GESTION CONVECTEUR CROUS

Table des matières

Introduction

PARTI 1 : Présentation commune du projet

3. Présentation du sujet

4. Le but du projet

5. Le principe de réalisation

5. *Synoptique simplifié du système*

*5. Synoptique simplifié du boîtier de régulation autonome*

6. Analyse Fonctionnelle du système

*6. Diagramme de cas d’utilisation simplifié*

*6. Diagramme d’exigence*

*6. Diagramme de classe*

*6. Diagramme de séquence*

7. Organisation du projet

*7. GANTT Prévisionnel*

*7. GANTT Réel*

*7. Répartition des tâches*

8.Organisation de l’équipe

*8. Compte rendu d’activité (CRA )*

*8. Cahier de bord*

*8. Github et Versionning*

*8. Démarrage projet et classe de simulation*

*8. Logiciel d’analyse et de développement*

*8. Maquettage et Prototypage*

9. Choix technique et Étude physique

*9. Choix de la carte contrôleur pour le boitier connecté*

*9. Choix des capteurs et module*

10. Recette

11. Tests d’intégration du prototype

12. Avancement et Conclusion

PARTI 2 : Partie individuel de Bouet Valentin (20-30 pages)

13. Analyse 1

*1 Cas d’utilisations*

*2 Diagramme de séquence*

*3 Diagramme de classe*

*14. Suivie Du projet*

*1 Compte Rendu D’activité*

*2 Mise à niveau des heures déjà allouer*

*15. Conception du boitier*

*1 Alimentation des composants*

*16.Materiel*

*1 Carte Programmable*

*2 Systèmes de communication*

*3 Capteur*

*4 Actionneur*

*17. Code*

*1 Matériel de développement et langage*

*2 Présentation des différentes classes*

PARTI 3 : Partie individuel de Wantelez Florian (20-30 pages)

PARTI 4 : Partie individuel de Malter Alexandre (20-30 pages)

PARTI 5 : Partie individuel de Garcia Florian (20-30 pages)

26. Analyse 1

*1 Cas d’utilisations*

*2 Diagramme de séquence*

*3 Diagramme de classe*

*30. Suivie Du projet*

*1 Compte Rendu D’activité*

*2 Mise à niveau des heures déjà allouer*

*32. Matériel*

PARTI X : ANNEXE

PARTIE 1 : Présentation commune du projet

*Présentation du sujet*

Le but de notre projet est de permettre au CROUS de pouvoir gérer les radiateurs électriques présents dans les appartements à distance.

Les deux grandes exigences de notre client sont de rendre autonome les radiateurs et de centraliser leurs gestions. Actuellement, de nombreux logements étudiants possèdent des radiateurs à convection électrique. Pour répondre aux besoins exprimés par le client, nous allons créer un boîtier connecté qui pourra se greffer aux radiateurs et gérera leurs utilisations en leurs imposant des consignes de chauffes.



*Le but du projet*

Le but de notre projet est donc de répondre aux deux besoins principaux de notre client qui sont :

