service  PERMIT=16  SIZE=4  HANDLE=0011  UUID=2A05 ; index 5 Service changed
```

Figure 2 - Fichier devices.txt

avec 6 caractéristiques. La première correspond au nom du serveur. Les deux suivantes aux coordonnées x et y que nous voulons avoir sur la matrice. La quatrième est utilisée pour recevoir la couleur que l'on désire et la sixième est utilisé pour envoyer du texte.

Enfin la cinquième est utilisé pour dire quel mode nous voulons utiliser (textuel ou pixel par pixel) ainsi que d'un bit d'envoi pour tout afficher sur la matrice.

La bibliothèque btferret va alors s'occuper des connections sur le Raspberry Pi qui va servir de serveur et l'appareil Android sera reconnu en tant que client.

Une fois que l'appareil Android aura envoyé les valeurs souhaitées des caractéristiques, une fonction de callback sera appelée et après avoir vérifié que le bit d'envoi a été mis à 1 on vient lire les caractéristiques.

En fonction des options choisis (mode textuel ou mode pixel par pixel par exemple), le programme utilise la bibliothèque led-matrix³ fournis par Adafruit pour allumer les LEDs sur la matrice.

Si vous souhaitez mettre en place l'application, vous pouvez aller suivre le guide qui a été mis en place dans le README sur le Gitlab de la HE-ARC.

Maintenant que vous en savez un peu plus sur le fonctionnement du serveur BLE du côté du Raspberry Pi, nous pouvons passer au fonctionnement côté client.

Pour les besoins des tests, nous avons utilisé dans un premier temps un téléphone sous iOS pour tester le fonctionnement du serveur BLE.

Cependant par manque de temps nous avons décidé de nous consacrer uniquement à l'application Android. Si vous souhaitez cependant voir le code source de cette

² <https://github.com/petzval/btferret>

³ <https://github.com/hzeller/rpi-rgb-led-matrix>

application, il est disponible sur le Gitlab de la HE-ARC⁴ sous la branche iOS. Vous trouverez aussi des images et une courte explication de l'interface de l'application en Annexe.

Documentation officielle

La documentation officielle disponible sur le site de Google donne une explication approximative du mode de fonctionnement du BLE, mais ne rentre pas dans les détails et ne donne pas de guide clair sur ce que voulions faire.

C'est pour cela que nous avons commencé à chercher des exemples de client BLE sur internet et tout particulièrement sur GitHub. Nous avons trouvé quelques exemples qui utilisaient le BLE mais aucun ne nous donnait de solution toute faite qui pouvait être adapté à notre besoin.

Cependant, la gestion des permissions, en particulier pour le Bluetooth et le BLE, est l'élément qui nous a le plus donné le plus fil à retordre, car il n'y avait pas d'indications sur la documentation officielle ni sur les sites d'informations sur lesquels nous regardions.

Bibliothèque BLE Library

Puisque nous n'arrivions pas à trouver de solution toute faite, nous nous sommes tournées vers des librairies qui proposaient des solutions pour pouvoir utiliser le BLE. La bibliothèque BLE Library de Nordic Semiconductor⁵ nous a semblé la meilleure solution, car elle est l'une des librairies les mieux supportés par les développeurs, elle supporte un grand nombre d'options de la norme BLE et est mis à jour régulièrement pour éviter les problèmes de sécurité et disposer des dernières nouveautés de la norme BLE et de Kotlin.

La bibliothèque dispose aussi d'une grande documentation qui nous a beaucoup aidé ainsi que des exemples à partir desquels nous sommes partis pour écrire notre interface Bluetooth.

