



# Projet Trottinette Électrique Connectée

## **Revue Finale**



## Trottinette Électrique Tout Terrain(TTE) SXT 1000 XL

## Version 1.0

Projet Trottinette Électrique Connectée





Introduction	3
Expression du besoin	3
Présentation générale du projet	3
Contraintes matérielles et logicielles	4
Diagramme de déploiement	4
Présentation d'Android	5
Activité	5
Resources XML	6
La classe R	7
La classe layout	7
Lien entre XML et Activité	8
Thread	9
Handler	9
Planification prévisionnelle et tâches à réaliser	11
Diagramme de cas d'utilisation	14
	17
Étude préliminaire	15
IHM	15
Protocole de Transmission	16
VerifierTrame()	17
DecoderTrame()	19
Diagramme de classes	20
Rôle des classes	21
La classe TReception	21
La classe PeripheriqueBluetooth	21
La classe TrameTTE	21
La classe Trajet	21
La classe MainActivity	22
Étude détaillée	23
Scénarios	23
Le cas d'utilisation "Dialoguer avec la TEC"	23
Diagramme de séguence	23
Diagramme de classes partiel	25
Tests	26
Code du filtrage	27
Le cas d'utilisation "Visualiser les données de fonctionnement"	29
Diagramme de séquence	29
Diagramme de classes partiel	
Tests	
Le cas d'utilisation "Visualiser la localisation de TEC sur une carte"	.34
	01





Diagramme de Séquence	34
Initialisation	35
AfficherPositionTTE()	36
getVitesseMoyenneTTE()	37
calculVitesseMoyenne()	37
decoderTemp()	38
conversionTemps()	38
getConsommationTrajet()	39
getDistanceParcourue()	39
Diagramme de classes partiel	40
Tests	41
Tests de validation	45
Conclusion	45
Glossaire	46



## Introduction

## Expression du besoin

L'exploitant veut développer un système embarqué sur une trottinette électrique équipée de capteurs afin d'assister l'utilisateur sur son trajet. Un accessoire est monté sur le guidon pour permettre d'y poser un terminal mobile et d'accéder en temps réel aux données de la trottinette qui sont transmises par la trottinette.

## Présentation générale du projet

Nous devrons donc :

- Acquérir les données de fonctionnement de la trottinette
- Transmettre les données de fonctionnement de la trottinette via une communication sans fil
- Visualiser les données de fonctionnement reçues de la trottinette, la durée d'utilisation et l'autonomie sur le terminal mobile
- Géolocaliser la trottinette et la visualiser sur une carte de l'écran du terminal mobile
- Enregistrer les données de fonctionnement de la trottinette sur une carte SD (en option)
- La régulation de la vitesse et/ou l'arrêt de la trottinette (en option)
- Protection contre le vol (en option)

Les données de la trottinette seront transmises grâce à une communication Bluetooth entre le terminal mobile et la trottinette.

Les tâches à réaliser seront réparties sur 2 étudiants :

#### Étudiant EC :

HILLION Alexis devra acquérir des capteurs, établir un protocole de communication avec le terminal mobile, transmettre les données de fonctionnement vers le mobile.

Étudiant IR :

HACHETTE Alexandre devra établir le protocole de communication avec la TTE<sup>1</sup>, réceptionner les données de fonctionnement de la TTE, visualiser les données de fonctionnement de la TTE et la durée d'utilisation sur l'écran du terminal mobile, calculer et afficher l'autonomie pour un parcours, afficher et actualiser la carte avec la géolocalisation de la TTE.

<sup>&</sup>lt;sup>1</sup> TTE : Trottinette tout Terrain Électrique Projet Trottinette Électrique Connectée





## Contraintes matérielles et logicielles

#### Les ressources matérielles mises à disposition pour mener à bien le projet sont :

Désignation	Caractéristiques techniques	A cquisition	Existant
SXT 1000 XL	Trottinette Electrique Tout Terrain (TTE) Plomb 48V 12Ah de la marque SXT		×
SE ATMEL	Carte de développement ATMEL (ou équivalente)		×
TERMINAL	Terminal mobile sous Android		×
BLE	Module Bluetooth Low Energy		×
CAPTEURS	Ensemble de capteurs à définir	×	
SD	Carte ( <i>Secure Digital</i> ) carte mémoire amovible de stockage de données numériques minimum 1GO (en option)		×

#### Ainsi que les logiciels sont :

Environnement de développement	© ATMEL Studio
Systeme d'exploitation du terminal mobile	Android
Système de gestion de bases de donnée relationnelles	SQLite3
Gestion et administration de bases de donnees	Sqliteman ou SQLiteManager
Atelier de génie logiciel	Bouml 7.4
Logiciel de gestion de versions	subversion
Generateurs de documentation	Doxygen version 1.8.11
Gestionnaire de projet	Planner (version 0.14.5)

#### Diagramme de déploiement

Le SE (Système Embarqué) est construit autour d'une carte Atmel sur laquelle est reliée les différents capteurs pour l'acquisition des données de fonctionnement (partie EC).



Le terminal mobile (partie IR) fonctionne sous Android et communique avec le SE par une communication Bluetooth.

Cela implique qu'une communication est nécessaire et qu'il nous faut un protocole de communication entre la trottinette et notre terminal mobile.





## Présentation d'Android

**Android** est un système d'exploitation Open Source de la société Google pour terminal mobile (téléphone, tablette, ...).

Pour développer des applications pour ce système, il est nécessaire de disposer du **SDK** (*Software Development Kit*) Android et d'un environnement de développement. Pour le projet, nous utiliserons **Android Studio**. Le langage de programmation utilisé est **Java**.

#### Activité

Une **activité** est la composante principale d'une application Android. Elle représente à la fois le code et les interactions avec l'interface graphique.

On appellera donc les différentes fenêtres d'une application une **activité**. Une activité prend tout l'écran et il ne peut donc y en avoir qu'une à la fois. Dans notre programme, nous ne disposons que d'une seule activité : **MainActivity**.

Une activité est associée à un fichier ressource XML qui fournit l'aspect graphique de l'interface sous forme de *View*. Une activité permet aussi de contenir des information sur l'état actuel de l'application, ces information s'appellent le *context*.

Pour créer une activité principale, il suffit de déclarer une nouvelle classe héritant d'une classe mère **AppCompatActivity** (ou **Activity**) :

Le point d'entrée d'une application Android est la méthode **onCreate()** (équivalent du main() dans d'autres environnements) d'une activité. Une application Android a un cycle de vie décrit dans le schéma suivant :







Notre activité passe donc par une phase onCreate sans faire de onStart car nous n'utilisons pas de système de onPause et onStop, elle continue sur un onResume a ce moment l'activité est fonctionnel l'utilisateur peut cliquer sur les boutons et sélection des objet dans les liste qui lui sont dédiés l'application se ferme après un onDestroy.

