**La Station Météo**

Rapport de projet



# Remerciements

Arrivés au terme du projet Station Météo, nous tenons à remercier les différents professeurs qui ont contribués à la réalisation de ce projet.

Parmi eux, M. Martincic et tout le corps enseignant sans qui nous n’aurions pas eu les compétences nécessaires pour mener à bien notre projet.

# Résumé

Ce rapport consiste à expliquer le contenu du travail effectué, toutes les connaissances acquises, ainsi que le principe de fonctionnement du projet.

Nous présenterons donc les éléments qui constituent notre système de Station Météo et leur programmation, nous présenterons également les objectifs de réaliser un tel projet.

Travailler en groupe pour un tel projet permet de partager nos connaissances, ce qui nous permet d’avancer plus rapidement pour la réalisation du projet. Un programme principal en Arduino sera le cerveau du projet.

Sommaire

[Remerciements 2](#_Toc11934678)

[Résumé 3](#_Toc11934679)

[Introduction 5](#_Toc11934680)

[I. Présentation du projet 6](#_Toc11934681)

[A. Cahier des charges 6](#_Toc11934682)

[B. Schéma synoptique 6](#_Toc11934683)

[II. Station Météo, le Hardware 7](#_Toc11934684)

[A. Présentation des composants 7](#_Toc11934685)

[III. Station Météo, Software 10](#_Toc11934686)

[A. Le protocole I2C 10](#_Toc11934687)

[B. L’interface graphique 10](#_Toc11934688)

[1. Le fonctionnement de l’application 10](#_Toc11934689)

[2. Bluetooth 12](#_Toc11934690)

[3. Graphique 14](#_Toc11934691)

[C. La programmation du système 14](#_Toc11934692)

[1. Le protothreading : 14](#_Toc11934693)

[Conclusion 17](#_Toc11934694)

# Introduction

Ce projet est la finalisation de notre année de licence et nous permet donc de mettre en œuvre toutes les connaissances apprises tout au long de note cursus à l’IUT.

Le projet consiste à la programmation d’une carte Arduino sur laquelle est fixée une carte d’extension qui nous permet d’avoir différents capteurs. Le but principal de notre projet est de récupérer des données, puis les stockées dans des cases mémoires, pour par la suite les envoyés par Bluetooth a une application mobile.

L’objectif global est donc de concevoir un système de mesures autonomes pouvant enregistrer des données sur une longue période.

# Présentation du projet

## Cahier des charges

Pour ce projet, nous avons eu besoin de certains éléments :

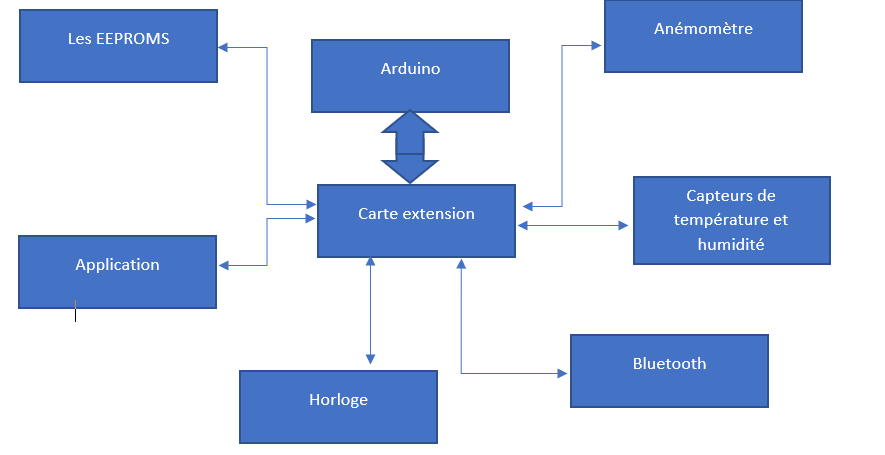
* Carte Arduino
* Carte d’extension
* Capteur de température LM35
* Capteur de température et humidité AM2315
* Anémomètre
* Girouette
* Horloge temps réelle DS1307
* Module Bluetooth HC-05
* Pile AA

Nous avons également eu besoins d’utiliser certains logiciels :

* Arduino
* Altium Designer
* Android Studio

L’objectif de ce projet est de développer un système de mesures autonomes pouvant stocker les différentes données sur une certaine période.