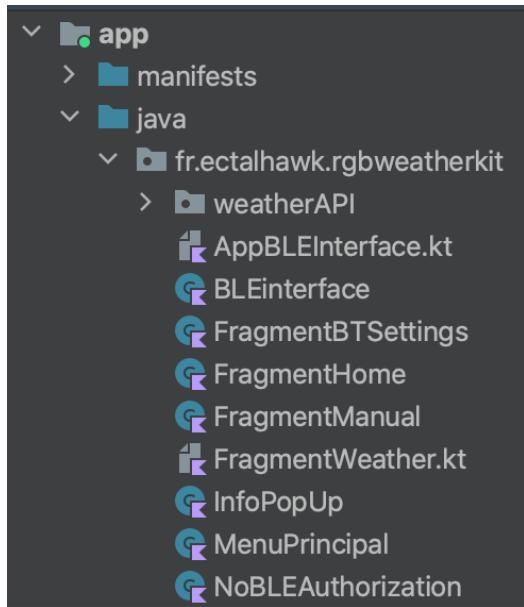
⁴ <https://gitlab-etu.ing.he-arc.ch/isc/2022-23/niveau-3/3294-4-devemob/devmob-rgbweatherkit/-/tree/iOS>

⁵ <https://github.com/NordicSemiconductor/Android-BLE-Library>

4. Structure de l'application

L'application est découpée en différents éléments :

- Les API/Interfaces
- ApplInterface
- Les activités et fragments



Le premier niveau est représenté par les API/Interfaces que nous utilisons pour communiquer d'une part avec les API du kernel pour le BLE et d'autres part pour pouvoir communiquer avec OpenWeatherMap qui est le site d'où nous récupérons les données météo.

Vous trouverez ainsi dans l'arborescence dans Android Studio (voir Figure 3) tout d'abord le dossier contenant les fichiers Kotlin pour l'API communiquant avec OpenWeatherMap.

Ces fichiers viennent de l'application myWeather⁶ et nous n'avions plus qu'à générer une clé d'API sur le site de OpenWeatherMap pour pouvoir l'utiliser.

L'autre Interface visible dans l'arborescence est le fichier BLEinterface, celui-ci sera expliqué dans la prochaine section.

En commençant à utiliser notre interface BLE, nous nous sommes assez vite rendu compte que puisque cette Interface avait été modélisé par une classe et que l'objet initialisé à partir de cette classe devait pouvoir être accessible à partir de n'importe quelle classe dans le projet. C'est pour cela que nous avons crée la classe AppBLEInterface qui nous permet d'avoir un Singleton de la classe BLEinterface ainsi que d'un Singleton de la classe weatherService qui est utilisé par l'API d'OpenWeatherMap.

Enfin le reste des fichiers correspondent aux Activités et Fragments utilisés par l'application, ceux-ci vous seront expliqués plus en détails dans la suite du rapport.

BLEInterface

Nous allons maintenant nous concentrer sur la classe BLEInterface qui permet à l'application de communiquer facilement avec les outils Bluetooth fournis par le kernel d'Android.

⁶ <https://github.com/Atifsid/myWeather>

Initialisation

Lors de l'initialisation du Singleton BLEInterface, qui se fait lors du lancement de l'application, celui va initialiser la liste d'appareils qui seront découvert par l'appareil Android. Cette liste est faite avec un adaptateur écrit dans le fichier AppBLEInterface.

Méthodes publiques

Une fois le Singleton créé, celui-ci nous donne accès à diverses méthodes publiques.

La méthode prepareAndStartBleScan est utilisé lorsque nous voulons chercher des serveurs BLE autour de notre appareil. Ceux-ci seront alors enregistrés dans liste énoncé plus haut et on pourra cliquer sur l'un d'eux pour essayer de se connecter à cet appareil. Les appareils qui sont dans liste ont été triés pour que seul un appareil avec un nom soit affiché et que ce nom n'apparaisse qu'une seule fois dans la liste.

La méthode sendPixel sert, comme son nom l'indique à envoyer un pixel unique sur la matrice de LEDs. Celui-ci va alors s'ajouter aux autres. La méthode prend en paramètres d'entrées la position x, y, la couleur souhaitée et si nous voulons rafraîchir ou nous l'écran. La fonction va dans un premier temps vérifier que les coordonnées x et y données en paramètres sont possibles sur notre matrice 32x64 pixels et va ensuite appeler les différentes méthodes privées pour envoyer le pixel.

La méthode sendText fonctionne de la même façon à la différence qu'elle envoie du texte. Ici aussi on vient vérifier que les coordonnées x et y sont respectées.

Enfin la méthode clearMatrix permet d'envoyer une commande pour effacer la matrice.

Méthodes privées

Les méthodes publiques vues au-dessus utilisent ensuite des méthodes privées que nous allons voir maintenant.

Une fois que nous avons cliqués sur un appareil disponible de la liste, une fonction de callback⁷ va être appelée. Cette dernière va commencer par chercher les caractéristiques disponibles sur l'appareil et si ces caractéristiques correspondent à celles que nous cherchons alors la fonction va accepter la connexion. Sinon un message d'erreur sous la forme d'un toast va s'afficher.