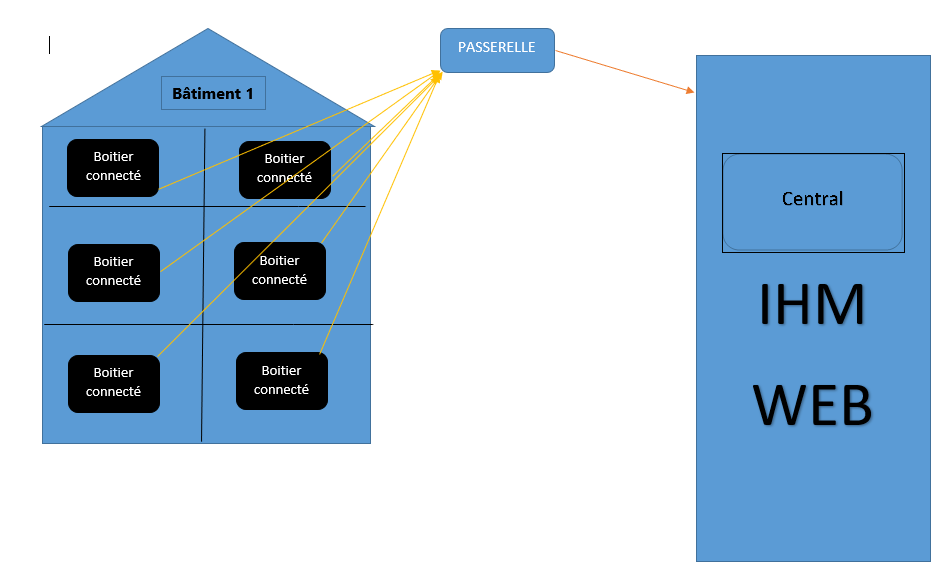
* **1) Rendre autonome les radiateurs**

* **2) Centraliser la gestion des radiateurs**

L’objectif final du projet est de réduire l’importante part de la consommation électrique que représente le chauffage de l’ensemble de ces logements étudiants en permettant au Crous de réguler leur utilisation.

*Le Principe de réalisation du projet*

**Synoptique simplifiée du système.**

****

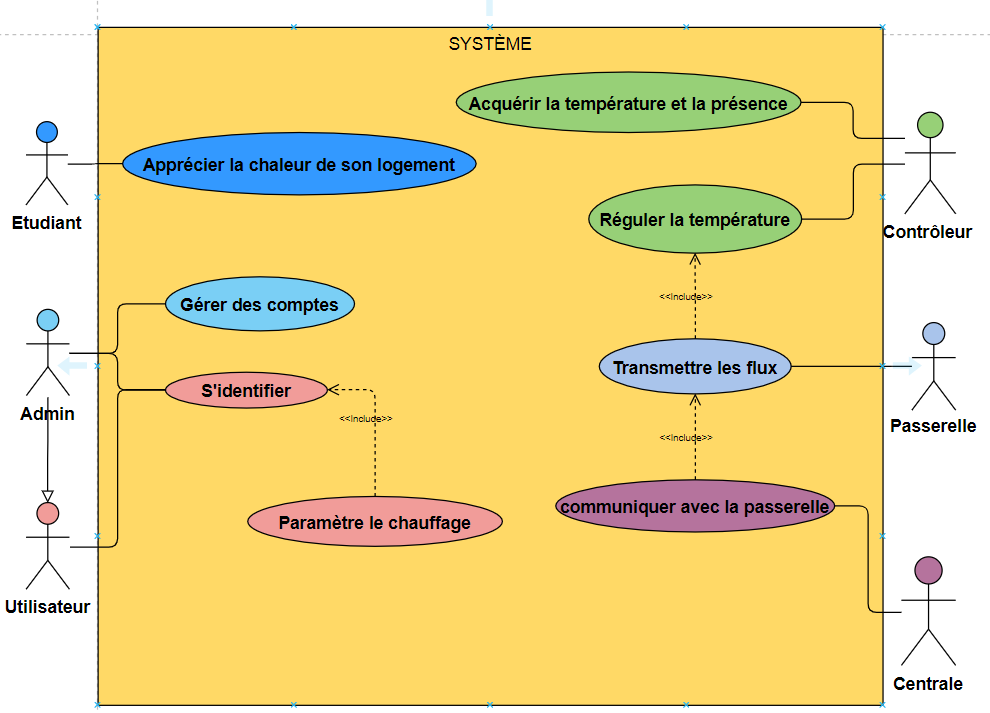
* Chaque radiateur se verra équipé d’un boîtier qui pourra contrôler le radiateur.
* Les boîtiers pourront communiquer avec la centrale présente aux bâtiments du CROUS via une passerelle de communication qui sera propre à chaque bâtiment de logement étudiant.
* Les opérateurs du Crous pourront voir et gérer chaque radiateur équipé d’un boîtier via un site web.

