Application Android Greenway

EGOR

Sommaire

Introduction

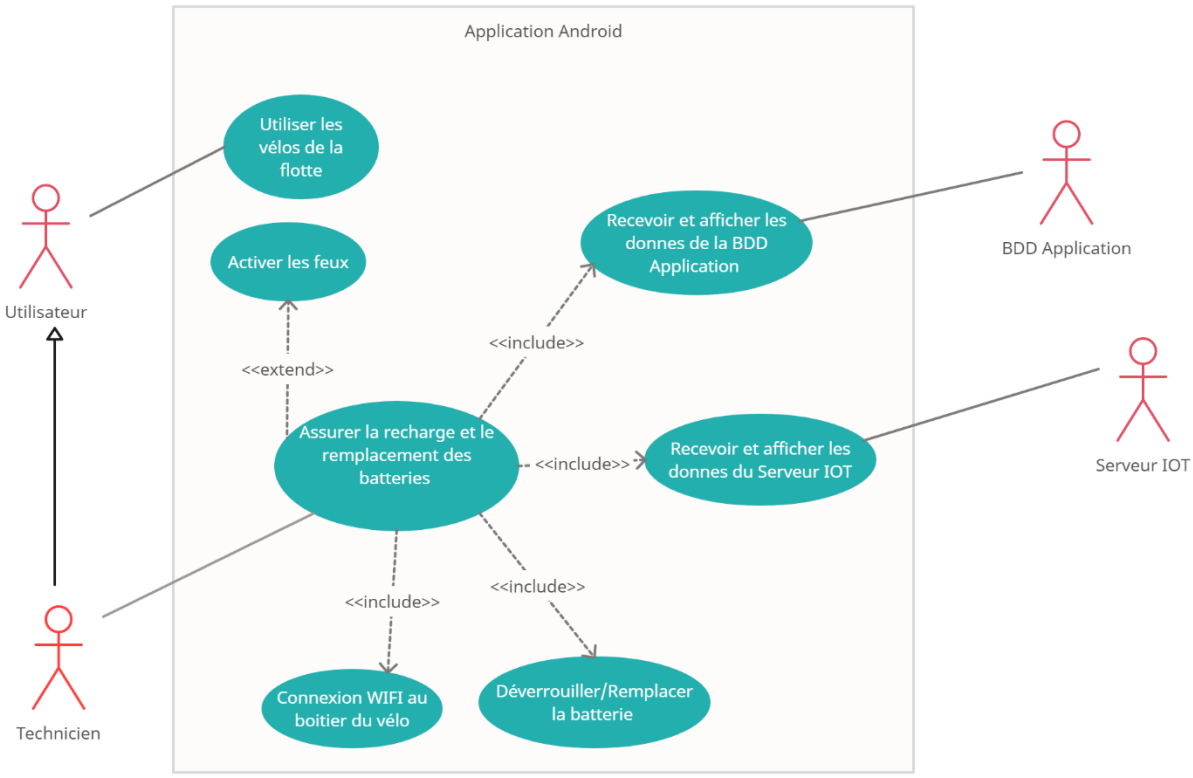
Je suis ravi de vous présenter aujourd'hui mon projet d'application Android, conçu pour faciliter le travail des techniciens chargés de la maintenance des vélos. L'objectif principal de cette application est de permettre aux techniciens de localiser rapidement les vélos et de suivre une procédure détaillée pour effectuer les tâches de maintenance requises.

Lorsqu'il s'agit de systèmes de vélos en libre-service, nous sommes tous conscients de l'importance d'un entretien régulier pour assurer leur bon fonctionnement et leur disponibilité pour les utilisateurs. Cependant, il peut être très difficile pour les techniciens de repérer un vélo spécifique parmi des centaines, voire de milliers de vélos.

C'est là que mon application intervient pour simplifier leur travail. En combinant différentes fonctionnalités clés, nous avons créé un outil convivial spécialement conçu pour les techniciens. Tout d'abord, l'application permet aux techniciens de localiser rapidement un vélo. Une fois la connexion WIFI établie, le technicien peut facilement activer les feux du vélo.

En plus de la localisation, l'application offre également la possibilité de mettre à jour la base de données. Ainsi, après avoir effectué une tâche de maintenance, le technicien peut enregistrer les informations pertinentes, telles que le nouvel identifiant de la batterie, dans la base de données. Cette fonctionnalité garantit une traçabilité des travaux effectués sur chaque vélo.

Voici un diagramme de cas d'utilisation pour montrer ce que représente ma partie sur ce projet :



Consignes

Application Android :

- ✓ Localiser le vélo
- ✓ Créer un trajet
- ✓ Effectuer une connexion WIFI au vélo
- ✓ Allumer les phares du vélo pour pouvoir le localiser
- ✓ Changer la batterie
- ✓ Mettre à jour dans la base de données le nouvel identifiant de la batterie.

Partie ESP32 :

- ✓ Accepter des requêtes HTTP pour allumer une LED.
- ✓ Créer une connexion WIFI possible quand la batterie est inférieure à 15%.

Matériel utilisé

Pour la création l'application Android j'ai décidé de passer par l'IDE Android studio. J'avais le choix entre deux langages Java et Kotlin.

J'ai choisi Kotlin car c'est un langage récent (2011) et très populaire chez les développeurs Android.

Il présente plusieurs avantages par rapport à Java :

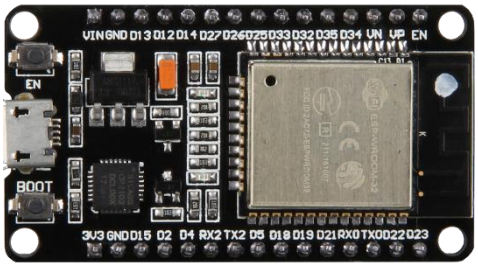
+ Syntaxe concise : Kotlin permet d'écrire un code plus court et plus clair, ce qui rend le développement plus rapide et plus facile à comprendre.
+ Interopérabilité avec Java : Kotlin fonctionne parfaitement avec les bibliothèques et le code Java existants.
+ Manipulation des collections améliorées : Kotlin propose des opérations plus simples pour manipuler les collections de données.
+ Fonctionnalités d'extension : Kotlin permet d'ajouter des fonctionnalités supplémentaires à des classes existantes sans les modifier.

Pour faire le point d'accès WIFI/Serveur WEB j'ai eu besoin d'un ESP32.

Pour le coté programmation je suis passé par Visual Studio avec l'extension PlatformIO, avec un programme écrit en C++.



Quelques caractéristiques de l’ESP 32 :



FRÉQUENCE D'HORLOGE 240 MHz

SRAM 512 Ko

MÉMOIRE 4 Mo

STANDARD SANS FIL 802.11 b/g/n

FRÉQUENCE 2,4 GHz

INTERFACES DE DONNÉES UART / I2C / SPI / DAC / ADC

TENSION DE FONCTIONNEMENT 3,3 V (fonctionnement possible via micro-USB 5V)

TEMPÉRATURE DE FONCTIONNEMENT -40°C - 125°C

DIMENSIONS 48 x 26 x 11,5 mm

Partie graphique

Le design a été réalisé avec l'outil intégré d'Android Studio qui m'a permis de convertir mes "dessins" en code et vice-versa. Tout d'abord je me suis basé sur une maquette déjà existante que j'ai modifiée pour me faire une idée globale de mon application. Les modifications sont faites dans les fichiers .xml.

J'ai décidé de réaliser l'application sur 3 onglets. Pour pouvoir changer de pages le technicien pourra le faire via une barre de navigation.

Pour définir le visuel dans chaque fichier la procédure est identique. Nous avons le choix entre :

- Des boutons créant des interactions avec l'utilisateur.
- Des Spinners qui permettent de choisir une option parmi une liste déroulante.
- Des TextView affichant du texte.
- Des ImageView affichant des images.
- Des RecyclerView affichant des items déjà définis.