Un autre callback que nous avons utilisé est onCharacteristicWrite. Cette fonction de callback va être appelée lorsque nous modifions une caractéristique. Nous l'avons utilisé, car nous avions remarqué que lors de l'envoi de pixels ou de texte, certaines caractéristiques n'étaient tout simplement pas écrites et l'application Android passait à la prochaine ligne. Nous avons tout d'abord essayé de rajouter des délais entre deux lignes, mais ceux-ci étaient différents pour chaque appareils Android sur lesquels nous testions l'application et nous avions remarqués que ces envois fonctionnaient le mieux sur les anciens appareils (ce qui est sûrement lié à leur rapidité). C'est pour

⁷ onConnectionStateChange

cela que nous avons ensuite décidé de créer une liste de caractéristiques que nous allions envoyer et cette liste serait ensuite lu par la fonction de callback à chaque envoi d'une caractéristique. Ainsi, l'appareil Android pouvait prendre le temps qu'il lui faudrait pour envoyer chaque caractéristique à son rythme.

Enfin les autres méthodes privées disponible dans la classe BLEInterface sont utilisés pour vérifier que les permissions du Bluetooth et de la Localisation ont bien été acquis.

Problèmes perçus

Pendant le développement et l'adaptation de ces méthodes, nous nous sommes heurtés à plusieurs problème. Le plus gros selon nous est le manque de documentation sur le sujet et le manque de normalisation selon la marque, le modèle de l'appareil et la version d'Android qu'il possède. En effet, certaines méthodes fonctionnaient parfaitement sur un téléphone, puis en changeant de téléphone, l'application n'arrivait plus à envoyer de caractéristiques ou n'en envoyait que certains.

Même après avoir modifié la technique d'envoi pour que celle-ci se fasse au rythme de l'appareil et non avec un rythme prédéfini, un des téléphones⁸ envoi toujours maintenant les caractéristiques de façon hasardeuse.

Activités et fragments

MenuPrincipal

L'activité MenuPrincipal (voir Figure 4) est le cœur de l'application. C'est en effet elle qui va gérer l'affichage des différents fragments ainsi que la barre de navigation. On retrouve également une grande partie de l'initialisation des fonctionnalités Bluetooth.

Lors de la création de l'activité, celle-ci va commencer par demander les permissions nécessaires pour utiliser le BLE. Elle va ensuite initialiser Retrofit qui est notre API html utilisé par l'API pour OpenWeatherMap, puis va initialiser les fragments et afficher le fragment FragmentBTSettings.

Enfin elle va initialiser la barre de navigation en bas de l'écran.

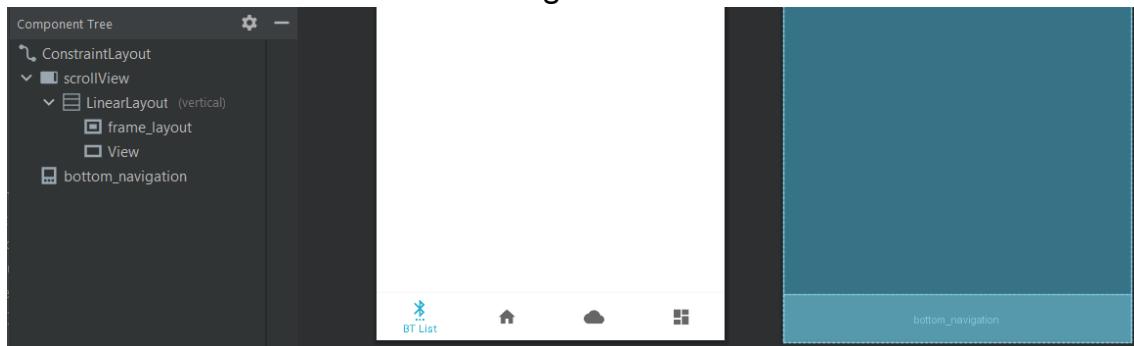


Figure 4 Vue dans Android Studio de menu principal

⁸ OnePlus Nord sous Android 12

On utilise aussi un FrameLayout où l'on va afficher les différents Fragments cité par la suite. En effet, la classe MenuPrincipal dispose d'une méthode replaceFragment qui permet de lui dire quel Fragment afficher. Elle dispose aussi des méthodes activateBottomNavigation et deactivateBottomNavigation qui comme leurs noms l'indiquent permettent d'activer ou de désactiver la barre de navigation.