#### **Resources XML**

En Android des layouts en XML (*Extensible Markup Language*) nous permettent de gérer le côté graphique de notre application. Pour déclarer des ressources, on passe très souvent par le format XML.

Le XML est un langage de balise simple qui ressemble au HTML. Les langage de balise s'oppose au langage de programmation tel que le java ou le C++ car il ne donne pas d'ordre à l'ordinateur pour effectuer des calculs mais ont juste pour objectif de mettre en forme l'information de façons à ce que l'on puisse la lire. Pour mettre en forme notre information on utilise des balises. Par exemple:

En android 5 types de ressources sont utilisées :

- **Drawable** qui contient toutes les images et fichiers de dessins.
- Layout qui contient nos interfaces graphique et nos différente vues.
- Menu qui contient toutes les déclarations d'éléments pour confectionner des menus.
- Raw qui contient tout les ressources de format.
- **Values** qui contient les valeur pour les chaînes de caractère, les dimension ou encore les couleur ...

Projet Trottinette Électrique Connectée



#### La classe R

BTS SN

La Salle

des Écoles Chrétienne

La classe R est une classe nécessaire pour accéder a nos ressources XML depuis notre code en java. Elle est situe dans le fichier R.java .

Cette classe n'aura pas besoin d'être instancié comme nous allons le voir après.

#### La classe layout

La classe layout est un classe dite **interne** car elle a été déclaré dans une autre classe. Pour y accéder, il faut donc faire référence à la classe qui la contient de ce cas si la classe R. la classe layout est de type public static final et a le nom layout.

- Public car elle doit être accessible par tout le monde sans restriction.
- Static, car elle ne dépend pas de l'instanciation de la classe R ( nous ne somme pas obligé de créer un objet de type R)
- Final signifie que l'on ne peut pas avoir de classe dérivé de layout.

Le fichier **content\_main\_ihm.xml** décrit la structure de l'IHM principale en définissant les ressources graphiques :

#### Dans la ressource content\_main\_ihm.xml

```
[...]
```

```
<GridLayout
```

```
android:id="@+id/layoutDonneesTrottinette"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:layout_marginBottom="5sp"
     android:layout_marginLeft="10dp"
     android:layout_marginRight="10dp"
     android:layout_marginTop="5sp"
     android:columnCount="2"
     android:rowCount="3">
     <TextView
          android:id="@+id/affichageVitesse"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_column="0"
          android:layout_gravity="start"
          android:layout_marginRight="20dp"
          android:layout_row="0"
          android:text="Vitesse : -- km/h"
          android:textSize="20sp"/>
      [...]
 </GridLayout>
...]
```





Ici, le **TextView** est placé dans un **GridLayout** qui permet un positionnement en grille. Le TextView permet l'affichage d'un texte dans l'IHM, ici l'id **affichageVitesse** affichera la vitesse de la trottinette en km/h :

Palette	Q, #+ I*		🛇 + 🗌 Nexus 4 + 🛎 25 +	AppTheme      GLanguage      I		Properties	Q 🚅	\$\$- →!
All	Ab TextView	+ 1		⊖ 53% ⊕ [	ः 📲 📴	ID	affichageVitesse	
			0 100	200 300	400	layout_width	wrap_content	
	initi de la companya					layout_height	wrap_content	
Componer	to the stand and the stand	<u> </u>		20.700		TextView		
~	avoutDoppeesTrottinette (GridLavout)			▼ 27:00		text	@string/vitesse	
	Ab affichageVitesse (TextView) - "@string/vitesse"	E I	Trottinette STX			≁ text		
	Ab affichageChargeBatterie (TextView) - "@string,	E.				contentDescription		
	Ab affichageDistanceParcourue (TextView) - "@str	i – É	Vitesse : km/h	Vitesse moy. : km/h		v textAppearance	Material.Small	
	Ab affichageVitesseTrajet (TextView) - "Vitesse mo	Y	Batterie : %	Consommation : %		fontFamily	sans-serif	
	Ab affichageChargeBatterieTrajet (TextView) - "Co	É.	Distance : m	Parcours : m		typofaco		-
_	Ab affichageDistanceParcourueTrajet (TextView) -	E				typerace		
<b>▼ =</b>	ayoutCarte (LinearLayout) (horizontal)	200				textbize	20sp	
	avoutl og (Lipearl avout) (borizontal)					lineSpacingExtra	none	
	≡ listview					textColor		
. ≡	layoutSpinnerListeAppareils (LinearLayout) (horizo	F				textStyle	BITr	
	Ab listeTTE (TextView) - "Liste Trottinette :"	300				textAlignment	IE E 포 3 3	
	≓ spinnerListeAppareils					Favorite Attribute:	5	
. ■	layoutBoutons (LinearLayout) (horizontal)		Item 1	1		layout_gravity	start	
	•• boutontrajet (Button)		Sub Item 1			visibility	none	
	🛎 boutoncommunication (Button)	400	ltem 2 Sub Item 2					
		200	Liste Trottinette :	•				
			<b>°</b> •	<b>○</b> □	-7			

Lien entre XML et Activité

Pour faire le lien entre le layout et le code en java on utilise la ligne suivant dans le fichier XML :

```
tools:context="com.example.iris.myapplication.MainActivity"
```

Ensuite dans notre code Java on pourra utiliser les ressources que nous avons placées dans le layout à l'aide de leur **id**.

```
[...]
private TextView affichageVitesse;
[...]
affichageVitesse = (TextView) findViewById(R.id.affichageVitesse);
[...]
affichageVitesse.setText("Vitesse : " + trame.getVitesse() + " km/h");
[...]
```

Nous pouvons grâce à ce morceau de code **modifier le texte** que contient notre label en ajoutant la valeur que nous recevons lors de la transmission des données par notre trottinette.





#### Thread

Un thread est un fil d'exécution dans un programme. Il permet d'exécuter du code en parallèle.

Le thread UI (*User Interface*) est le fil d'exécution d'une activité. Il est responsable de l'affichage et des interactions avec l'utilisateur et surtout c'est le seul thread qui doit modifier l'affichage.

Par contre, on ne peut pas effectuer des traitements consommateurs en temps dans le thread UI car celui-ci se "figerait" et ne répondrait plus aux actions de l'utilisateur. Si une activité réagit en plus de cinq secondes, elle sera tuée par l'ActivityManager d'Android qui la considérera comme morte.

Il faudra donc créer et exécuter des threads d'arrière-plan pour les traitements lourds. Nous utilisons dans notre programme un thread notamment pour la réception de trame et pour la connexion et déconnexion des modules bluetooth.