## Schéma synoptique

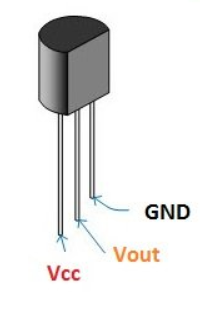
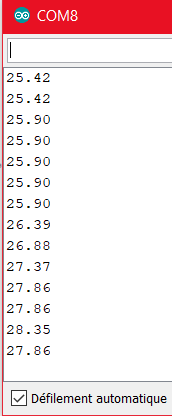


# Station Météo, le Hardware

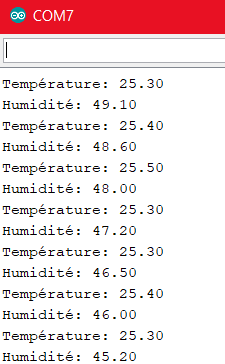
## Présentation des composants

Nous allons maintenant présenter les composants que nous avons dû utiliser tout au long ce projet. Tous les composants cités précédemment ont tous été utilisés.

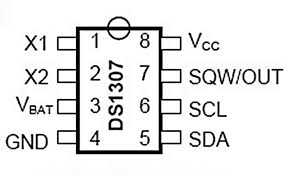
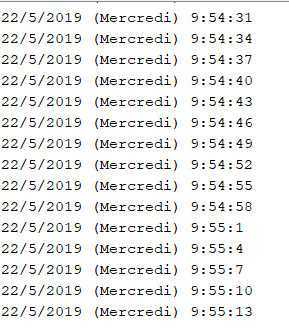
**Le capteur de Température LM35 :**

Le LM35 est un capteur de température qui peut récupérer une température variant de de -55°C jusqu’à +150°C avec une précision à près d’un quart de degrés. Il ne nécessite aucun calibrage et est donc près à une utilisation sans problèmes, il correspond donc parfaitement au cahier des charges qui nous demande de récupérer la température au niveau de la carte.

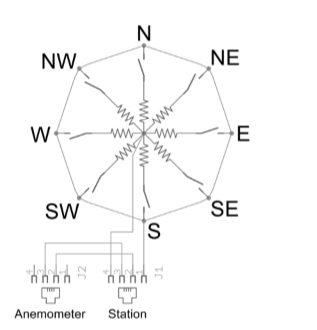
**Le capteur de Température et d’Humidité AM2315 :**

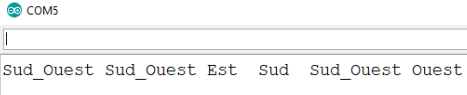
Le AM2315 est un capteur de température et d’humidité équipé d’une interface I2C contenu dans un boitier qui contient notamment des senseurs de température DS18B20 ainsi qu’un capteur d’humidité à effet capacitif. Le fait qu’il embarque une interface I2c signifie qu’il s’agit d’un capteur qui est près à l’emploi. Il permet de mesurer la température en variant de -20°C jusqu’à +80°C avec une précision de 0.1°C, en ce qui concerne l’humidité, celui nous donne une précision à environ 2% de la vraie valeur.

**L’horloge DS1307 :**

Pour mener à bien ce projet, nous avons dû utiliser une horloge temps réel afin de stocker les valeurs récupérées tout en les datant. C’est donc grâce à cette horloge que ce travail a pu être réalisé. Il s’agit d’une horloge fonctionnant en I2C.

**L’anémomètre et girouette :**

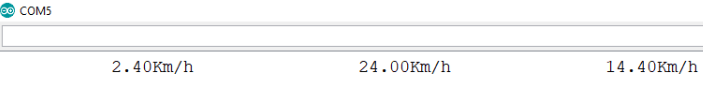
Nous nous sommes occupés des deux informations, le vent et la direction de celui-ci. Le vent avec un capteur anémomètre et la direction avec une girouette. Ces deux capteurs fonctionnent ensemble, l’anémomètre est connecté à la girouette par un câble RJ12, la girouette est ensuite connectée elle-même à notre carte.

Ci-contre, le fonctionnement de la girouette. On injecte du 5V dans une résistance de 10K sur la broche 4, celle-ci connectée à chaque résistance, nous renvoie une valeur prédéfinie.

Ensuite pour le second capteur, anémomètre, il faut ATTENTION EGALEMENT (car l’information n’est présente sur aucune documentation) mettre le 5 V et une résistance de 10kOhms sur la broche 3, sans cela, vous aurez seulement récupéré des valeurs incompréhensibles.

Contrairement à la girouette, il y a une seule résistance, et celle-ci, nous renvoie un 0 ou bien 1023,

Après différentes façons d’aborder, utiliser la fonction interrupts et non les timers de l’Arduino ont été le choix préférer. Attention à bien utiliser « interrupts () ; » et non « cei() ;



**Le module Bluetooth HC-05 :**

Pour pouvoir communiquer avec l’appareil android, on a besoin d’utiliser un émetteur bluetooth qui effectue l’échange de données avec la carte arduino.