Nous la désactivons, car nous ne voulons pas que ces éléments soient accessibles lorsque l'appareil Android n'est pas connecté à un appareil.

FragmentBTSettings

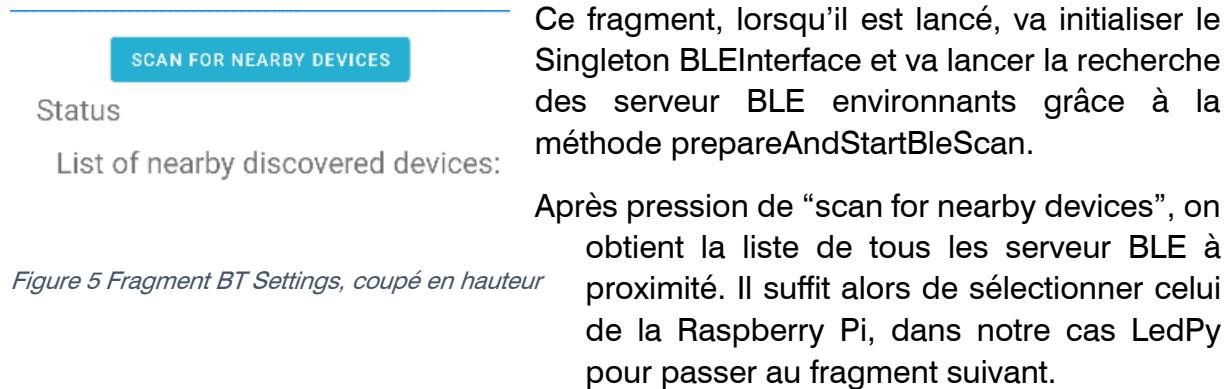


Figure 5 Fragment BT Settings, coupé en hauteur

FragmentHome



Figure 6 Fragment Home

FragmentManual

FragmentManual provient en grande partie de l'activité de test que nous utilisions au début pour tester la matrice de LEDs, le programme sur le Raspberry Pi ainsi que l'API BLEInterface.

Le fragment est découpé en deux parties, la partie haute permettant d'envoyer des pixels (voir Figure 7) et la partie basse permettant d'envoyer du texte et d'effacer la matrice de LEDs (voir Figure 8).

On y retrouve deux sliders allant de 0 à 63 en x et 0 à 31 en y, ce qui correspond à la taille de la matrice de LEDs.

On utilise ensuite une view pour afficher la couleur que l'on veut sélectionner, en cliquant sur le bouton "Pick Color", celui-ci ouvre un color picker venant de la bibliothèque ambilwarna⁹. Ce color picker va aussi afficher la dernière couleur choisie lorsqu'il s'ouvre.

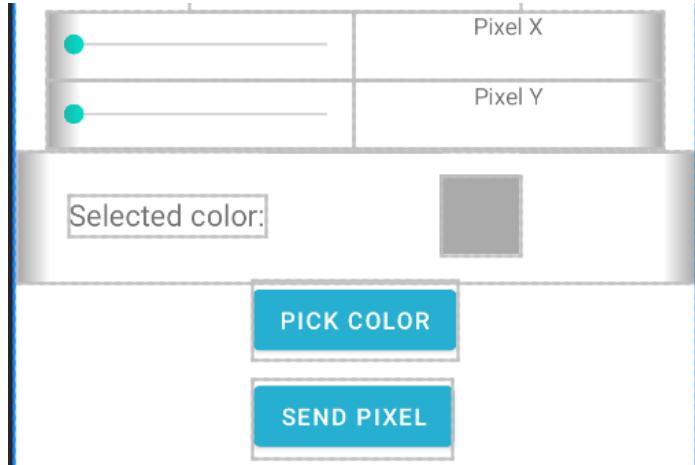


Figure 7 FragmentManual : envoi de pixel



Figure 8 FragmentManual : envoi de texte

Un bouton placé en dessous du bouton "Pick Color" permet d'envoyer le pixel sur la matrice.

En descendant encore, on peut voir sur la Figure 8 à gauche, que nous avons mis un élément "EditText" qui permet de rentrer le texte que nous souhaitons, pour que ce dernier soit affiché sur la matrice.

Enfin on retrouve le bouton "Send Text" qui sert à envoyer le texte écrit de la couleur choisie plus haut à la matrice de LEDs et le bouton "Clear Matrix" qui sert à effacer la matrice de LEDs.