**Synoptique simplifié du boîtier de régulation autonome**



*Analyse fonctionnelle du système*

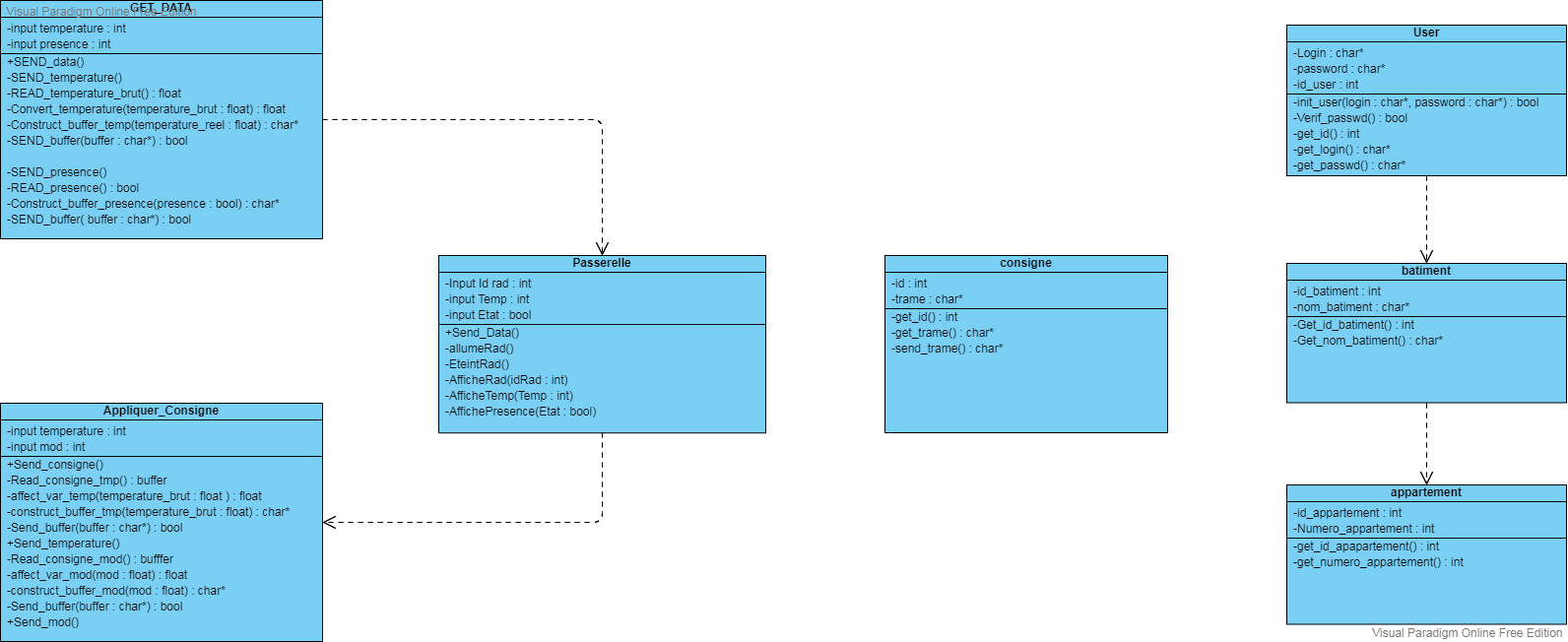
*Diagramme de cas d’utilisation simplifié*