****

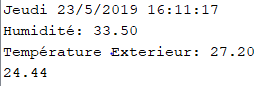
Ici on utilise le module Bluetooth HC05 de gotronics. Ce module est alimenté en 5V et utilise les pins RX TX 11 et 10 de la carte arduino.



**Les EEPROM I2C :**

Nous allons utiliser des mémoires de type EEPROM afin de sauvegarder les données reçues de la station météo.

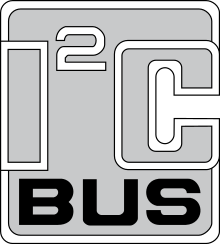
Après le rassemblement des programmes on obtient un affichage :



# Station Météo, Software

## Le protocole I2C

Le protocole I2C a été conçu par l’entreprise PHILIPS en 1982. Il s’agit d’un protocole de transfert de données qui permet de relier facilement un microprocesseur à plusieurs capteurs. Aujourd’hui de nombreux composants utilisent ce protocole, qui est très facile d’utilisation.



Son principe de fonctionnement est simple et n’utilise que trois fils :

* SDA : le signal de données
* SCL : signal d’horloge
* Masse : la référence électrique

Seulement trois fils sont utilisés, car il s’agit d’un signal bidirectionnel half-duplex, c’est-à-dire que les différentes informations transitent sur une seul et même fil, mais pas en même temps. La transmission se fait lorsque le maitre initialise le dialogue qui va générer un signal d’horloge. Par la suite, en fonction du signal reçu, l’esclave va être en mesure d’envoyer ou de recevoir des données.

## L’interface graphique

Nous avons décidé de faire une application pour pouvoir observer et examiner les valeurs des capteurs de température, d’humidité, la direction et la vitesse du vent. Nous avons réalisé une application sur Android Studio.

### Le fonctionnement de l’application

Android Studio utilise des fichiers ressources tel que le XML pour générer une interface graphique. Ce fichier est appelé par le fichier principal grâce à la commande suivante

La figure ci-dessous vous montre comment générer une interface graphique en XML. Les composants ou les widgets sont définis par des balises et les attributs représentent les propriétés.

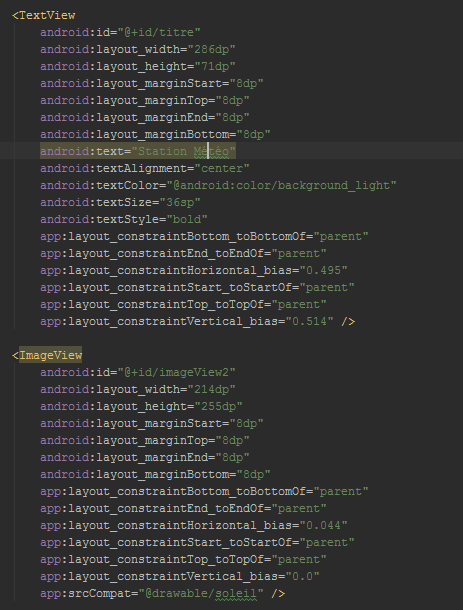
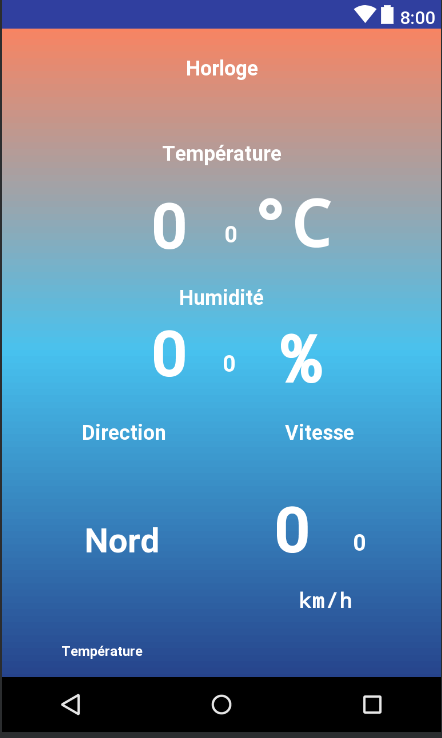


Figure 2 Création de l'interface via les balises

Pour chaque interface, on a une classe qui gère sa propre interface. Pour notre projet, nous avons 5 classes : - Menu.java qui anime juste l’interface pour ensuite passer à une autre activité. - Bluetooth.java qui donne une liste d’appareil et lorsqu’on choisit le bon appareil (pour nous c’est le module HC – 05), il tente de se connecter avec le module pour ensuite afficher une autre interface.