⁹ com.github.yukuku:ambilwarna:2.0.1

FragmentWeather

Le FragmentWeather (voir Figure 9) sert à afficher la météo sur la matrice de LEDs. En effet, lorsqu'on va appuyer sur le bouton "Confirm this location", celui-ci va faire une requête html à l'API d'OpenWeatherMap au travers de Retrofit. On va utiliser le texte écrit dans l'Edit Text au-dessus pour connaître la météo à cette localisation.

On peut aussi cliquer sur le bouton "Open location tool" pour ouvrir une carte Google Maps et sélectionner un lieu. Cette option a été réalisé grâce à la bibliothèque Leku¹⁰, nous n'avons eu qu'à créer une compte développeur Google et à créer une clé d'api Google Maps que nous avons ensuite donné à la bibliothèque.

Nous voulions d'abord essayer de créer nous même une activité avec la carte Google Maps, mais après avoir suivi le guide dans la documentation officielle, celle-ci ne nous disait pas comment récupérer la position. Nous avons donc cherché des solutions toutes faites et Leku nous semblait approprié à cette tâche.

Une fois la recherche faite, l'API d'OpenWeatherMap nous rends une structure de données de laquelle nous pouvons extraire le nom de la ville et la température que nous allons envoyer sur la matrice de LEDs.

InfoPopUp

InfoPopUp est comme son nom l'indique, une PopUp qui contient les informations de crédit des auteurs, ainsi que des bibliothèques importantes utilisées par l'application.

Cette Activité a l'avantage d'être totalement customisable et d'être autonome, ainsi si on a besoin d'une autre PopUp à un autre endroit dans l'application, nous avons juste besoin de faire une Intent et de lui donner le texte à afficher avec la méthode "putExtra". Cette activité a été créé à l'aide d'un guide de John Codeos¹¹.

NoBLEAuthorization

NoBLEAuthorization est un fragment qui apparaîtra si l'utilisateur n'a pas accepté toutes les demandes d'autorisation Android telles que la localisation, le scan à proximité etc.

Il invite celui-ci à fermer l'application, lui fournir les droits puis de la rouvrir.

¹⁰ <https://addevintaspain.github.io/Leku/>

¹¹ <https://johncodeos.com/how-to-create-a-popup-window-in-android-using-kotlin/>

Send Weather

City