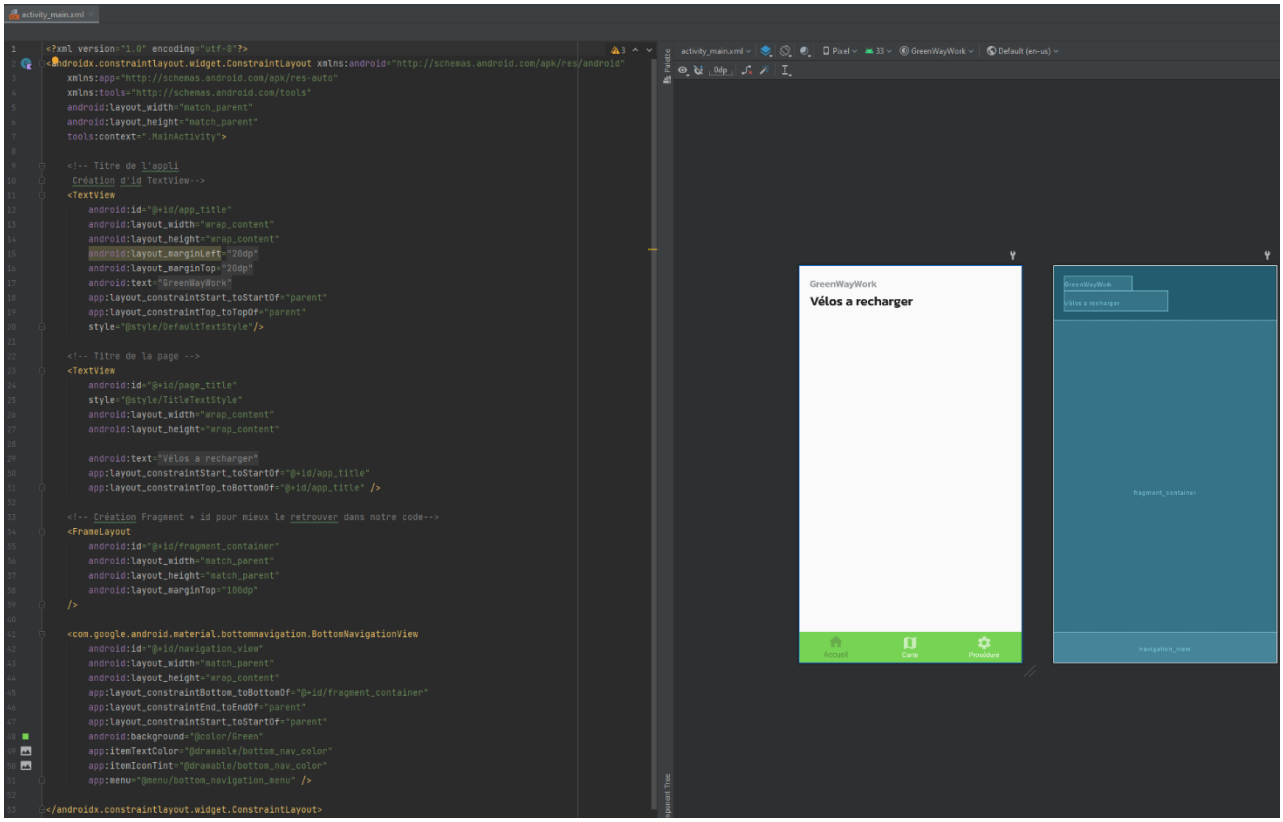
(Items verticaux et horizontaux pour ma part).

Il y a énormément d'autres possibilités.

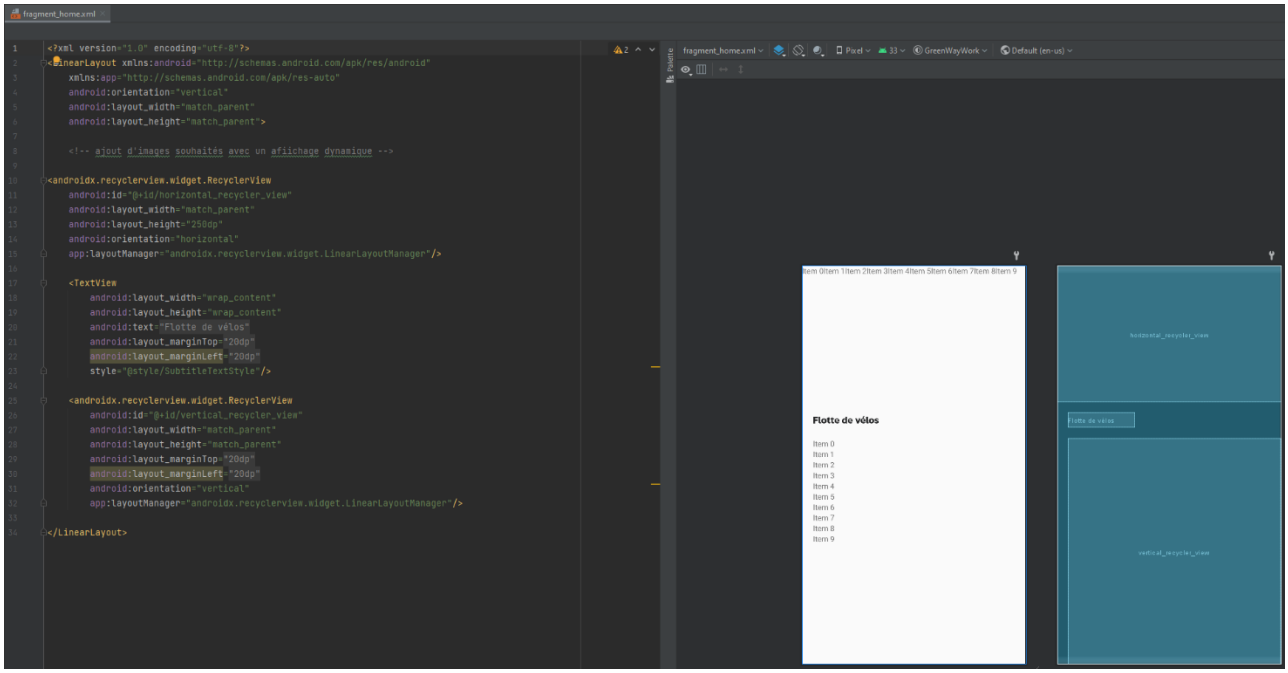
Il ne reste plus qu'à les définir avec des balises dans nos .xml

Définition de l'onglet accueil :

Titre de l'application, intégration de la navbar.