En Java, il y a plusieurs façons de créer et exécuter un thread. Le principe de base revient à dériver (*extends*) une classe **Thread** ou à implémenter (*implements*) l'interface **Runnable** et à écrire le code du thread dans la méthode run(). Ensuite, on appellera la méthode start() pour démarrer le thread et la méthode stop() pour l'arrêter.

Le thread UI (*User Interface*) de l'activité principale est le seul thread qui peut modifier l'affichage. Il faudra donc mettre en place une communication en threads pour interagir avec l'IHM. On utilisera les services de la classe **Handler**.

#### Handler

Lorsqu'un thread doit interagir avec l'IHM, on utilisera la classe **Handler** pour communiquer avec le thread UI qui a la responsabilité des interactions avec l'IHM.

Le Handler est associé à l'activité principale (qui le déclare) et travaille au sein du thread UI. Le handler se chargera de mettre à jour l'IHM. Le thread peut communiquer avec cet handler au moyen de messages :

- le thread récupère un objet Message du Handler par la méthode Message.obtain(). Il peut ensuite ajouter des données en utilisant un objet Bundle ;
- le thread envoie le message au Handler en utilisant la méthode **sendMessage()** ;
- La présence d'un message déclenchera l'exécution de la méthode handleMessage() du Handler qui lui permettra de traiter le message et les données contenues.





Dans le thread TReception	Dans l'activité MainActivity
<pre>Handler gestionnaire; " Message message = Message.obtain(); message.what = PeripheriqueBluetooth.CODE_CONNEXION; Bundle paquet = new Bundle(); paquet.putString("nom",getNom()); paquet.putString("adresse",getAdresse() ); paquet.putInt("etat",CODE_CONNEXION); paquet.putString("donnees",""); message.setData(paquet); gestionnaire.sendMessage(message);</pre>	<pre>final private Handler gestionnaire = new Handler() {     public void handleMessage(Message msg)     {         Bundle b = msg.getData();         switch(b.getInt("etat"))         {             case     PeripheriqueBluetooth.CODE_CONNEXION:</pre>





## Planification prévisionnelle et tâches à réaliser

Pour organiser notre travail nous avons donc eu recours à un diagramme de  $Gantt^2$  :

TDÉ		Travai	État o	Aorica	Compine 7	Compine R	mars 20	)18 Comain
-	<ul> <li>Projet TTE( Trottinette )</li> </ul>	17j Sh	0					
1.1	▼ Prise en main du projet	3j 8h	100					
1.1.1	Prise en charge du cahier des charges	2h	100	-				
1.1.2	Etablir le Gant	Sh	100	•				
1.1.3	Etablir les diagrammes UML	1j 1h	100	<b></b>				
1.1.4	Lister les fonctionnalitées necessaire en fin de projet	1h	100	T	•			
1.1.5	Reunion avec client	2h	100		÷			
1.1.6	Faire une veille technologique	7h	100		<b>.</b>			
1.1.7	Prise en main de la programmation sur android	8h	100		9			
1.2	▼ Mise en Oeuvre	2j	100		-			
1.2.1	Etablir le protocole de transmission	2h	100			•		
1.2.2	Etablir la connexion bluetooh	1j 7h	100					
1.2.3	Revue n°2	N/D	100			•		
1.3	▼ Programmation TTE	3j 8h	100			-1		
1.3.1	Programmation de la liaison bluetooh	1j 7h	100					
1.3.2	Programmer l'IHM	Sh	100					
1.3.3	Programmer la carte	4h	100					
1.3.4	Progammer le trajet	1j 1h	90					
1.4	▼ Finalisation	4j 3h	100					
1.4.1	TesterL'ihm	1j 1h	100					
1.4.2	Mise en place du système	1j 1h	100					
1.4.3	Preparation du document a rendre	2j 1h	100					
1.4.4	Revue n°3	N/D	0					-
1.4.5	Remise du projet	N/D	0					
1.5	▼ Bonus	3j 4h	0					
1.5.1	Rajout de Widget( compteur de vitesse, chronometre)	1j бh	0					
1.5.2	Rajout d'une base de donnée pour enregistrer	1j 7h	0	-				

<sup>2</sup> Diagramme de Gantt : c'est un outil permettant de visualiser dans le temps les diverses tâches composant un projet.

Projet Trottinette Électrique Connectée



Session 2018









Dans ce diagramme de gantt on peut observer que nous avons divisé notre travail en 5 itérations :

- Prise en main du projet
- Mise en oeuvre
- Programmation TTE
- Finalisation
- Bonus

Avec une première itération d'analyse pour mieux connaître le projet et analyser le temp de travail à fournir sur chaque fonctionnalité de notre application. Ensuite nous avons effectué une étape importante pour établir le protocol de communication et la connexion bluetooth qui devait nous emmener a la 2eme revue de projet. Nous pouvions alors attaquer la face de programmation pour établir la connexion bluetooth.

Les tâches à réaliser par l'étudiant HACHETTE Alexandre sont :

- Mettre en oeuvre un protocole de communication avec la TTE
- Réceptionner les données de fonctionnement de la TTE
- Visualiser les données de fonctionnement de la TTE et la durée d'utilisation
- Calculer l'autonomie pour un parcours et l'afficher
- Afficher et actualiser la carte avec la géolocalisation de la TTE





## Diagramme de cas d'utilisation



Une communication Bluetooth entre la trottinette et le terminal mobile est nécessaire pour pouvoir visualiser les données de fonctionnement de la TEC<sup>3</sup>. La visualisation de sa localisation nécessite une géolocalisation par le terminal mobile. L'utilisateur pourra éventuellement visualiser son trajet.

A partir du terminal mobile, grâce à l'application, l'utilisateur pourra donc :

- Visualiser les données de fonctionnement (vitesse en km/h, charge de la batterie en % et la distance parcourue en mètres)

- Visualiser la position de sa trottinette sur une carte
- Visualiser son trajet

L'utilisateur pourra lorsqu'il active le mode trajet visualiser les données et le tracé de son trajet depuis le début de l'activation du bouton. Il ne pourra voir que le tracé qu'il a effectué depuis que le bouton est activé.

<sup>&</sup>lt;sup>3</sup> TEC : Trottinette Électrique Connectée (la TTE avec le système développé) Projet Trottinette Électrique Connectée





# Étude préliminaire IHM







## Protocole de Transmission

#### Trame de données



La trame reçu doit donc contenir:

- Un caractère de début : \$
- Un type de protocole : TEC
- Un numéro de trottinette : X
- La vitesse : en km/h
- La charge : en %
- La distance parcourue : en m
- Un caractère de fin de trame : \*

Exemple de trames :

Trames Reçues
\$TEC1, <mark>25</mark> ,70, <b>1525</b> *
\$TEC1, <mark>28</mark> ,69, <b>1750</b> *

Décodage :

TEC1	25	70	1525
TEC1	28	69	1750
ID TTE	Vitesse (km/h)	Charge restante (%)	Distance parcourus (m)

Pour traiter cette trame, on utilisera un type **String** afin d'utiliser une fonction **split()** qui permet de mettre chaque donnée dans un tableau.