CONFIRM THIS LOCATION

OPEN LOCATION TOOL

Figure 9 Fragment Send Weather, coupé en hauteur

Fonctionnement final

Nous allons détailler ici un exemple de l'utilisation complète de l'application.

Dans un premier temps, on lance l'application, qui va demander les autorisations d'appareils à proximité, ainsi que la géolocalisation. Il faut impérativement accepter des autorisations, sans quoi l'application ne sera pas fonctionnelle. Une fois les autorisations acceptées, et le Raspberry Pi allumé, il suffit d'appuyer sur le bouton **SCANNER LES APPAREILS A PROXIMITE**

On aura normalement ledPy qui apparaît. On clique dessus, la connexion s'établit et on arrive sur le fragment Home, qui explique tous les logos de la barre de navigation.

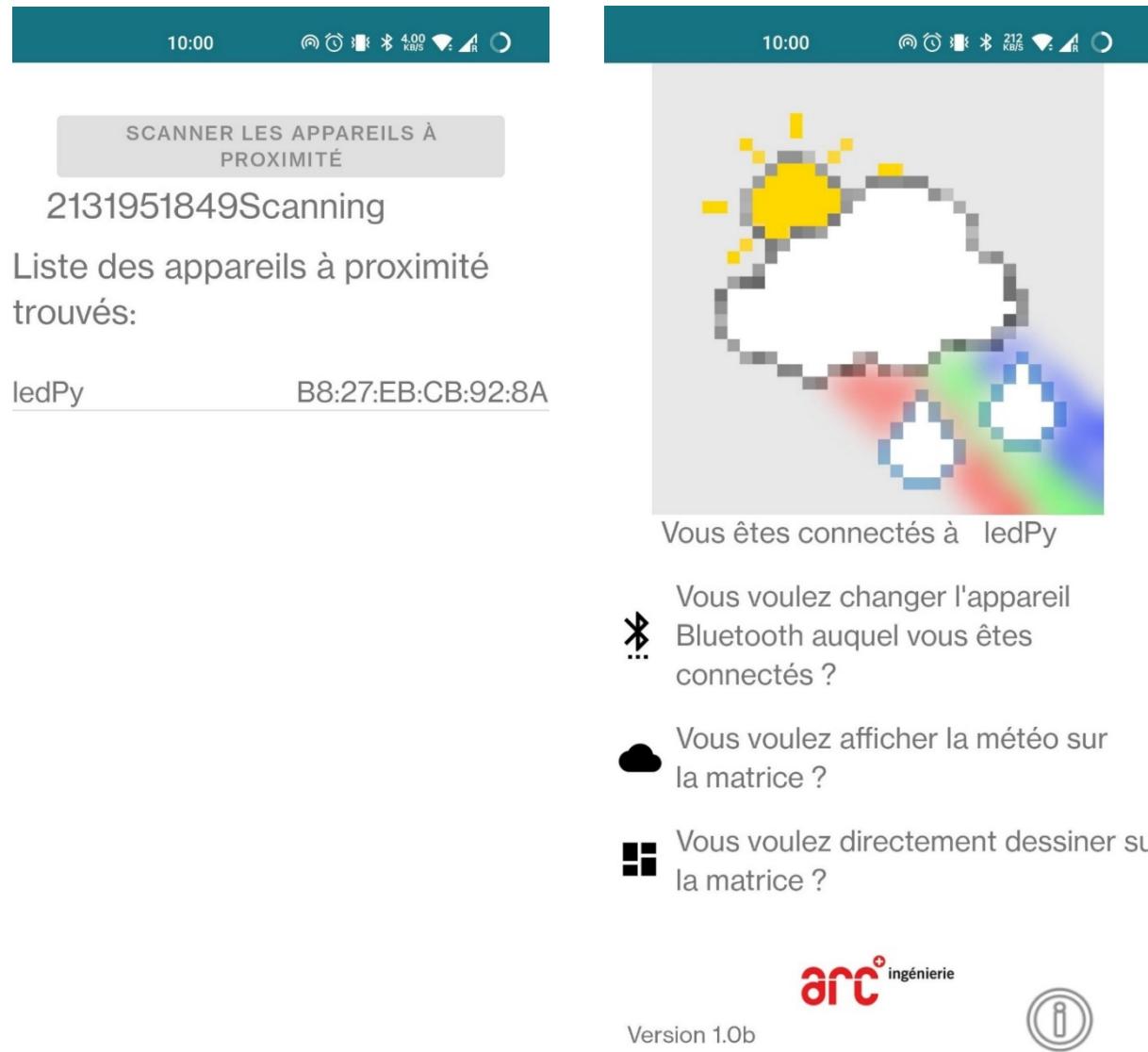


Figure 11 - Capture d'écran de FragmentBTSettings

Figure 10 - Capture d'écran du fragment Home

On retrouve l'icône de notre application, ainsi que le logo de la HE Arc ingénierie. On peut alors appuyer sur l'image d'information (voir Figure 15) cela nous ouvrira les crédits (voir Figure 12).

On peut alors cliquer sur la barre de *Figure 15 - Bouton d'information* navigation en bas sur le nuage, pour accéder à l'onglet envoi météo (voir Figure 14). On a alors deux options. Soit envoyer la météo avec une localisation manuelle, puis appuyer sur **VALIDER CETTE LOCALISATION**, qui va faire une requête API pour ce lieu, puis envoyer les informations sur le



Envoi de la météo

Neuchâtel

VALIDER CETTE LOCALISATION

OUVRIR L'OUTIL DE LOCALISATION



Figure 14 - Capture d'écran du fragment Weather



Figure 12 - Crédits



Figure 13 - Capture d'écran du fragment Google Maps

Raspberry Pi pour affichage sur matrice. Sinon, on peut [OUVRIR L'OUTIL DE LOCALISATION](#).

Cela va nous ouvrir un fragment Google Maps (voir Figure 13). Il suffit d'appuyer sur **USE THIS LOCATION** pour mettre la position sur soi-même. Comme précédemment, il faut **VALIDER CETTE LOCALISATION**.

Enfin, on peut se rendre sur le fragment Manual (voir Figure 17), qui nous permet simplement d'envoyer un pixel de la couleur de notre choix à la coordonnée de notre choix, avec **CHOISIR LA COULEUR** et **ENVOYER LE PIXEL**.