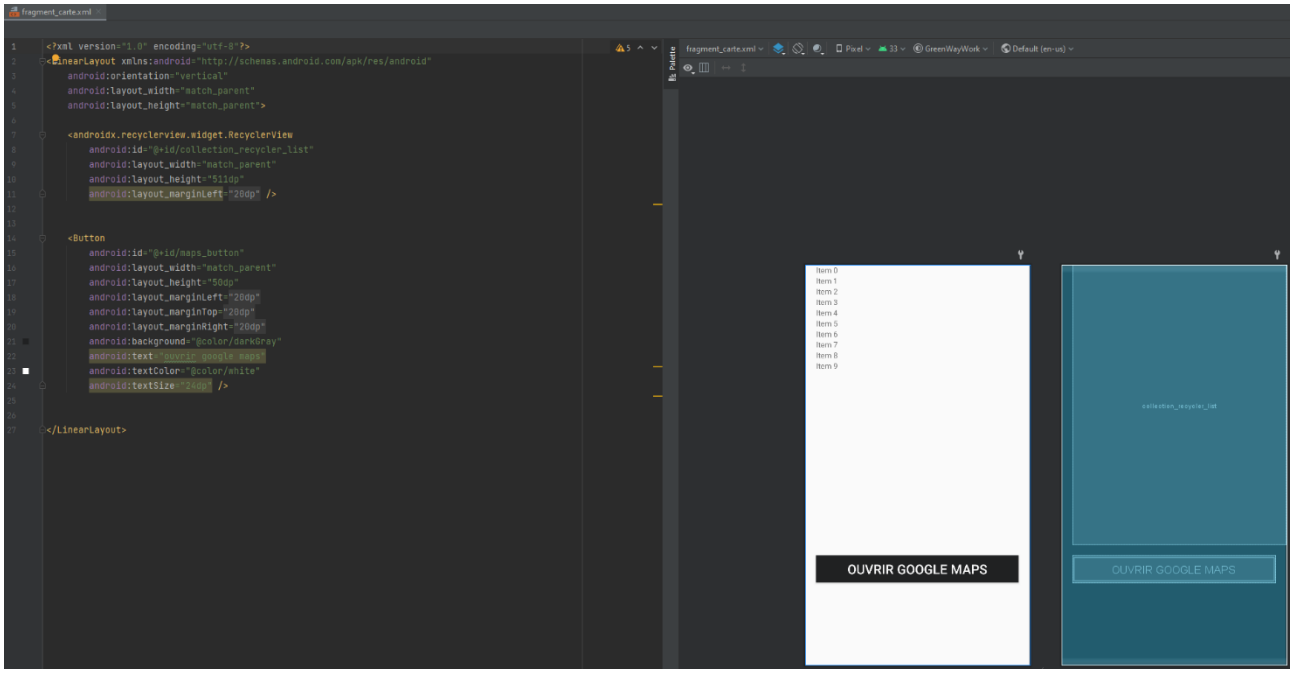


Ajout des items.



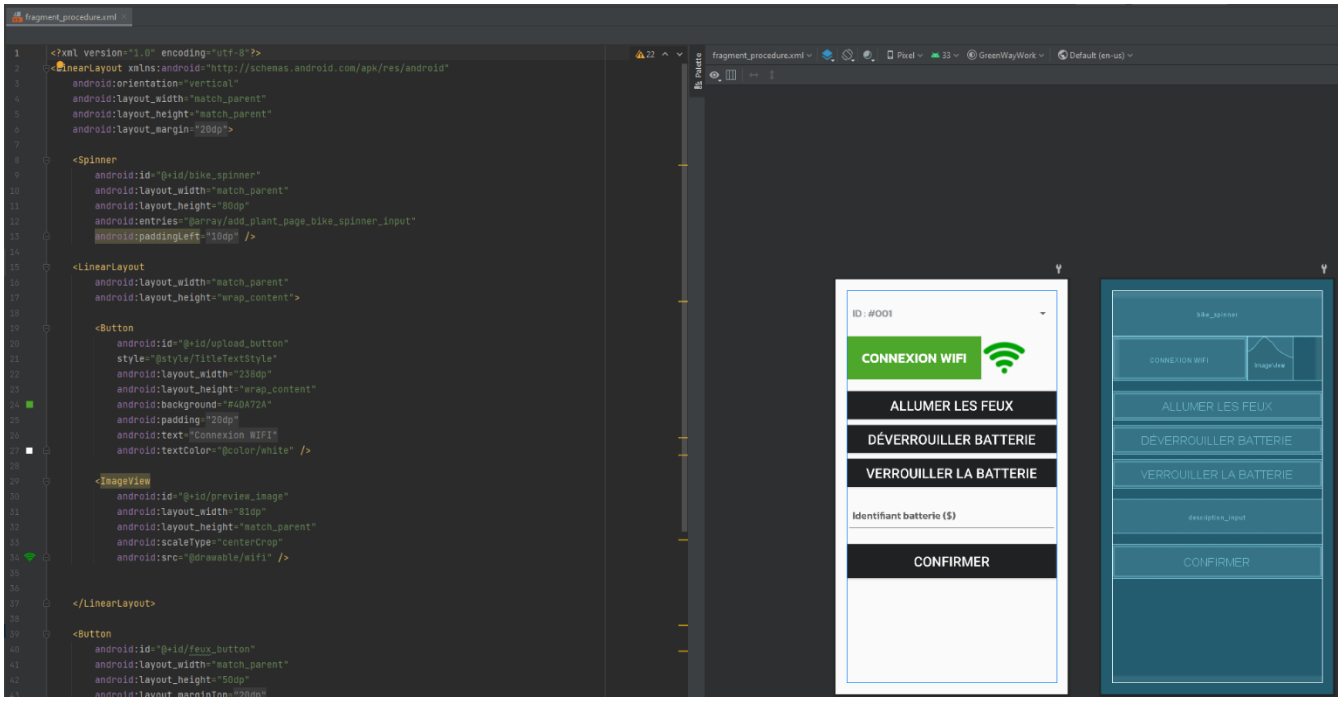
Définition de l'onglet carte :

Ajout des items définis, création d'un bouton, choix dimensions.



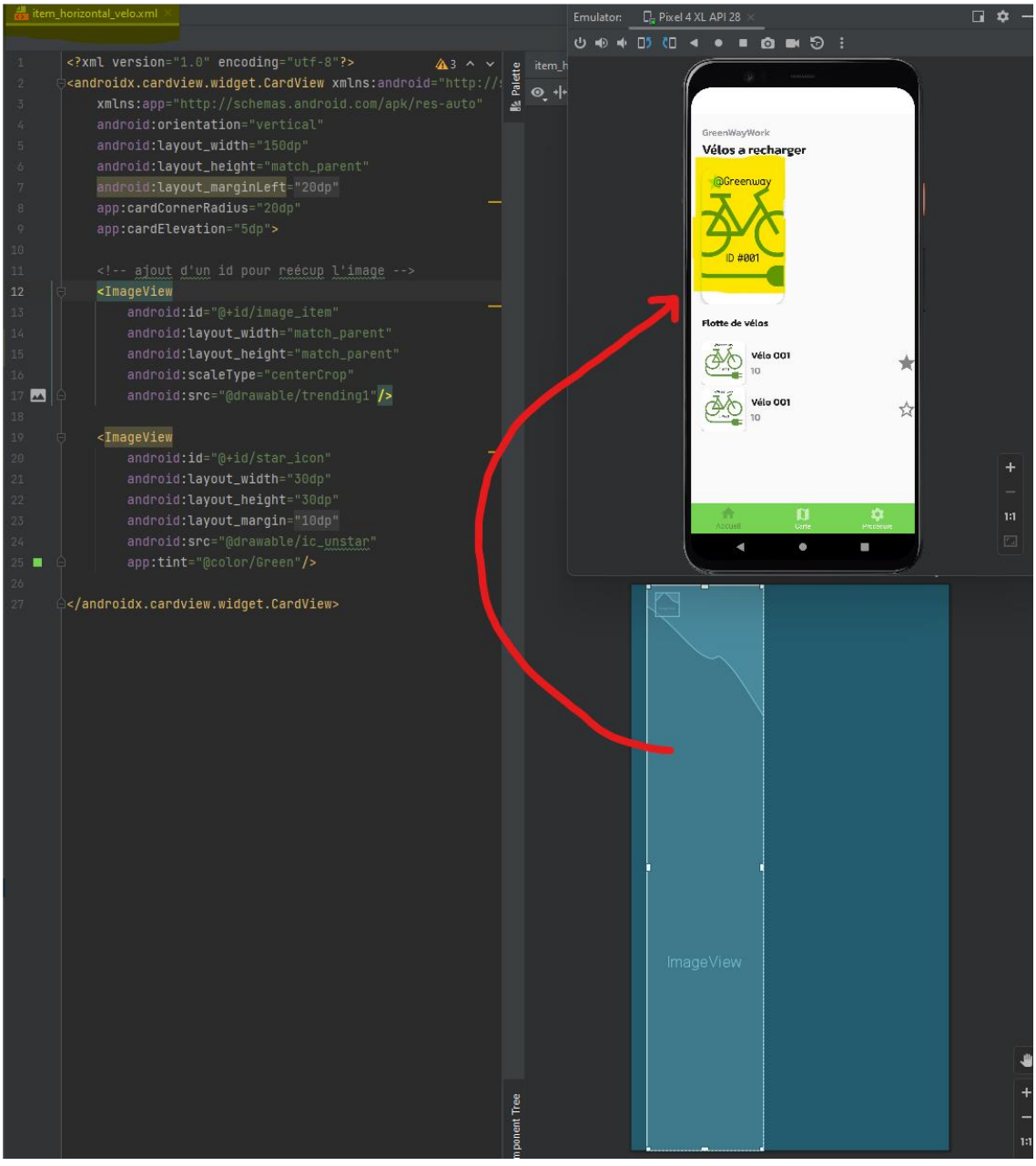
Définition de l'onglet procédure :

Créations de multiples boutons, Spinner et EditText.