Projet Trottinette Électrique Connectée



Les trames seront envoyées toutes les 1000ms (1s) par défaut. La périodicité est fixée par l'étudiant EC.

On peut donc établir la fonction VerifierTrame() puis DecoderTrame() de la classe TrameTTE.

- VerifierTrame()

VerifierTrame() a pour but de vérifier si la trame est bien conforme au protocole de transmission.

```
VerifierTrame()
private Boolean VerifierTrame(String trame) throws InterruptedException
  if(trame.length() != 0)
   {
       if(trame.startsWith(DEBUT_TRAME))
       {
           if(trame.startsWith(DEBUT_TRAME + TYPE_PROTOCOLE))
           {
               if(trame.contains(FIN_TRAME))
               {
                   return true;
               }
               else
                   Log.e("<TrameTTE>","<FiltrerTrame> Trame non valide pas
de caractere de Fin");
           }
           else
               Log.e("<TrameTTE>","<FiltrerTrame> Trame non valide mauvais
type de trame");
       }
       else
           Log.e("<TrameTTE>","<FiltrerTrame> Trame non valide caractere
de debut different");
   }
  else
   ł
       Log.e("<TrameTTE>","<FiltrerTrame> Trame vide");
   return false;
}
```





Exemple : **\$TEC1**,25,70,1525\*

On va successivement vérifier :

• si la trame est vide ou non :

```
if(trame.length() != 0)
{
    [...]
}
else
{
    Log.e("<TrameTTE>","<FiltrerTrame> Trame vide");
}
```

• Si la trame possède le caractère de début \$ :

```
if(trame.startsWith(DEBUT_TRAME))
{
    [...]
else
    Log.e("<TrameTTE>","<FiltrerTrame> Trame non valide caractere de debut
different");
```

• Si la trame est du type TEC :

```
if(trame.startsWith(DEBUT_TRAME + TYPE_PROTOCOLE))
{
    [...]
}
else
    Log.e("<TrameTTE>","<FiltrerTrame> Trame non valide mauvais type de
trame");
```

• Si la trame possède le caractère de fin \* :

```
if(trame.contains(FIN_TRAME))
{
    return true;
}
else
    Log.e("<TrameTTE>","<FiltrerTrame> Trame non valide pas de caractere
de Fin");
```





#### - DecoderTrame()

Avant de décoder les données de la trame, on vérifie si elle possède le nombre correct de champs :

```
DecoderTrame()
```

```
public boolean DecoderTrame(String trame)
   trame = trame.substring(0, trame.length()-1);
   List<String> champs = new
ArrayList<String>(Arrays.asList(trame.split(",")));
   if(champs.size() == 4)
   {
       int position = 0;
       Vitesse = Integer.parseInt(champs.get(++position));
       ChargeBatterie = Integer.parseInt(champs.get(++position));
       DistanceParcourue = Integer.parseInt(champs.get(++position));
       String id = trame.substring(champs.get(0).indexOf("C") + 1,
champs.get(0).length());
       IdTEC = Integer.parseInt(id);
       return true;
   }
   return false;
}
```

Exemple : *\$TEC1*, 25, 70, 1525\*

On commence par enlever le caractère de fin de trame avec :

```
trame = trame.substring(0, trame.length()-1);
```

Puis on utilise les fonctions associés à la classe String, telles que split() qui permet de diviser une donnée délimitée par des virgules :

```
List<String> champs = new
ArrayList<String>(Arrays.asList(trame.split(",")));
```

On termine par extraire chaque donnée :

```
if(champs.size() == 4)
{
    int position = 0;
    Vitesse = Integer.parseInt(champs.get(++position));
    ChargeBatterie = Integer.parseInt(champs.get(++position));
    DistanceParcourue = Integer.parseInt(champs.get(++position));
    String id = trame.substring(champs.get(0).indexOf("C") + 1,
    champs.get(0).length());
    IdTEC = Integer.parseInt(id);
    return true;
}
```