Plus bas, on peut écrire un texte, puis **ENVOYER LE TEXTE**. Qui écrira le texte entré



Figure 16 - Photo du résultat d'envoi de texte avec les paramètres ci-contre



Figure 17 - Capture d'écran du Fragment Manual

Traduction

Comme précisé dans le cahier des charges, notre application est disponible dans 2 langues. Elle est intégralement traduite, qu'il s'agisse du texte affiché, toasts, boutons et même du texte affiché sur la matrice de LEDs.

Les deux langues sont le français et l'anglais.

CI/CD et Gitlab

Pour pouvoir travailler ensemble, nous avons décidé de mettre les différents projets sur Gitlab au tout début du projet. Ainsi nous pouvions aussi revenir en arrière si une mise à jour ne nous plaisait pas ou garder une trace de l'avancement du projet.

Ainsi à ce jour (23/01/2023), nous avons fait plus d'une centaine de commit vers Gitlab et avons pu être mis au courant de bugs potentiels au travers de notre système de CI/CD.

En effet, un CI/CD (Continuous Integration / Continuous Delivery) a été mis en place pour ce projet car celui-ci devait servir de base de travail pour le projet que nous devions faire dans le cours de Qualité Logiciel.

A chaque envoi d'une nouvelle version de notre code sur le Gitlab, celui-ci a été compilé dans un container docker géré par Gitlab. Pendant la compilation, le compilateur va s'occuper de regarder si le code écrit en Kotlin est juste syntaxiquement. D'autre part il va s'occuper de regarder si tous les termes traduisibles ont bien été traduits et regarder si tous les éléments graphiques contenus dans les layout sont contraints. Vous pourrez ainsi voir sur la Figure 18 un test de compilation CI/CD qui n'a pas eu d'erreurs.

✓ Pipeline has been fixed and #10186 has passed!

Project	3294.4 DEVEMOB / DevMob RGBWeatherKit
Branch	Android
Commit	15380867 Fix permissions for Android 13+
Commit Author	Hueber Xavier

Pipeline #10186 triggered by Hueber Xavier
successfully completed 4 jobs in 2 stages.

5. Annexe

Changelog

Changelog

4 octobre 2022

Ecriture de la V1 du cahier des charges.

6 octobre 2022

Début de la programmation du Raspberry Pi, ajout de la librairie de BLE.

8 octobre 2022

Cassage de tête

10 octobre 2022

Programme raspberry prêt, développement iOS fonctionnel

17 octobre 2022

Changement du nom de l'application

18 octobre 2022

Fin de la création du cahier des charges (voir pdf).
Création du Gantt.
Création de l'interface graphique sur Balsamiq.

20 octobre 2022

Démarrage de la création de l'application Android.

8 novembre 2022

Finir de debugger l'interface Bluetooth et Noé qui continue de programmer l'interface graphique des activités.

15 novembre 2022

Merge de l'interface graphique que Noé a programmé avec l'interface Bluetooth fonctionnelle.

16 novembre 2022

Test d'une première API pour écrire du texte sur la matrice (Ancienne API). Passage d'activités en fragments.

24-25 novembre 2022

Adaptation du programme C pour écrire du texte depuis le téléphone (Nouvelle API).

28-29 novembre 2022

Recherche d'une API météo (on utilisera openweathermap au final), adaptation de l'API sur l'application android actuelle et merge de la nouvelle application C depuis le Raspberry.

6 décembre 2022

Pas mal de debug et trouvé un exemple pour la partie dessin en pixel art.

11 décembre 2022

Revert de ce qui a été fait le 6 décembre (Problème lié à la vitesse des appareils Android).

Test d'un fix définitif.

Implémentation fonctionnelle de BottomNavBar avec fragments. Début du merge de l'interface de Noé avec mon backend.

Figure 18 Changelog disponible sur le GitLab

Application iOS

Attention : Les images de l'application iOS visibles ci-dessous ne sont pas définitives et étaient utilisés dans un cadre de test uniquement.