* Station.java qui nous montre toutes les données qu’on reçoit via l’Arduino
* Température.java qui nous donne un graphe de toutes les données de température extérieure.
* Humidité.java qui nous donne un graphe de toutes les données acquises sur L’humidité.

Appuyer sur retour

Cliquer sur les valeurs de °C et %

Connexion

Attente

Appuyer sur retour

Figure 3 Déroulement de l'application

On va maintenant parler du Bluetooth qui est le centre de cette application.

### Bluetooth

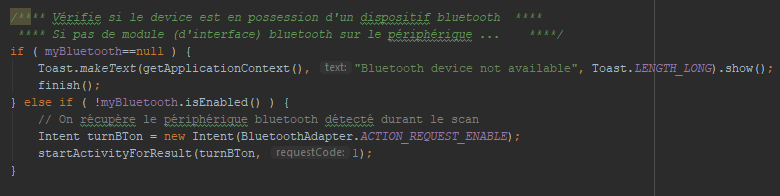
Pour pouvoir utiliser le Bluetooth sur l’application, nous devons d’abord fournir les autorisations dans le fichier AndroidManifest.xml.



Dans la classe Bluetooth.java, on initialise d’abord les variables du Bluetooth



Pour éviter des erreurs peu communes et faire crasher l’application, il est préférable de vérifier si le périphérique est doté d’un adaptateur Bluetooth et s’il est activé.



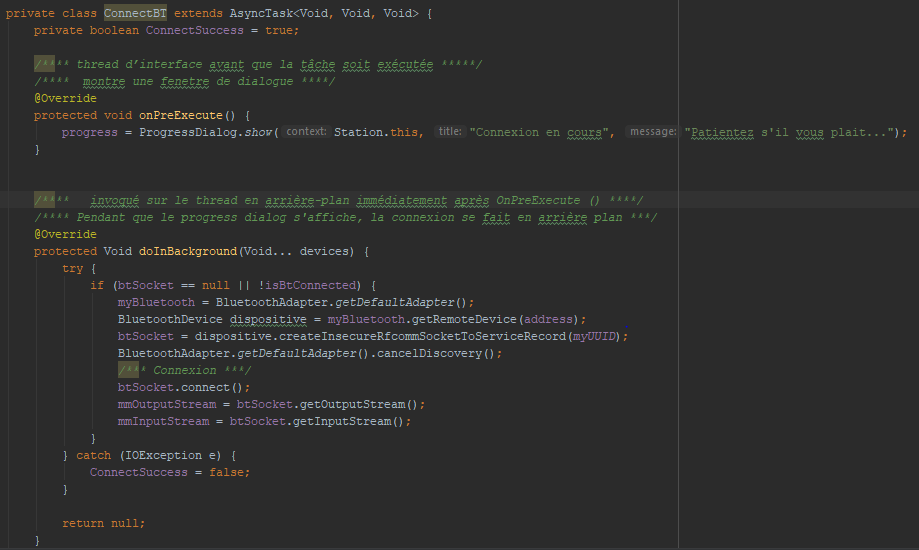
Selon les documents Android, un Intent est un objet de messagerie qu’on peut utiliser pour demander une action à un autre composant de l'application. Les Intent facilitent la communication entre les composants de plusieurs manières.

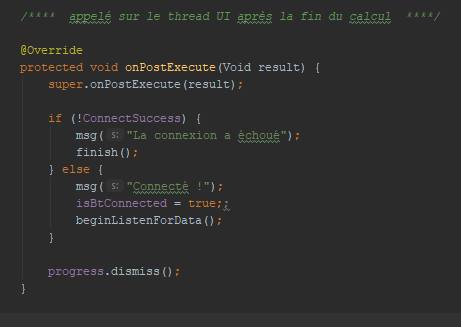
Maintenant on veut récupérer l’adresse de périphérique Bluetooth obtenue dans la classe paireDeviceList () dans la classe Bluetooth.java

Dans la classe Station.java, on reçoit l’adresse du périphérique.



Pour la connexion du Bluetooth, nous avons utilisé AsyncTask. Cela permet une utilisation correcte et facile du ThreadUI (responsable de l'affichage de l'Activité). Idéalement, les AsyncTasks sont utilisées pour des utilisations courtes.





La méthode beginListenForData est là pour récupérer les données envoyées par Bluetooth. Nous avons d’ailleurs eu un peu de mal à recevoir, nous avons donc changé cette fonction plusieurs fois lors des assemblements de programme au sein du groupe On va maintenant parler des graphiques qu’on va dessiner pour observer variations de températures et d’humidités au fonction du temps.