**

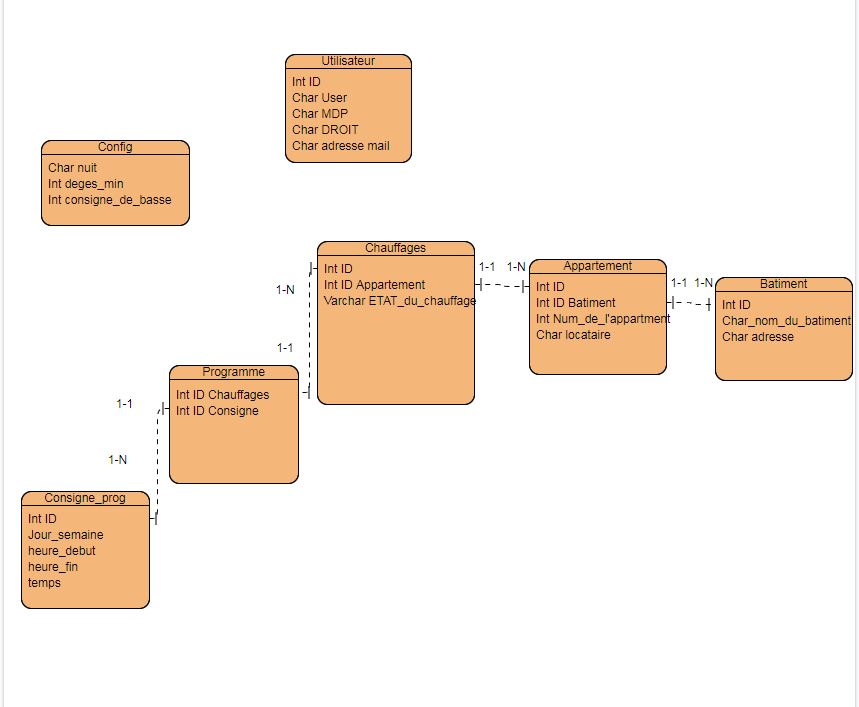
*Diagramme d’exigence*



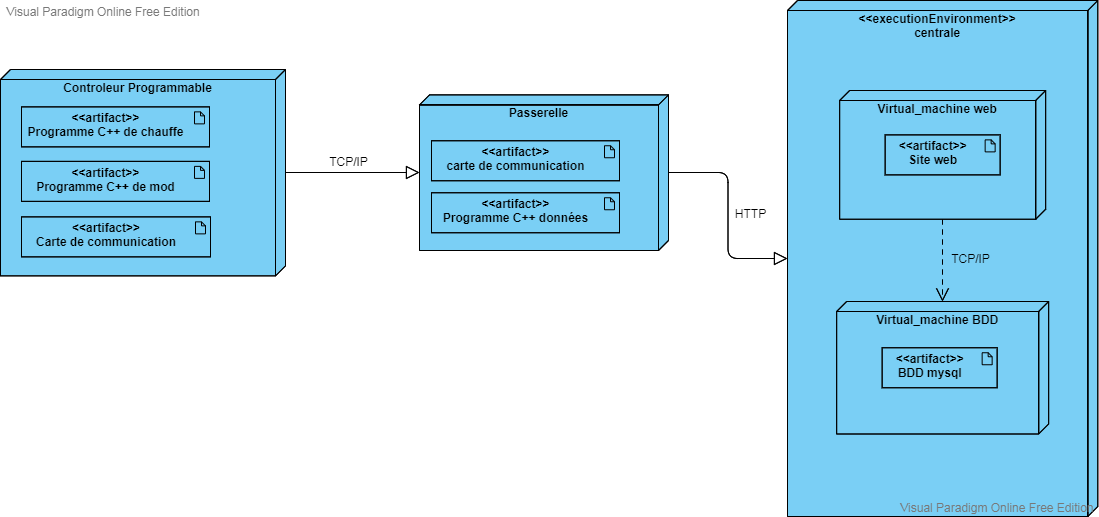
*Diagramme de classe*



*MCD*

**

*Diagramme de déploiement*

**

*Organisation du projet*

**Partage des ressources et méthodes du projet**

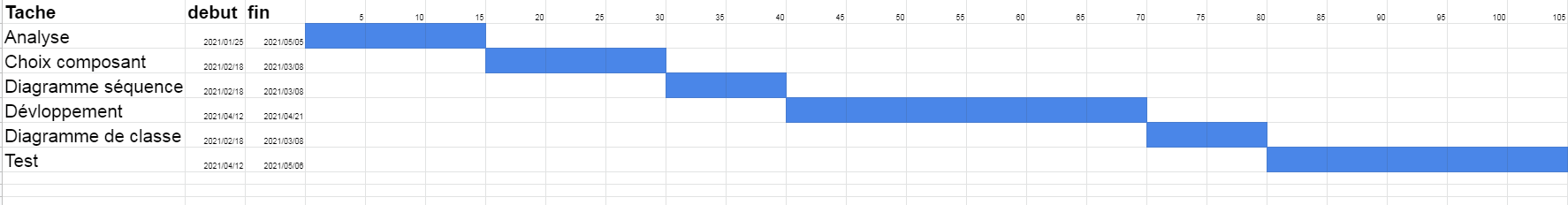
Une Virtual Machine (ip : 192.168.65.113) a été mise en place pour partager les ressources, diagrammes SYSML, code C++ et web lié au projet.