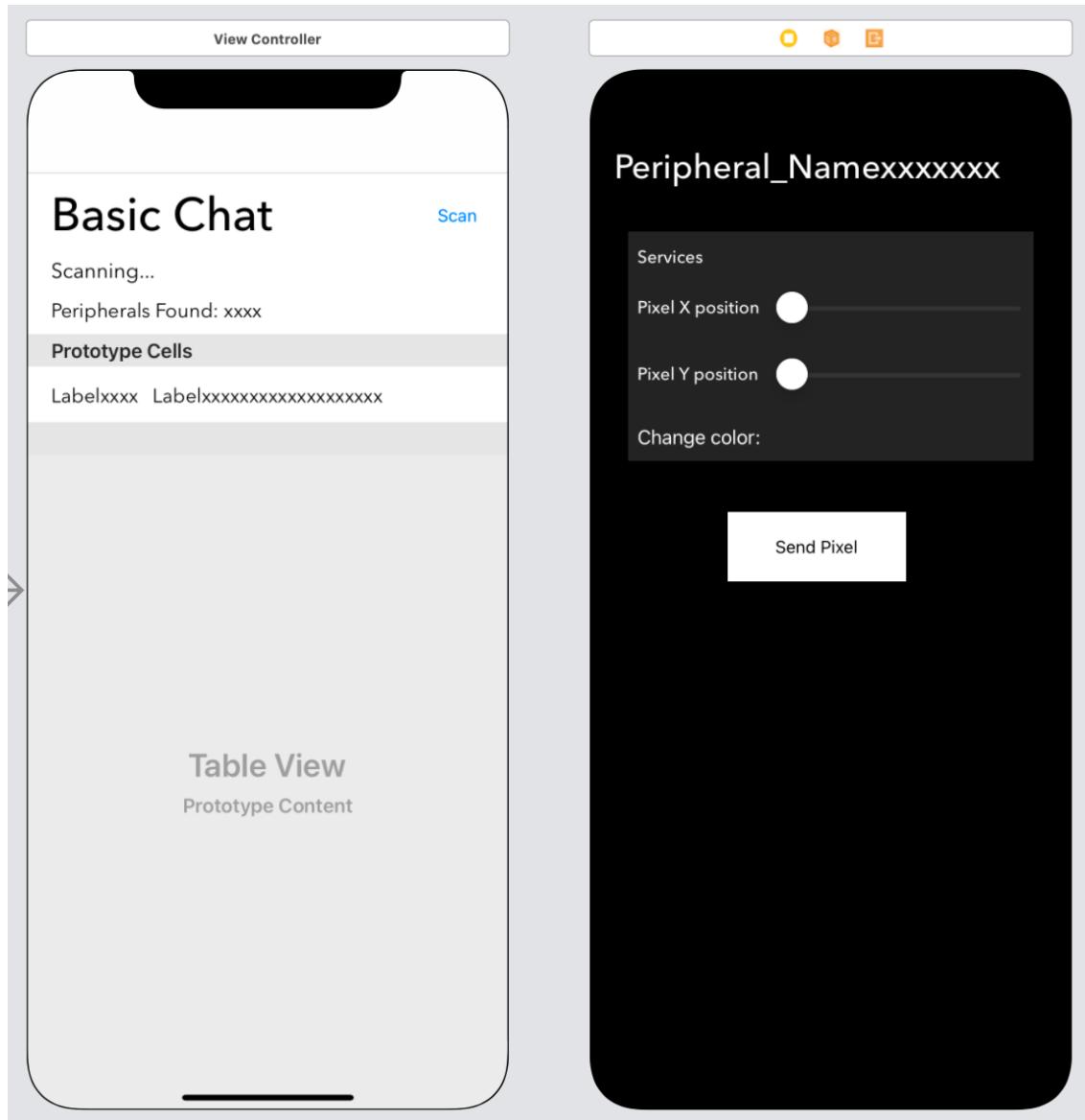


Figure 19 Capture de xCode montrant l'application iOS

On peut voir sur l'image ci-dessus à gauche le menu de sélection d'appareil auquel on veut se connecter et à droite on peut voir le menu d'envoi de Pixels à la matrice (mode pixel par pixel). Celui-ci ressemble au menu “Manuel” disponible sur l'application Android.

Bibliographie

- <https://gitlab-etu.ing.he-arc.ch/isc/2022-23/niveau-3/3294-4-devemob/devmob-rgbweatherkit/-/tree/Raspberry>
- <https://github.com/petzval/btferret>
- <https://github.com/hzeller/rpi-rgb-led-matrix>
- <https://gitlab-etu.ing.he-arc.ch/isc/2022-23/niveau-3/3294-4-devemob/devmob-rgbweatherkit/-/tree/iOS>
- <https://github.com/NordicSemiconductor/Android-BLE-Library>
- <https://github.com/Atifsid/myWeather>
- <https://adevintaspain.github.io/Leku/>
- <https://johncodeos.com/how-to-create-a-popup-window-in-android-using-kotlin/>