### Graphique

Nous avons plusieurs librairies pour dessiner des graphiques sur Android Studio. J’ai opté pour une librairie MPChart Android. Il en avait d’autres tels que SCI Chart qui est plus performant mais il faut acheter la licence. Pour utiliser MPChart Android, nous devons rajouter dans le gradle d’Android studio.



Nous pouvons dessiner des graphiques en utilisant les bibliothèques spécifiques. Cependant, nous avons eu quelques problèmes sur comment dessiner les graphiques. On aurait aimé utiliser des graphiques à temps réel pour observer directement les variations. Mais nous avons eu des problèmes, l’application crachait, donc nous avons opté pour l’actualisation du graphique via un bouton Actualiser. Or les valeurs qu’on envoie à l’activité qui gère les graphiques ne changent pas, alors qu’on reçoit bien les changements avec la fonction qui permet de recevoir les valeurs.

## La programmation du système

### Le protothreading :

Il est possible de simuler une exécution de tâches quasi-parallèles avec les fonctions du programme Arduino de la station météo.

Sur Arduino, la bibliothèque Protothreading permet de créer différents threads : des boucles qui s'exécutent à la suite, indéfiniment.

On peut mettre en pause chaque thread avec une condition telle qu’un timeout ou la bonne réception d’un message.

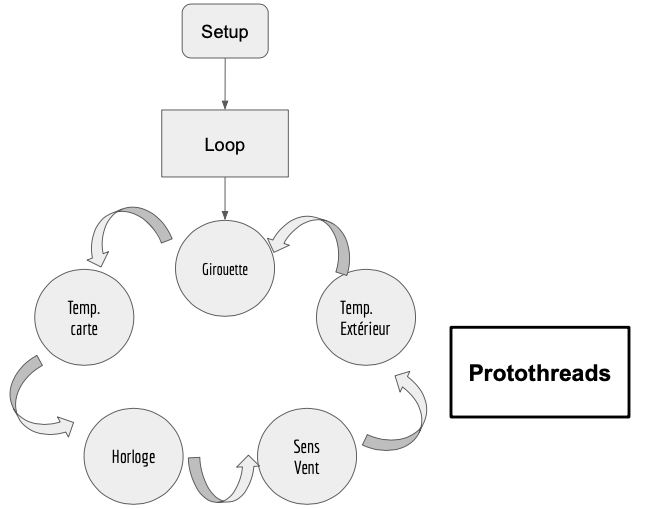
Lorsqu’une boucle n’a plus rien à faire, le programme cherche une autre boucle moins lourde à exécuter, ou se met en veille s’il n’y a rien à exécuter à cet instant.

On peut ainsi effectuer les différentes mesures et écritures de données ainsi que leur envoie à des intervalles différents.

**Exemple d’utilisation du protothreading :**

|  |
| --- |
| static struct pt threadHorloge,threadTempCard,threadTempExt,threadGirouette,  threadVent; //déclaration des threads  void setup() {  PT\_INIT(&threadHorloge); // Initialisation des threads  PT\_INIT(&threadTempCard);  PT\_INIT(&threadTempExt);  PT\_INIT(&threadGirouette);  PT\_INIT(&threadVent); }  void loop() {  //On appel les threads avec un temps d’execution protothread1(&threadHorloge,10000); protothread2(&threadTempCard,11000); protothread3(&threadTempExt,12000); protothread4(&threadGirouette,13000); protothread5(&threadVent,14000);  }  static int protothread1(struct pt \*pt,int interval) {  static unsigned long timestamp=0;   PT\_BEGIN(pt); // Début du thread   while(1){  //On attend jusqu’à ce que le temps soit dépassé, on exécute ensuite la boucle  PT\_WAIT\_UNTIL(pt,millis()-timestamp>interval);  timestamp=millis();   //////////////////////////////////////////////////////  // Ici se trouve le programme à exécuter en continu // /////////////////////////////////////////////////////   }  PT\_END(pt); } |

Dans notre projet, on utilise un thread pour chaque fonction de la station météo.



# Conclusion

Nous avons réussi à arriver au terme de ce projet, en revanche, nous avons rencontré quelques difficultés notamment sur la compréhension des capteurs anémomètre et girouette. Désormais, le projet associé à une application Android sur laquelle est retranscrit les différentes valeurs récupérées par la carte, ce qui nous permet d’avoir une visibilité sur les variations des valeurs.

Afin d’évoluer ce projet il serait possible de terminer l’association des graphiques aux valeurs récupérées afin d’avoir un suivi des différentes variations.