Deux Virtual machines permettent d’héberger la Base de Données sous mariaDB (ip : 192.168.64.155 et le site web (ip : 192.168.65.4).

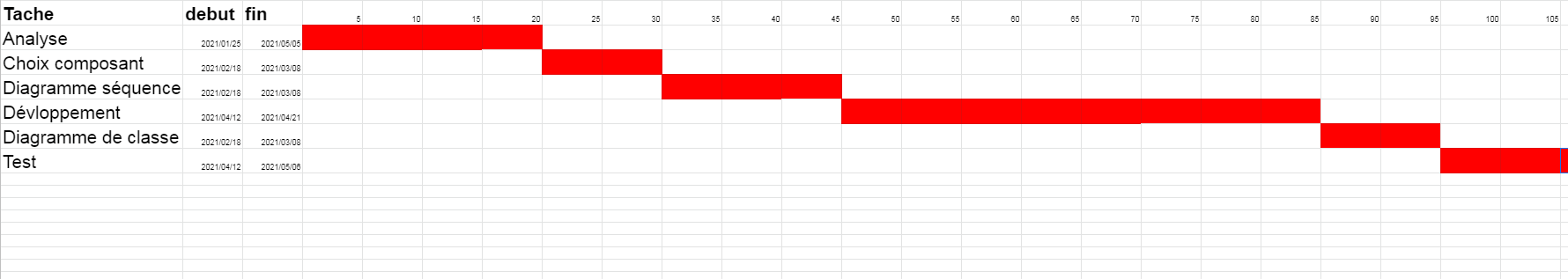
Un dossier Google drive mets en commun tous les fichiers sources et SYSML du projet afin de pouvoir les récupérer à notre domicile.

Enfin, un répertoire GITHUB est mis en place pour partager les codes web et c++ et pour nous permettre de les récupérer sur n’importe quel ordinateur.



**Gantt Prévisionnel**

**Gantt Réel**



**Répartition des tâches.**

Voici la répartition des tâches. Cette dernière nous est imposée par le sujet de BTS SN.

Etudiant 1 : Création de l’interface de gestion de pilotage centralisé en PHP/MYSQL

- Interface CRUD de Gestion des utilisateurs

- Interface CRUD de Gestion des radiateurs par Passerelles

- Interface CRUD de Gestion des Maisons (Passerelles)

- Création du MCD

- Page de Connexion

- Page d’affichage de toutes les informations des radiateurs

- Page de Paramétrage d’un Radiateurs

- Page de Paramétrage d’une Passerelle

Etudiant 2 : Création d’un prototype de boitier connecté autonome avec les composants. Régulation de température.

- Choix de la carte programmable

- Choix des capteurs de présence, de température, d’alimentation

- Tests unitaires de chaque Capteurs

- Algorithme de Gestion Autonome Reprenant les exigences de chauffes.

Etudiant 3 : Communication entre le boitier connecté et la passerelle (Sans fil) Communication entre la passerelle et l’interface de gestion (Via TCP/IP)

- Choix du composant sans fil de communication entre Boitier connecté et la passerelle

- Test de communications Bidirectionnel entre la passerelle et le boitier

- Mise en place d’un protocole pour identifier les différents radiateurs

- Création de l’application Passerelle qui récupère les informations des radiateurs pour les envoyer à l’application web de Centralisation.

- Test d’installation d’un Boitier et d’une passerelle en respectant les exigences.