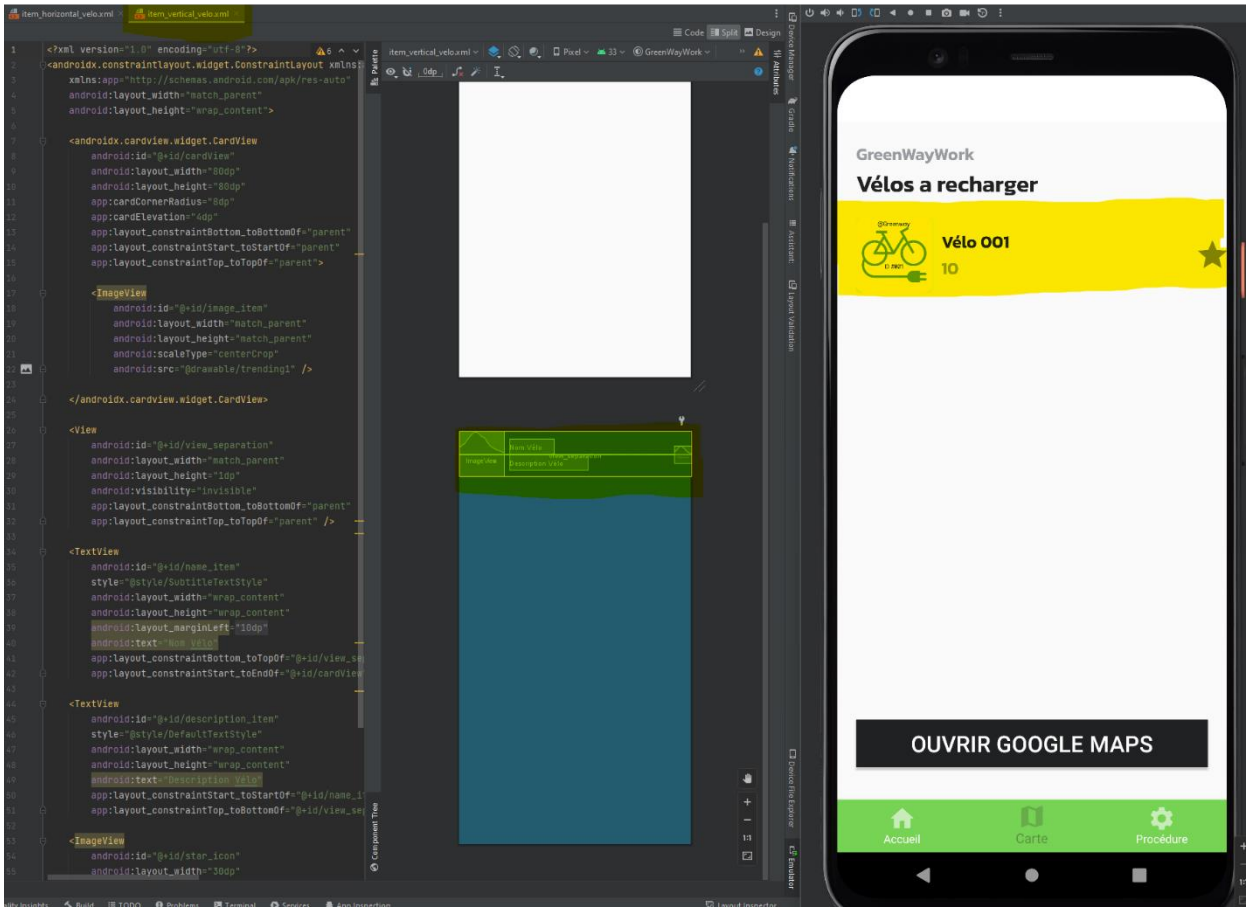
Définition item horizontal :

Après l'avoir créé je l'ai injecté dans l'onglet accueil.



Définition item vertical :

Une fois créé, il est aussi injecté dans l'onglet accueil et carte.



La navbar

Elle est définie de cette manière, je lui ai attribué des IDs pour retrouver les redirections dans mon code.

VeloRepository.kt | bottom_navigation_menu.xml | VeloModel.kt | VeloPopup.kt

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/home_page"
        android:title="Accueil"
        android:icon="@drawable/ic_home"/>

    <item
        android:id="@+id/carte_page"
        android:title="Carte"
        android:icon="@drawable/ic_map"/>

    <item
        android:id="@+id/procedure_page"
        android:title="Procédure"
        android:icon="@drawable/ic_proc"/>

</menu>
```

Elle est personnalisée dans le fichier themes.xml :

```xml
<!-- Créer un style pour la bottom navigation view -->
<style name="BottomNavigationStyle" parent="Widget.Design.BottomNavigationView">
    <item name="android:textSize">18sp</item>
    <item name="fontFamily">@font/kanit</item>
</style>

<!--personnaliser la navbar -->
<style name="BottomNavigationTheme" parent="Widget.Design.BottomNavigationView">
    <item name="android:textSize">18sp</item>
    <item name="android:fontFamily">@font/kanit</item>
</style>
```

Les redirections sont définies dans le fichier MainActivity avec les IDs crées :

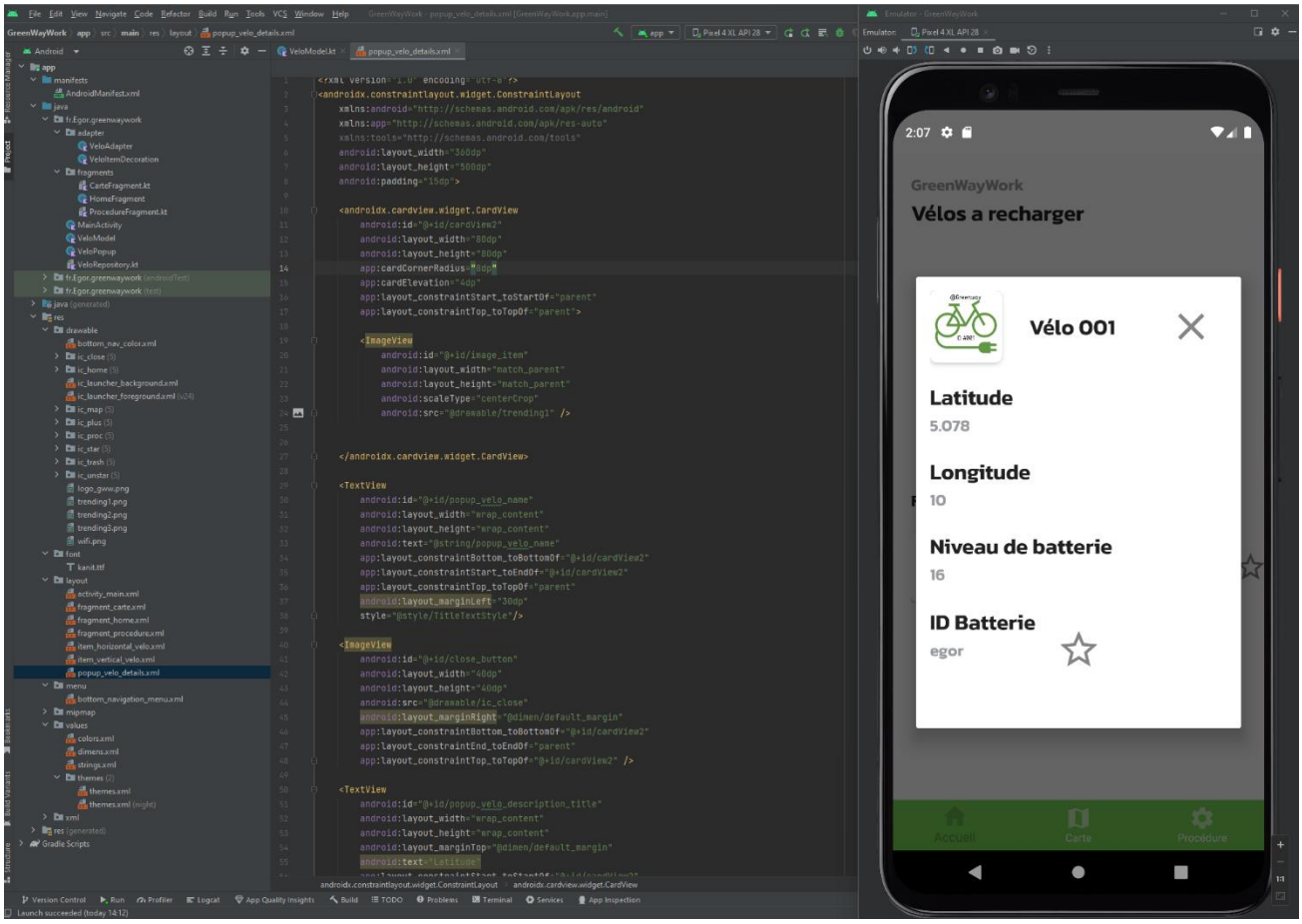
```
loadFragment(HomeFragment( context: this), "Vélos a recharger")