Projet Trottinette Électrique Connectée





## Diagramme de classes

AppCompatActivity	Handler	Thre	ad /			
	Ę		<u> </u>	24		
Ť		1	1			
MainActivity	/	Peripheriq	ueBluetooth	1		TReception
TAG : String		nom : String				handlerUI : Handler
bluetoothAdapter : BluetoothAdapter		adresse : String				tini : boolean
noms : String		gestionnaire : Handler				nbTrames : long
devices : BluetoothDevice		peripherique : BluetoothDevice				link : long
trajetEncours : Boolean		socket : BluetoothSocket				envoye : boolean
ID_DISCOVERABLE_INTENT : int		fluxEntree : InputStream			<u>م</u>	inactif : boolean
REQUEST CODE ENABLE BLUETOOTH : int		fluxEnvoie : OutputStream			<b>•</b>	TReception(inout h : Handler)
spinnerListePeripheriques : Spinner		CODE CONNEXION : int				run() : void
mListView : ListView		CODE RECEPTION : int				recevoirTrame() : void
journal : String		CODE DECONNEXION : int				attendreProchaineLecture(in millis : int) : void
mListLog : String		CODE INACTIF : int				attendreConnexion() : void
affichageDistanceParcourue : TextView	peripherique	CODE ACTIF : int				viderBuffer() : void
affichageChargeBatterie : TextView	A.	TAILLE BUFFER : int				arreter() : void
affichageVitesse : TextView		PROCHAINE LECTURE : int				
affichageDistanceParcourueTrajet : TextView		PERIODE INACTIVE : int				
affichageConsommationTrajet : TextView		PeripheriqueBluetooth(inout peripherique :	BluetoothDe	vice, inout gestionnaire : Handler)		BluetoothSocket
affichageVitesseMoyenneTrajet : TextView		getDevice() : BluetoothDevice			$\diamond \rightarrow$	
boutonCommunicationTTE : Button		getNom() : String				
boutonTrajet : Button	nerinheriques	getAdresse() : String				
boutonParametrage : Button	periprienques	estConnecte() : boolean				
myOpenMapView : MapView	< <lisi>&gt;</lisi>	setNom(in nom : String) : void				
myScaleBarOverlay : ScaleBarOverlay		toString() : String				
mCompassOverlay : CompassOverlay		connecter() : void				
mRotationGestureOverlay : RotationGestureOverlay		envoyer(in data : String) : void				
myLocationOverlay : DirectedLocationOverlay		deconnecter(inout fermeture : boolean) : bo	olean			
locationManager : LocationManager		TrameTTE	2		-	
fournisseur : String		Iramette				
geocoder : Geocoder		Id IEC : Int				
etatLocalisation : int		Vitesse : Int				
mAzimuthAngleSpeed : float		ChargeBattene : Int				
mSpeed : double		DistanceParcourue : Int				
ecouteurGPS : LocationListener		TYPE PROTOCOLE China				
bluetoothReceiver : BroadcastReceiver	trama	FIN TRAME : Ching				
gestionnaire : Handler	uame	TerretTE 0	e			
onCreate(inout savedInstanceState : Bundle) : void		Traine TTE()			Trajet	
onResume() : void		Vatien Trame (in trame : String) : Boolean		encours : Boolean		
onDestroy() : void		DecoderTrame(in trame : String) : Boolean		vitesseMoyenneTTE : String		
initialiserActivite() : void		Decoder frame (in trame : String) : boolean		chargeInitialeTTE : int		
gererBoutons() : void		getChargeBatterie() : int		tempsTrajet : String		
onClick(inout view : View) : void		getonargeDattere() . int		distanceParcourueInitiale : int		
journaliser(in text : String) : void		getIdTEC() : int		distanceParcourueTrajet : doubl	е	
initialiserCarte() : void		generico() . Int		heureDebut : String		
InitialiserLocalisation(): Void			traint	Trajet()		The second s
arreterLocalisation(): Void			uajet	Trajet(in chargeInitialeTTE : int, i	n distanceParco	ourueInitiale : int)
InitialiserCommunication(): void				demarrerTrajet() : void		
activerbluetooth(): Void				arreterTrajet() : void		
affeterbluetootn(): vold				getVitesseMoyenneTTE() : String	3	
afficherDenneneTTE(ingut ek : Beeleen)				getConsommationTrajet(in charg	eTTE : int) : int	
amonerDonnees I I E (Inout ok : Boolean) : Void				getEtatTrajet() : Boolean		
onActivityResult(in requestCode : Int, in resultCode : Int, inout data : Intent) : void				getDistanceParcourue(in distance	eParcourue : in	t) : int
getApplication/name(inout context : Context) : String				calculVitesseMoyenne() : String		
getPosition (in auresse : Sting) : Int				decoderTemp(in tempsAConvert	ir : String) : doul	ble
serbourons(in position : Int) : Volo				conversionTemps(in Heure : Stri	ng, in minute : S	tring, in seconde : String) : double
onCreateOntioneMenu(inout menu : Menu) : boolean				L		
onOntionsItemSelected/inout item : MenuItem) : boolean						





#### Rôle des classes

#### - La classe TReception

Elle permet d'écouter les connexions bluetooth sur le terminal et d'envoyer les différents codes et les données en cas de réception. Il est donc nécessaire que cette classe dispose de son propre fil d'exécution elle hérite donc de la classe Thread<sup>4</sup>.

#### - La classe PeripheriqueBluetooth

Cette classe est aussi mise dans un thread pour avoir accès à ces fonctionnalités la connexion au bluetooth, la deconnexion et si besoin l'envoie de trame. Elle permet la création de module bluetooth sur notre terminal pour la connexion avec eux. Elle est reliée à la classe MainActivity grâce à un Handler<sup>5</sup>.

#### - La classe TrameTTE

Cette classe permet de gérer l'analyse de la trame pour identifier si c'est une trame TEC a la suite de cela elle décode la trame pour stocker les données de la TEC qui sont ensuite envoyées par le handler au MainActivity.

#### - La classe Trajet

La classe trajet permet de gérer les données d'un trajet lorsque nous appuyons sur le bouton trajet a partir de ce moment est calcule la vitesse moyenne la distance parcours et le taux de décharge de la batterie durant le trajet.

Une fonction est offerte avec la classe Localisation qui permet d'update on peut donc modifier les paramètres pour afficher la position de la trottinette.

```
Dans l'activité MainActivity
private void initialiserLocalisation()
{
    [...]
    if (fournisseur != null)
        {
        [...]
        // on configure la mise à jour automatique : au moins 1 mètres et 5
    secondes
        locationManager.requestLocationUpdates(fournisseur, 5000, 1,
    ecouteurGPS);
    }
}
```

<sup>&</sup>lt;sup>4</sup> Thread : Fil d'exécution en parallèle de celui du main

<sup>&</sup>lt;sup>5</sup> Handler : Utiliser pour communiquer entre un thread et le main, il met aussi à jour l'IHM.

Projet Trottinette Électrique Connectée



# €\_\_0

## - La classe MainActivity

Cette classe représente notre point d'entrée dans le programme. C'est ici que les objets sont instanciés. Elle a pour rôle d'afficher l'IHM. Une liste déroulante permet de sélectionner le module bluetooth auquel nous voulons nous connecter. A la selection d'un module dans cette liste déroulante un périphérique sera alors créé et lors de la connexion ou déconnexion les information techniques seront affichées dans une "console".

#### Dans l'Activité MainActivity

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, noms);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_it
em);
spinnerListePeripheriques.setAdapter(adapter);
spinnerListePeripheriques.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener()
{
    public void onItemSelected(AdapterView<?> arg0, View arg1, int
position, long id)
    {
        peripherique = peripheriques.get(position);
        setBoutons(position);
    }
[...]
});
```

Dans cette classe on retrouve aussi le système de handler précédemment introduit.

Les instanciation de nos ressources pour l'affichage de nos données :

#### Dans l'Activité MainActivity

```
boutonTrajet = (Button) findViewById(R.id.boutontrajet);
spinnerListePeripheriques = (Spinner)
findViewById(R.id.spinnerListeAppareils);
affichageDistanceParcourue = (TextView)
findViewById(R.id.affichageDistanceParcourue);
affichageChargeBatterie = (TextView)
findViewById(R.id.affichageChargeBatterie);
affichageVitesse = (TextView) findViewById(R.id.affichageVitesse);
```





## Étude détaillée

## Scénarios

#### - Le cas d'utilisation "Dialoguer avec la TEC"

< <actor>&gt;</actor>	$\bigcirc$
	dialoguer avec la TEC

La récupération des données de fonctionnement de la trottinette nécessite un dialogue via le Bluetooth.

#### - Diagramme de séquence







On remarque sur ce diagramme un échange d'un message depuis TReception qui est un Thread. Ce message contient donc :

- Le nom du module
- Son adresse MAC
- Un etat (connection, deconnection ou reception)
- Et la données

#### Dans la classe TReception