//importer la navbar
val navigationView = findViewById<BottomNavigationView>(R.id.navigation_view)
navigationView.setOnNavigationItemSelectedListener { it: MenuItem
    when(it.itemId)
    {
        R.id.home_page -> {
            loadFragment(HomeFragment( context: this), "Vélos a recharger")
            return@setOnNavigationItemSelectedListener true
        }
        R.id.carte_page -> {
            loadFragment(CollectionFragment( context: this), "Vélos a recharger")
            return@setOnNavigationItemSelectedListener true
        }
        R.id.procedure_page -> {
            loadFragment(ProcedureFragment( context: this), "Commencer la procédure")
            return@setOnNavigationItemSelectedListener true
        }
        else -> false ^setOnNavigationItemSelectedListener
    }
}
```

Pour arriver à ce résultat :

La popup :

A été définie dans le fichier popup_velo_details.xml :

Elle est composée des informations utiles, d'un bouton fermer, et d'un bouton "like" (remplacé par la suite par une icône de batterie plus intuitive). Ce système va me permettre de trier l'affichage pour afficher les vélos à recharger et toute la flotte de vélos.



Voici la classe qui permets de l'injecter :

```kotlin
package fr.Egor.greenwaywork

import ...

class VeloPopup(private val adapter: VeloAdapter,
                private val currentVelo: VeloModel
                 ) : Dialog(adapter.context) {

    //injecter le layout popup_velo
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        //on ne veut pas de titre
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        //injecter notre layout
        setContentView(R.layout.popup_velo_details)

        //initialise nos composants avec la bonne valeur
        setupComponents()
        setupCloseButton()
        setupStarButton()
    }

    //actualisation en temps réél
    private fun updateStar(button: ImageView){
        if (currentVelo.liked){
            button.setImageResource(R.drawable.ic_star)
        }
        else {
            button.setImageResource(R.drawable.ic_unstar)
        }

    }

    private fun setupStarButton() {
        //récupérer

        val starButton = findViewById<ImageView>(R.id.star_button)

        //remplacement du code redondant par la methode updateStar

        updateStar(starButton)




        //interaction avec la bdd

        starButton.setOnClickListener{ it: View!
            currentVelo.liked = !currentVelo.liked
            val repo = BikeRepository()
            repo.updateVelo(currentVelo)
            updateStar(starButton)

        }
    }
```

```
package fr.Egor.greenwaywork

import android.app.Dialog
import android.net.Uri
import android.os.Bundle
import android.view.Window
import android.widget.ImageView
import android.widget.TextView
import com.bumptech.glide.Glide
import fr.Egor.greenwaywork.adapter.VeloAdapter
```

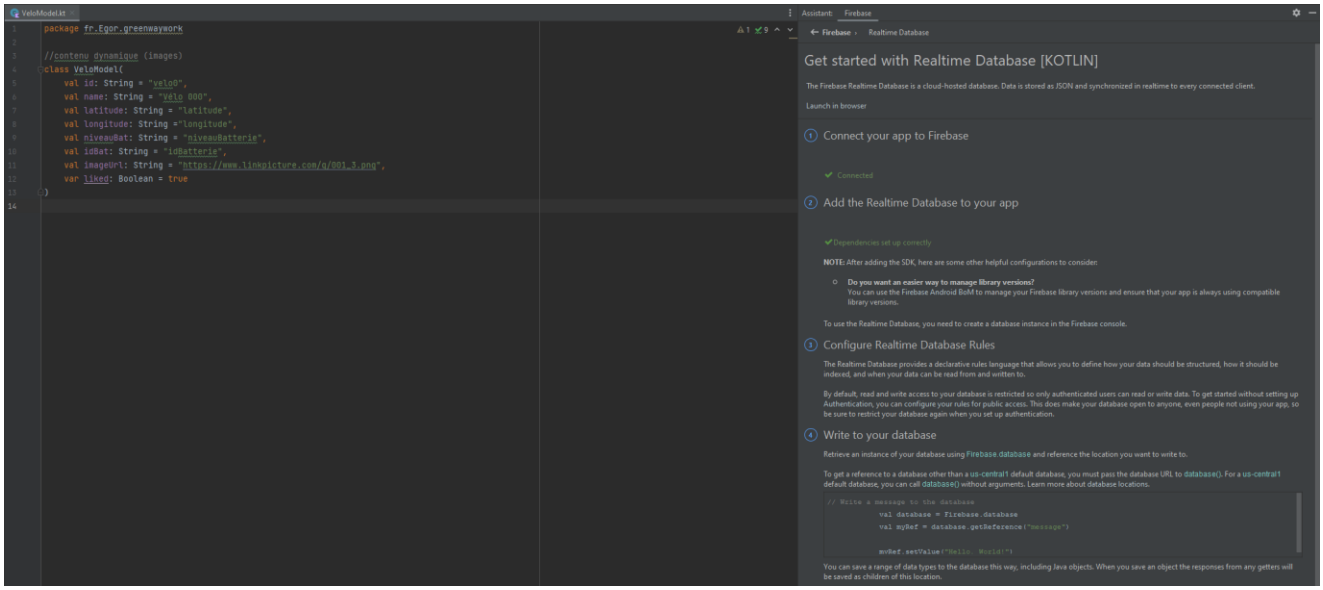
Le chargement des images a été effectué via Glide qui est une bibliothèque de chargement pour afficher des images de manière efficace et fluide dans les applications.

Quelques points forts de la bibliothèque :

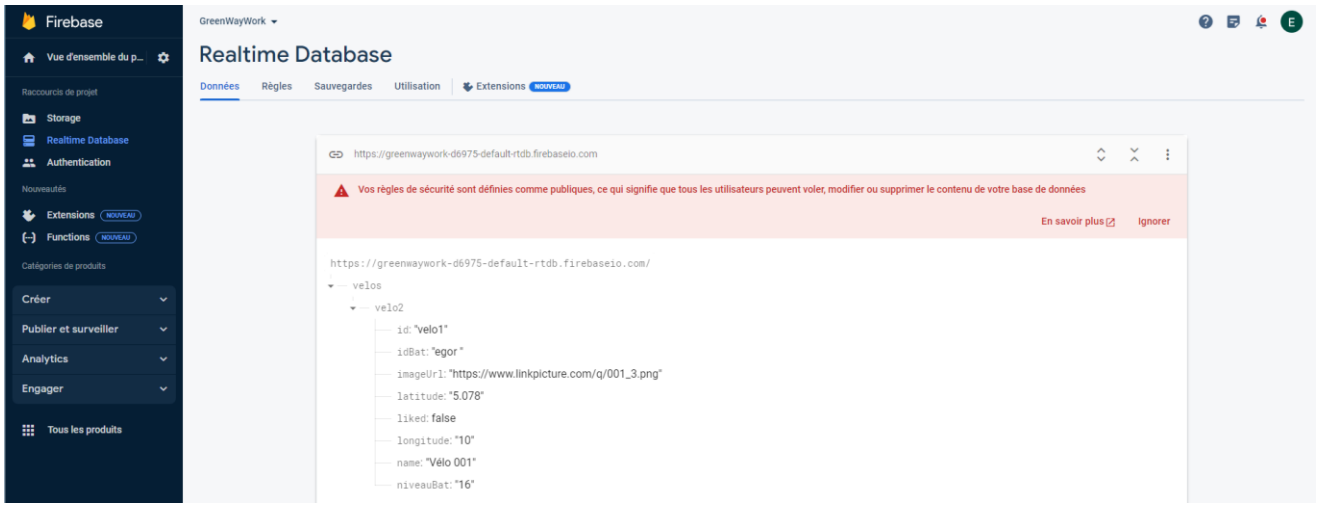
Chargement, mise en cache, intégration des images...

Base de données

J'ai intégré à mon application une base de données en temps réel en utilisant Firebase. Il y a une implémentation directe dans Android studio ce qui facilite la tâche.



Cette base de données me permet d'actualiser les informations de manière instantanée.



Voici le model local, il permet de gérer les informations dans mon application et de remplacer les valeurs si elles sont manquantes ou si la base de données n'est pas accessible :

VeloModel.kt

```kotlin
package fr.Egor.greenwaywork

//contenu dynamique (images)
class VeloModel(
    val id: String = "velo0",
    val name: String = "Vélo 000",
    val latitude: String = "latitude",
    val longitude: String ="longitude",
    val niveauBat: String = "niveauBatterie",
    val idBat: String = "idBatterie",
    val imageUrl: String = "https://www.linkpicture.com/q/001_3.png",
    var liked: Boolean = true
)
```

Pour finir Firebase prend en compte la gestion de multiples appareils, si 4 techniciens possèdent mon applications les informations seront actualisées pour tout le monde et en même temps.

Onglet Accueil

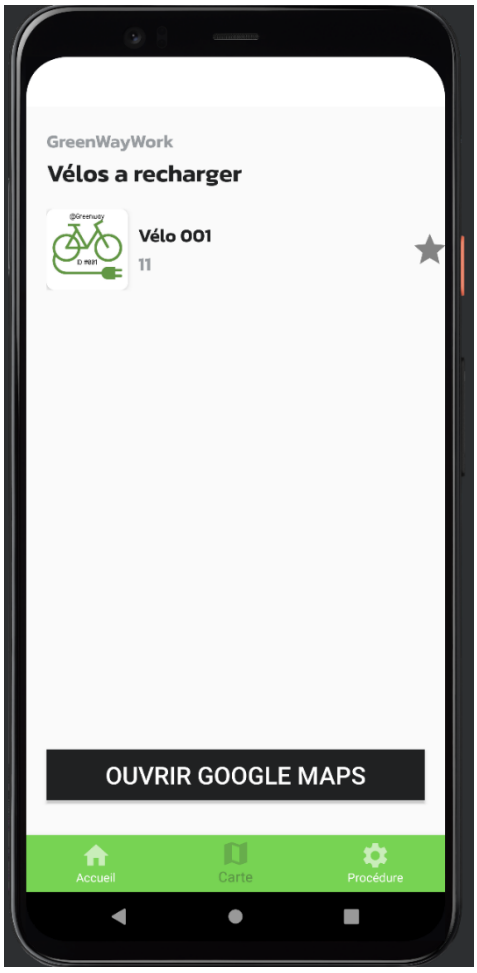
Cet onglet va permettre au technicien de visualiser la flotte de vélos ainsi que les vélos à recharger (batterie inférieure à 15 %).

L'étoile (remplacé par une batterie par la suite) indique le besoin d'intervention, elle est automatiquement désactivée après la résolution du problème.

Lorsqu'un technicien clique sur un vélo, il a accès à une popup qui répertorie toutes les données utiles :

Onglet Carte

a) Requête HTTP
b) Google Maps



Cet onglet va permettre au technicien de visualiser les vélos à recharger (batterie inférieure à 15 %). Il y a une seule interaction possible : ouvrir Google Maps.

Mais plusieurs processus tournent en arrière-plan, c'est ce que nous verrons dans cette partie.

a) Requête HTTP

Pour pouvoir récupérer les informations qui nous intéressent, nous devons requêter l'api de Altair en générant tout d'abord un bearer token.

Un bearer token utilisant le protocole OAuth est un jeton d'authentification. Il est couramment utilisé dans les applications web et les API pour vérifier l'identité d'un utilisateur et accorder l'accès à des ressources protégées.

Générer un token en vidéo (le bouton créer trajet a été ajoute pour la démonstration) :

https://www.youtube.com/watch?v=duSIXoSmiWs

La génération de ce token nous permet l'accès à ces informations durant 24h. J'utilise okhttp qui permet de faire un GET asynchrone. Je convertis le tout en json, puis je split mes données afin de les réinjecter dans ma base de données Firebase et les sauvegarder dans des variables.

Code de la requête, conversion json et split informations :

```
val database = FirebaseDatabase.getInstance()
private val client = OkHttpClient()

fun requestHttp() {
    //http request
    val request = Request.Builder()
        .url("https://api.swx.altairone.com/spaces/projectiotjf/categories/Velos/things-status/01GVYZZKC2BYW63F48WG3MMQRH")
        .header( name: "Authorization",  value: "Bearer ory_at_6Ybj-nsxao9UjrqvtWgKLqgHq7yuoRiWqKoSKkmZClk.JaB0ODXFPGzVqr5-136OCyoh4s82UcNu6xg4hjy2jqo")
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
        }

        override fun onResponse(call: Call, response: Response) {
            response.use { it: Response
                if (!response.isSuccessful) throw IOException("Unexpected code $response")

                for ((name, value) in response.headers) {
                    println("$name: $value")
                }
                val infos = (response.body!!.string())
                println(infos)

                //split infos
                val json = JSONObject(infos)

                val collection = json.getString( name: "collection")
                val description = json.getString( name: "description")
                val autonomieBatterie = json.getJSONObject( name: "properties").getInt( name: "Autonomie_batterie")
                val latitude = json.getJSONObject( name: "properties").getInt( name: "Latitude")
                val longitude = json.getJSONObject( name: "properties").getString( name: "Longitude")
                val space = json.getString( name: "space")
                val title = json.getString( name: "title")
                val uid = json.getString( name: "uid")


                latUrl = json.getJSONObject( name: "properties").getString( name: "Latitude")
                longUrl = json.getJSONObject( name: "properties").getString( name: "Longitude")


                println("collection: $collection")
                println("description: $description")
                println("autonomie_batterie: $autonomieBatterie")
                println("latitude: $latitude")
                println("longitude: $longitude")
                println("space: $space")
                println("title: $title")
                println("uid: $uid")
```

Mise à jour base de données :

```
//maj bdd
val myRef = database.getReference( path: "velos/velo2/")
val data = HashMap<String, Any>()
data["id"] =  "velo1"
data["imageUrl"] = "https://www.linkpicture.com/q/001_3.png"
data["name"] =  "Vélo 001"
data["niveauBat"] = "$autonomieBatterie"
data["latitude"] = "$latUrl"
data["longitude"] = "$longUrl"
data["idBat"] = "$idBat"
if (autonomieBatterie < 15){
    data["liked"] = true
}
else{
    data["liked"] = false
}
myRef.setValue(data) ^use
```

Cette requête est appelée dans le constructeur (onCreateView). Cela permet d'avoir les informations actualisées en temps réel.

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?

): View? {

    val view = inflater?.inflate(layout.fragment_carte, container,  attachToRoot: false)
    requestHttp()
```

Voici comment ça marche :
https://www.youtube.com/watch?v=Z6mmjGSsoQY

b) Google Maps

Je me suis servi de la longitude et la latitude récupérées précédemment afin de pouvoir avoir la position exacte via Maps.