```
Message msg = Message.obtain();
Bundle b = new Bundle();
b.putString("nom", getNom());
b.putString("adresse", getAdresse());
b.putInt("etat", CODE_RECEPTION);
b.putString("donnees", data);
msg.setData(b);
handlerUI.sendMessage(msg);
```

Ce message est ensuite empaqueté dans un bundle pour le transfert au main activity via le handler.







#### - Diagramme de classes partiel





Session 2018



#### - Tests

#### Notre application au demarrage :



Projet Trottinette Électrique Connectée



Dans cette partie notre test a pour but d'observer une connexion vers un module bluetooth ici nous avons filtré les modules HC-05 :

Liste Trottinette : HC-05

#### Code du filtrage

Dans l'activite MainActivity
for (BluetoothDevice blueDevice : devices)
{
 if((blueDevice.getName().equals("HC-05")))
 {
 peripheriques.add(new PeripheriqueBluetooth(blueDevice,
 gestionnaire));
 noms.add(blueDevice.getName());
 }
}

Si notre filtre n'est pas actif on a alors une liste beaucoup plus large de tous les modules appareillés :

A Manager	TTPA-Table	
===> Localisation : Latitude ===> Localisation : Vitesse	HC-05	Ĩ
===> Adresse : 8 Rue Pont ` 84000 Avignon France	TTPA-Lanceur	
===> Périphérique sélectior	TEC1	ł
Liste Trottinette :	TTPA-Ecran	I

A partir de cette liste, nous sélectionnons le module bluetooth que nous voulons connecter.

Dans la console du technicien, les informations du module sont affichées :

===> Périphérique sélectionné HC-05 98:D3:31:F5:6E:7AJ déconnecté

On y retrouve le nom du module, son adresse mac et son état.

Nous allons ensuite pouvoir nous connecter grâce au bouton suivant :



Le module est alors en cours de connexion et on peut observer dans la console :

===> Connexion HC-05 [98:D3:31:F5:6E:7A]

Version 1.0

HACHETTE Alexandre (IR)







#### On obtient :





## Le cas d'utilisation "Visualiser les données de fonctionnement"



Visualiser les données de fonctionnement

L'utilisateur doit pouvoir visualiser les données de fonctionnement de la trottinette. Pour cela il est nécessaire qu'une connexion Bluetooth soit établie.

#### Diagramme de séquence

La Salle

Avignon

BTS SN



La réception de nouveau message est fixé à 1s et nous permet de recevoir une trame qui est ensuite vérifiée pour savoir si elle est du bon type avant qu'elle soit décodée pour récupérer les données de notre trottinette.

On remarque dans le diagramme qu'une trame est échangée entre MainActivity et TReception à l'aide du handler. Elle est ensuite traitée par la classe TrameTTE de telle sorte que la donnée soit affichable dans des TextView de notre IHM. La trame est un String et contient la vitesse, la charge de la batterie et la distance parcourue par la trottinette.





## - Diagramme de classes partiel

AppCompatActivity	Handler	Thread		
T /	6. C.		$\backslash$	
MainActivity		$\lambda$		
TAG : String	1		/	
bluetoothAdapter : BluetoothAdapter			`	\ \
noms : String				TReception
devices : BluetoothDevice		PeripheriqueBluetooth		handler II : Handler
trajetEncours : Boolean		nom : String		fini : boolean
ID_DISCOVERABLE_INTENT : int		adresse : String		nhTrames : long
REQUEST CODE ENABLE BLUETOOTH : int		gestionnaire : Handler		link : long
spinnerListePeripheriques : Spinner		peripherique : BluetoothDevice		envove : boolean
mListView : ListView		socket : BluetoothSocket		inactif : boolean
journal : String		fluxEntree : InputStream		TRecontion(input h : Handler)
mListLog : String		fluxEnvoie : OutputStream		rineception(inour in . nanuler)
affichageDistanceParcourue : TextView		CODE CONNEXION : int		run(). Volu
affichageChargeBatterie : TextView		CODE RECEPTION : int		attendre Prochainel, ecture (in millin : int) : wold
affichageVitesse : TextView		CODE DECONNEXION : int	•	attendre Prochaine Lecture (in minis . mi) . Void
affichageDistanceParcourueTrajet : TextView		CODE_INACTIF : int		viderBuffer() : void
affichageConsommationTrajet : TextView		CODE ACTIF : int		viderBuller() : void
affichageVitesseMoyenneTrajet : TextView	peripheriques	TAILLE BUFFER : int		anerel(). Volu
boutonCommunicationTTE : Button	< <list>&gt;</list>	PROCHAINE LECTURE : int		
boutonTrajet : Button		PERIODE INACTIVE : int		
boutonParametrage : Button		PeripheriqueBluetooth(inout peripherique : BluetoothDevice, inout gestionnaire : Handler)		
myOpenMapView : MapView		getDevice() : BluetoothDevice		BluetoothSocket
myScaleBarOverlay : ScaleBarOverlay		getNom() : String	◇	
mCompassOverlay : CompassOverlay		getAdresse(): String		
mRotationGestureOverlay : RotationGestureOverlay	peripherique	estConnecte() : boolean		
myLocationOverlay : DirectedLocationOverlay		setNom(in nom : String) : void		
locationManager : LocationManager		toString() : String		
fournisseur : String		connecter() : void		
geocoder : Geocoder		envoyer(in data : String) : void		
etatLocalisation : int		deconnecter(inout fermeture : boolean) : boolean		
mAzimuthAngleSpeed : float				
mSpeed : double				
ecouteurGPS : LocationListener				
bluetoothReceiver : BroadcastReceiver				
gestionnaire : Handler				
onCreate(inout savedInstanceState : Bundle) : void		TrameTTE		
onResume() : void		IdTEC : int		
onDestroy() : void		Vitesse : int		
initialiserActivite() : void		ChargeBatterie : int		
gererBoutons(): void		DistanceParcourue : int		
onClick(inout view : View) : void		DEBUT TRAME : String		
journaliser(in text : String) : void	~	TYPE PROTOCOLE : String		
initialiserCarte() : void	tra	Ime FIN TRAME : String		
initialiserLocalisation() : void		TrameTTE()		
arreterLocalisation() : void		TraiterTrame(in trame : String) : Boolean		
initialiserCommunication() : void		VerifierTrame(in trame : String) : Boolean		
activerBluetooth() : void		DecoderTrame(in trame : String) : boolean		
arreterBluetooth(): void		getVitesse() : int		
atticherPositionTTE(): void		getChargeBatterie() : int		
afficherDonnees I (E(inout ok : Boolean) : void		getDistanceParcourue() : int		
onActivityResult(in requestCode : int, in resultCode : int, inout data : Intent) : void		getIdTEC() : int		
getApplicationName(inout context : Context) : String				
getPosition(in adresse : String) : int				
setBoutons(in position : int) : void				
remplacerPerpherique(inout perpherique : PerpheriqueBluetooth) : void				
oncreateoptionsMenu(inout menu : Menu) : boolean				
onOptionsitemselected(inout item : MenuItem) : boolean	J.			





#### - Tests

L'objectif de ce test est de visualiser les données de fonctionnement de la trottinette Après la connexion au bluetooth, nous activons notre simulateur de trame :

Date	Heure	Vitesse (km/h)	Charge (%)	Distance (m)
18/05/24	09/35/46	0	00	8
18/05/24	09:35:67	14	59	10
18/05/24	09:15:48	54	59	24
18/05/24	09:35:49	13	.50	32
16/05/24	09:35:50	12	5.0	3.9
18/05/24	09:35:51	12	50	40
18/05/24	09.35:52	13	50	53
18/05/24	09:35:53	14	50	61
16/01/24	09.33.34	14	5.0	6.9
18/05/24	09:35:55	14	10	11
18/03/24	09.35:56	12	50	84
18/03/24	09.35:57	12	5.0	9.1
16/01/24	09.35.58	12	5.0	0.0
18/05/24	09:35:59	14	5.0	106
18/05/24	09:36:00	54	10	114
18/05/24	09:30:01	14	.99.)	122

Ce simulateur nous permet de générer des trames depuis un module bluetooth que nous avons relié à un PC. On effectuer les réglages suivants :

- La période en ms d'envois des trames
- Le port sur lequel notre module bluetooth est branché
- Le type de transmission que nous voulons

Modification des droits d'accès au périphérique Bluetooh :



On est alors prêt à envoyer nos trames vers notre application à l'aide du bouton Générer puis Simuler.

On obtient :

#### Trottinette Électrique Connectée

Session 2018





18/05/24	09:35:55	14	59	77
18/05/24	09(35:50	12	59	84
18/05/24	09:35:57	12	59	91
18/05/24	09:35:58	12	59	98
18/05/24	09:35:59	14.	59	106
18/05/24	09:36:00	14	59	114
18/05/24	09:36:01	14	59	122
Trame: STEC1,14,59,16* Trame: STEC1,14,59,24* Trame: STEC1,13,59,32* Trame: STEC1,12,59,38* Trame: STEC1,12,59,46* Trame: STEC1,13,59,53* Trame: STEC1,14,59,61* Trame: STEC1,14,59,61* Trame: STEC1,14,59,61* Trame: STEC1,14,59,61*				

Trottinette STX		I
Vitesse 12 km/h Batterie 59 % Distance 84 m	1	Parcours : - m Consommation : - % Vitesse moy. : - km/h
		And the second second
T H=	And	La Flore And Design And Particular And Particular A
- 7 -		
	- Plane de la Proprieta	Transa Receiver
A A A		
1.		Const
===> Données reques Hi ===> Trame valide HC-0 ===> Données reques Hi ===> Trame valide HC-0 ===> Trame valide HC-0	05 [98:03:31:F5:62:7A]: #TEC1,14:59,69* [98:03:31:F5:66:7A]: #TEC1,A >05 [98:03:31:F5:66:7A]: #TEC1,14:59,77* [98:03:31:F5:66:7A]: #TEC1,A >05 [98:03:31:F5:66:7A]: #TEC1,A [98:03:31:F5:66:7A]: #TEC1,A	
	A DECEMBER OF A	

Notre trame a donc été traitée de telle sorte à ce que nous puissions récupérer la vitesse la charge de la batterie et la distance parcourue.

Lorsqu'une trame est envoyée par notre simulateur, elle arrive à notre TReception qui va signifier à l'activité principale grâce à un code reception qu'une donnée est disponible :

Projet Trottinette Électrique Connectée

Version 1.0

HACHETTE Alexandre (IR)