Dans ce bout de code :

On trouve le bouton avec le ID que je lui ai attribué : maps_button.

Le setOnClickListener se traduit par : dès qu'on clique sur le bouton.

Et on ouvre Maps avec nos deux variables intégrées dans l'URL.

```
    val openMaps = view?.findViewById<Button>(R.id.maps_button)
    openMaps?.setOnClickListener { it: View!
        val openURL = Intent(android.content.Intent.ACTION_VIEW)
        openURL.data = Uri.parse( uriString: "https://www.google.com/maps/search/?api=1&query=$longUrl,$latUrl")
        startActivity(openURL)

    }
    return view

}
```

Une démonstration en vidéo :

https://www.youtube.com/watch?v=I4kNkTD1RBI

Onglet procédure

Dès que le technicien arrive sur place il va suivre la procédure prévue:

1. Se connecter en WIFI au vélo
2. Allumer les feux pour pouvoir le localiser
3. Changer la batterie (boutons factices)
4. Mettre à jour la base de données avec le nouvel identifiant de la batterie

1. Connexion WIFI au vélo

Voici le code qui permets d'ouvrir les paramètres de connexion Android :

La procédure est la même que pour le bouton Maps on change juste l'activité (WIFI_SETTINGS).

```
        //récuperer le bouton
        val pickupImageButton = view?.findViewById<Button>(R.id.upload_button)

        //lorsqu'on clique dessus ca ouvre les parametres wifi android
        pickupImageButton?.setOnClickListener { wifiConect() }

        return view
    }

    private fun wifiConect() {
 //code connexion wifi
        startActivity(Intent(Settings.ACTION_WIFI_SETTINGS))
    }
}
```

2. Allumer les feux

Pour allumer les feux j'ai décidé de passer par un serveur web hébergé sur un ESP32

Voici le programme écris en C++ :

Si la batterie est inférieure à 15 (consigne) le serveur web démarre et la LED est configurée en sortie.

```
#include <WiFi.h>

int batterie = 14;
const char* ssid     = "Greenway-Velo-001";
const char* password = "123456789";

WiFiServer server(80);
String header;


String outputPhareState = "off";

const int outputPhare = 2;

void setup() {

if (batterie < 15)
{


  Serial.begin(115200);
  pinMode(outputPhare, OUTPUT);
  digitalWrite(outputPhare, LOW);
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.println(IP);

  server.begin();
}
}
```

Ici un code permettant de recevoir des requêtes HTTP GET de l'application Android pour pouvoir allumer ou éteindre la LED.

```
void loop(){
  WiFiClient client = server.available();

  if (client) {
    Serial.println("New Client.");
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        header += c;
        if (c == '\n') {
          if (currentLine.length() == 0) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            if (header.indexOf("GET /26/on") >= 0) {
              outputPhareState = "on";
              digitalWrite(outputPhare, HIGH);
            } else if (header.indexOf("GET /26/off") >= 0) {
              outputPhareState = "off";
              digitalWrite(outputPhare, LOW);
            }


            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
            client.println(".button2 {background-color: #555555;}</style></head>");
            client.println("<body><h1><em> Greenway<em></h1> <h2>Allumer le feu avant du velo</h2>");
            client.println("<p>Phares - Statut " + outputPhareState + "</p>");
            if (outputPhareState=="off") {
              client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/26/off\"><button class=\"button button2\">OFF</button></a></p>");
            }

            client.println();
            break;
          } else {
            currentLine = "";
          }
        } else if (c != '\r') {
          currentLine += c;
        }
      }
    }
    header = "";
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

Puis le code Kotlin permettant de faire une requête GET sur l'IP de l'ESP32 :

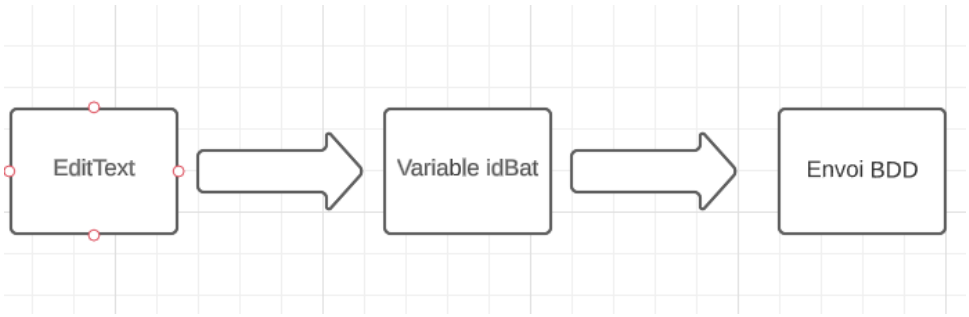
```kotlin
private fun wifiConnect() {
    startActivity(Intent(Settings.ACTION_WIFI_SETTINGS))
}

private fun toggleLedButtonClick() {
    ledState = !ledState

    val url = "http://192.168.4.1/26/${if (ledState) "on" else "off"}"

    Thread {
        try {
            val url = URL(url)
            val connection = url.openConnection() as HttpURLConnection
            connection.requestMethod = "GET"
            connection.connectTimeout = 5000
            connection.readTimeout = 5000

            val responseCode = connection.responseCode
            if (responseCode == HttpURLConnection.HTTP_OK) {
                // La requête GET a réussi
            } else {
            }

            connection.disconnect()
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }.start()

    btnLed?.text = if (ledState) "Éteindre la LED" else "Allumer la LED"
}
}
```

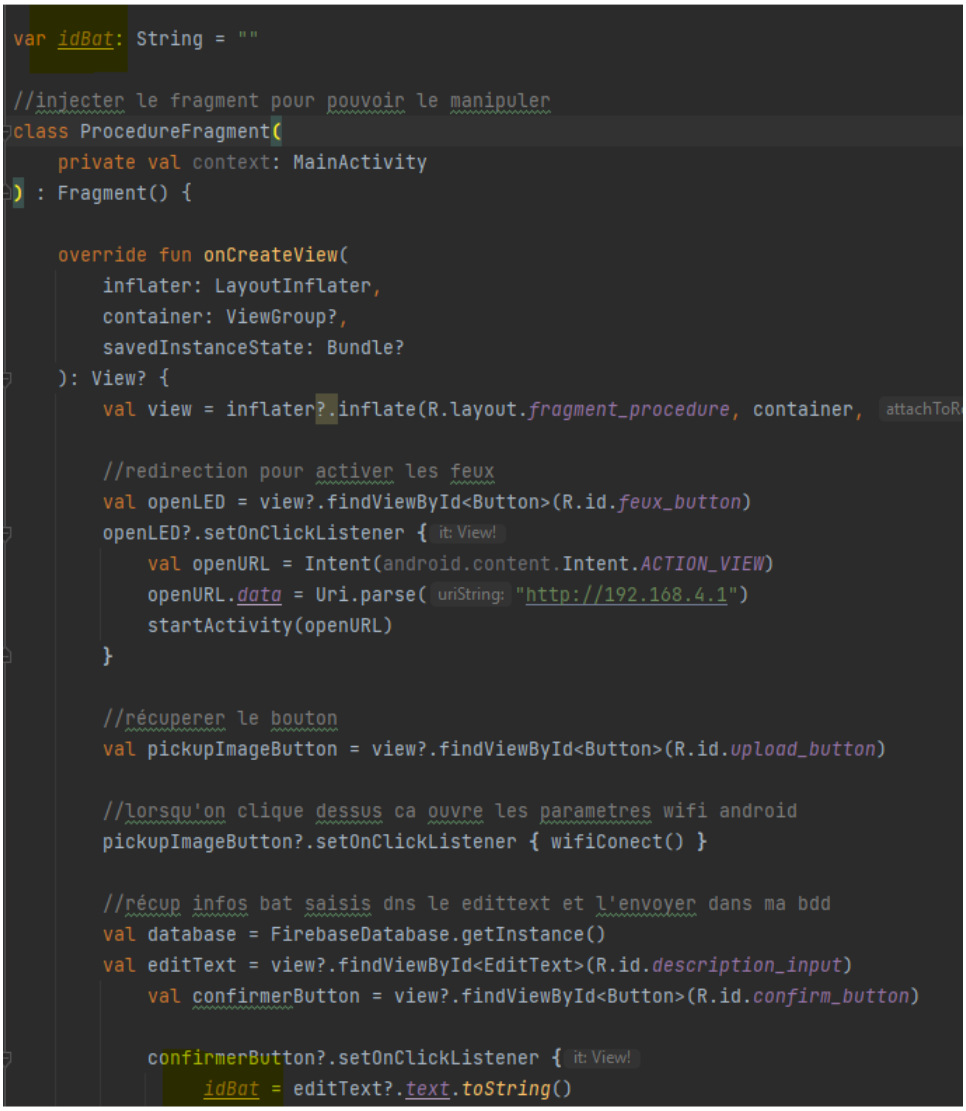
Voici la procédure en vidéo :
https://www.youtube.com/watch?v=GfsQQ7mH-Us

4. Mettre à jour la base de données

La mise à jour des donnes est effectuée une fois que le technicien a validé via le bouton confirmer.

Je vais récupérer le texte saisi dans le EditText , le stocker dans une variable nommée idBat et mettre à jour la base de données.