```
Dans la classe TReception
```

```
private void recevoirTrame()
{
    [...]
   String datas = new String(rawdata);
   System.out.println("<Bluetooth> Reception " + datas);
    StringBuffer d = new StringBuffer(datas);
    Message message = Message.obtain();
   Bundle paquet = new Bundle();
    paquet.putString("nom", getNom());
   paquet.putString("adresse", getAdresse());
   paquet.putInt("etat", CODE_RECEPTION);
    paquet.putString("donnees", datas);
   message.setData(paquet);
   handlerUI.sendMessage(message);
    [...]
}
```

Dans l'activité principale MainActivity, nous faisons alors appel à une classe TrameTTE ou la trame va être vérifiée et si elle est bonne l'affichage sera activé

Dans l'activite MainActivity

```
case PeripheriqueBluetooth.CODE_RECEPTION:
    [...]
    if(trame.TraiterTrame(donnees))
    {
        afficherDonneesTTE(true);
        [...]
    }
  [...]
    break;
```

#### Dans la classe trameTTE

```
public Boolean TraiterTrame(String trame) throws InterruptedException
{
    if(VerifierTrame(trame))
    {
        if(DecoderTrame(trame))
            return true;
    }
    return false;
}
```





### Le cas d'utilisation "Visualiser la localisation de TEC sur une carte"

Dans cette partie l'objectif est de pouvoir visualiser la localisation de la trottinette et le trajet effectué depuis l'activation du bouton pour cela il est nécessaire qu'une connexion bluetooth soit établie.



#### - Diagramme de Séquence







#### Initialisation

Nous devons en premier temps, pour cette activité, initialiser nos variables ce qui est effectué dans initialiserCarte() puis on continue avec la fonction initialiserLocalisation() :

```
Dans l'activite MainActivity
```

```
private void initialiserLocalisation()
{
   [...]
  if (fournisseur != null)
   {
      [...]
       Location localisation =
locationManager.getLastKnownLocation(fournisseur);
       if(localisation != null)
       {
           // on notifie la localisation
           ecouteurGPS.onLocationChanged(localisation);
       }
       // on configure la mise à jour automatique : au moins 1 mètres et 5
secondes
       locationManager.requestLocationUpdates(fournisseur, 5000, 1,
ecouteurGPS);
  }
}
```

On peut donc voir que si nous recevons un "signal gps" nous allons "regarder" la dernière position connue et si il y en a une, on installe le gestionnaire pour l'écoute :

ecouteurGPS.onLocationChanged(localisation);

Cette ligne signifie que l'on a placé en quelque sorte une mis a jour automatique lors de changement qui appellera la fonction onLocationChanged(). Elle permet de changer la visualisation sur la carte.

Demander la mise à jour de la carte ce qui est effectué par cette ligne :

```
locationManager.requestLocationUpdates(fournisseur, 5000, 1,
ecouteurGPS);
```

- **5000** represente donc le nombre de millisecondes.

- 1 représente le nombre de mètres.

Ces paramètres règlent la précision du changement de position.





#### AfficherPositionTTE()

On a ensuite recours à afficherPositionTTE() qui va permettre d'afficher la position de notre TTE en signifiant à notre ecouteurGPS que la localisation de la trottinette a changé.

```
Dans l'Activité MainActivity
private void afficherPositionTTE()
{
   [...]
  if (locationManager != null)
   {
       if (fournisseur != null)
       {
           Location localisation =
locationManager.getLastKnownLocation(fournisseur);
           if (localisation != null)
           {
               // on notifie la localisation
               ecouteurGPS.onLocationChanged(localisation);
           }
       }
  }
}
```

Notre ecouteurGPS aura alors recours dans sa fonction onLocationChanged à l'affichage sur la carte de la position de la trottinette.

```
Dans l'activité MainActivity
LocationListener ecouteurGPS = new LocationListener() {
    @Override
    public void onLocationChanged(Location localisation)
    {
       [...]
       GeoPoint nouvelleLocalisation = new GeoPoint(localisation);
       [...]
       myOpenMapView.getController().animateTo(nouvelleLocalisation);
       myOpenMapView.setMapOrientation(-mAzimuthAngleSpeed);
       [...]
    }
    [...]
};
```





Ensuite intervient la partie du trajet, nous allons alors avoir besoin d'afficher ces données pour cela une classe Trajet a été implémenté et 3 fonctions pour accéder aux données sont nécessaire.

getVitesseMoyenneTTE()

Dans le cas ou nous voulons récupérer la vitesse moyenne de notre trottinette, on va la calculer puis la retourner.

Dans la classe Trajet
public String getVitesseMoyenneTTE()
{
 vitesseMoyenneTTE = calculVitesseMoyenne();
 return vitesseMoyenneTTE;
}

calculVitesseMoyenne()

Dans cette partie on introduit le besoin d'avoir le **temps depuis le début du trajet** ce qui va nous permettre de **calculer la vitesse moyenne** 

Dans la classe Trajet

```
private String calculVitesseMoyenne()
{
    [...]
    if(heureDebut.length() > 0)
    {
        double dureeTrajet = decoderTemp(heureActuelle) -
    decoderTemp(heureDebut);
        tempsTrajet = String.format("%.2f",dureeTrajet);
        double vitesseMoyenne = (distanceParcourueTrajet/1000) /
dureeTrajet;
        vitesseMoyenneTTE = String.format("%.2f",vitesseMoyenne);
        return (vitesseMoyenneTTE);
    }
    return "0";
}
```





#### decoderTemp()

Cette dernière partie a pour objectif de séparer les heures, minutes et secondes de notre **temps à convertir** pour pouvoir effectuer sa conversion en double pour le calcul de la vitesse moyenne.

```
Dans la classe Trajet
```

```
private double decoderTemp(String tempsAConvertir)
{
   [...]
   for (int i=0; i < tempsAConvertir.length() ;i++)</pre>
   {
       [...]
       else
       {
           if (position == 0)
           {
               heure = heure + tempsAConvertir.charAt(i);
           } else if (position == 1)
           {
               minute = minute + tempsAConvertir.charAt(i);
           } else
           {
               seconde = seconde + tempsAConvertir.charAt(i);
           }
       }
   }
   return (conversionTemps(heure,minute,seconde));
```

#### conversionTemps()

Cette dernière fonction permet de convertir notre temps en double pour le calcul.

```
Dans la classe Trajet
```

```
private double conversionTemps(String Heure, String minute, String
seconde)
{
    double temps = 0;
    Double heureConversion;
    Double minuteConversion;
    Double secondeConversion;
    heureConversion = Double.parseDouble(Heure);
    minuteConversion = Double.parseDouble(minute) / 60;
    secondeConversion = Double.parseDouble(seconde) / 3600;
    [...]
    temps = heureConversion + minuteConversion + secondeConversion;
    return temps;
}
```





#### - getConsommationTrajet()

Ici on cherche à récupérer la charge depuis le début de notre trajet on soustrait donc la charge initiale de la trottinette lors de l'activation du mode trajet et on y soustrait la charge actuelle de la batterie.

Dans la classe Trajet

```
public int getConsommationTrajet(int chargeTTE)
{
    return chargeInitialeTTE - chargeTTE;
}
```

- getDistanceParcourue()

On cherche à récupérer la distance parcourue depuis le début de notre trajet on soustrait donc la distance parcourue initialement par la distance parcourue actuellement.

Dans la classe Trajet

```
public int getDistanceParcourue(int distanceParcourue)
{
    distanceParcourueTrajet = distanceParcourue -
    distanceParcourueInitiale;
    return (distanceParcourue - distanceParcourueInitiale);
}
```





#### - Diagramme de classes partiel







#### - Tests

L'objectif de ce test est de visualiser la localisation de la trottinette ainsi que pouvoir activer un mode trajet pour visualiser le parcours a effectué.

Pour le mode trajet, il est impératif que notre module bluetooth soit connecté à la trottinette.



Projet Trottinette Électrique Connectée





Nous pouvons maintenant activer le mode trajet à l'aide du bouton :



Le trajet passera alors en mode actif :





Le mode trajet nous permet d'obtenir les données de notre trajet comme vu précédemment depuis le code.

Vitesse moy : 27,00 km/h Consommation : 0 % Parcours : 23 m

La Salle

Avignon des Écoles Chrétiennes BTS SN

Une fonctionnalité a été ajouté qui est l'affichage de notre trajet en cours à l'aide d'une ligne rouge selon les déplacements depuis la dernière position.



Projet Trottinette Électrique Connectée



Session 2018



Une fois le mode trajet activé, si nous déconnectons le module bluetooth, le trajet s'arrête instantanément.



Projet Trottinette Électrique Connectée





## Tests de validation

Test de validation				
Nom du test	Oui	Non		
Le protocole de communication avec la TTE est spécifié et mis en oeuvre	Х			
La réception des données de fonctionnement de la TTE est effective	Х			
La visualisation des données de fonctionnement de la TTE et la durée d'utilisation est fonctionnelle	Х			
L'autonomie pour un parcours est calculée et affichée	Х			
La carte avec la géolocalisation de la TTE est affiché et actualisée périodiquement	Х			

## Conclusion

- Nous pouvons faire des améliorations graphique comme un compteur de vitesse.
- Créer une base de données pour stocker nos trajet et les données de ceci.
- Saisir une destination et crée un trajet pour y arriver.





## Glossaire

TTE : Trottinette tout Terrain Électrique TEC : Trottinette Électrique Connectée (la TTE avec le système développé) Diagramme de Gantt : c'est un outil permettant de visualiser dans le temps les diverses tâches composant un projet. Thread : Activité en multitâche Handler : Gestionnaire pour passer sur une autre activité