```
var idBat: String = ""

//injecter le fragment pour pouvoir le manipuler
class ProcedureFragment(
    private val context: MainActivity
) : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater?.inflate(R.layout.fragment_procedure, container,  attachToR

        //redirection pour activer les feux
        val openLED = view?.findViewById<Button>(R.id.feux_button)
        openLED?.setOnClickListener {  it: View!
            val openURL = Intent(android.content.Intent.ACTION_VIEW)
            openURL.data = Uri.parse( uriString: "http://192.168.4.1")
            startActivity(openURL)
        }

        //récuperer le bouton
        val pickupImageButton = view?.findViewById<Button>(R.id.upload_button)

        //lorsqu'on clique dessus ca ouvre les parametres wifi android
        pickupImageButton?.setOnClickListener { wifiConect() }

        //récup infos bat saisis dns le edittext et l'envoyer dans ma bdd
        val database = FirebaseDatabase.getInstance()
        val editText = view?.findViewById<EditText>(R.id.description_input)
            val confirmerButton = view?.findViewById<Button>(R.id.confirm_button)

            confirmerButton?.setOnClickListener {  it: View!
                idBat = editText?.text.toString()
```

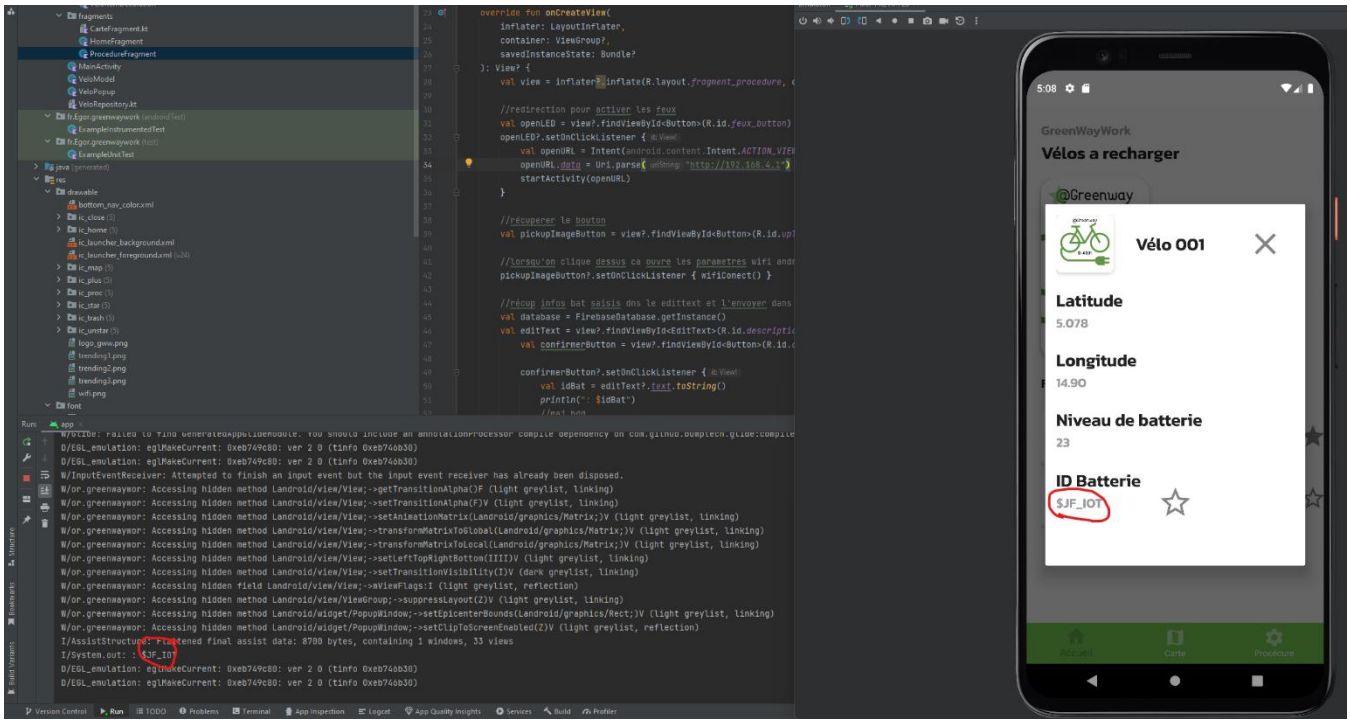
J’utilise une variable globale pour pouvoir mettre à jour la base de données en une fois. (La fonction appelée dans le constructeur)

```
println("uid: $uid")

//maj bdd
val myRef = database.getReference( path: "velos/velo2/")
val data = HashMap<String, Any>()
data["id"] =  "velo1"
data["imageUrl"] = "https://www.linkpicture.com/q/001_3.png"
data["name"] =  "Vélo 001"
data["niveauBat"] = "$autonomieBatterie"
data["latitude"] = "$latUrl"
data["longitude"] = "$longUrl"
data["idBat"] = "$idBat"
if (autonomieBatterie < 15){
    data["liked"] = true
}
else{
    data["liked"] = false
}
myRef.setValue(data) ^use
```

Interaction avec la base de données :

https://www.youtube.com/watch?v=OYRJpeEudkw

L’ID de la batterie sera visible sur le popup après validation :



Conclusion

En conclusion, j'ai choisi ce projet dans le but de combler mes lacunes en programmation et de m'améliorer. Grâce à cette expérience, j'ai pu acquérir de nouvelles compétences et connaissances significatives.

Tout d'abord, j'ai eu l'occasion d'apprendre le langage Kotlin, ce qui m'a permis d'élargir mes compétences de programmation et de découvrir de nouvelles possibilités de développement.

Ensuite, j'ai pu gérer une base de données Firebase, ce qui m'a donné l'opportunité de comprendre comment stocker et manipuler les données de manière efficace et sécurisée.

De plus, j'ai pu approfondir ma compréhension du protocole d'authentification OAuth, qui est fréquemment utilisé pour sécuriser les applications.

Enfin, j'ai appris à réaliser des requêtes HTTP, ce qui m'a donné la possibilité d'interagir avec des API externes et d'intégrer des fonctionnalités supplémentaires dans mon application.

Dans l'ensemble, ce projet a été extrêmement gratifiant et j'ai adoré le réaliser. J'ai pu combler mes lacunes en programmation, acquérir de nouvelles compétences techniques et développer une application fonctionnelle. Je suis reconnaissant d'avoir eu l'opportunité de partager mon rapport et je vous remercie d'avoir pris le temps de le lire.

Voici les liens ou vous pouvez retrouver l'entièreté du code de mon projet :

https://github.com/bvegor/Greenway

https://github.com/bvegor/ESP32_WIFI_AP

Voici mon mail si vous souhaitez que je génère un bearer token :

bvegorpro@gmail.com