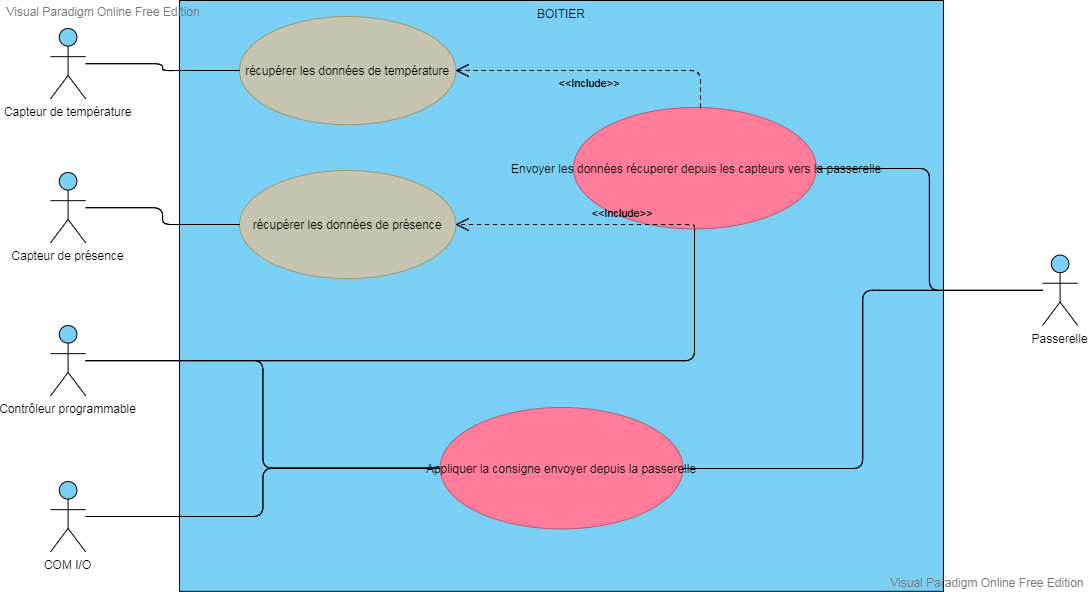
Etudiant 4 : Mise en place des échanges de commande entre la central et le radiateur. Mise en place du protocole de communication.

- Mise en place des échanges de messages entre Boitier et Central.

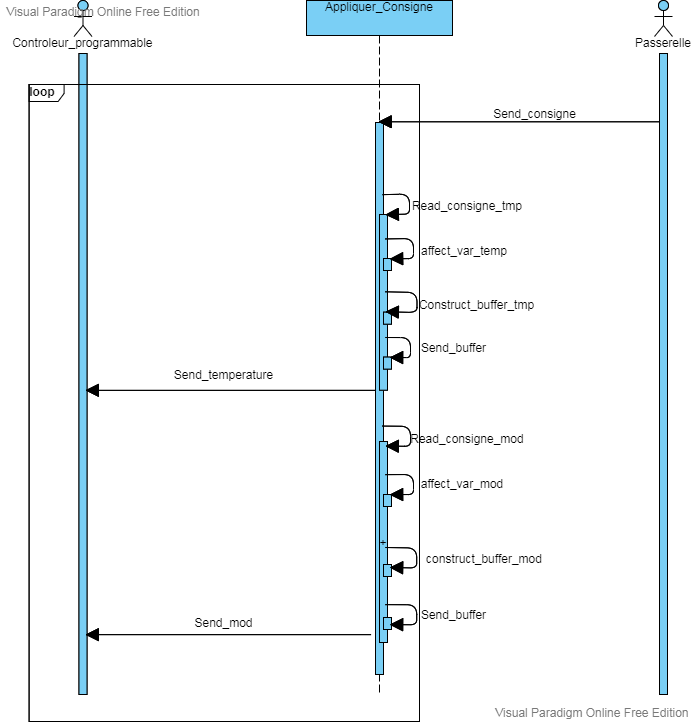
PARTIE 2 : Partie individuel de Valentin Bouet

*Analyse*

**Cas d’utilisations**

****

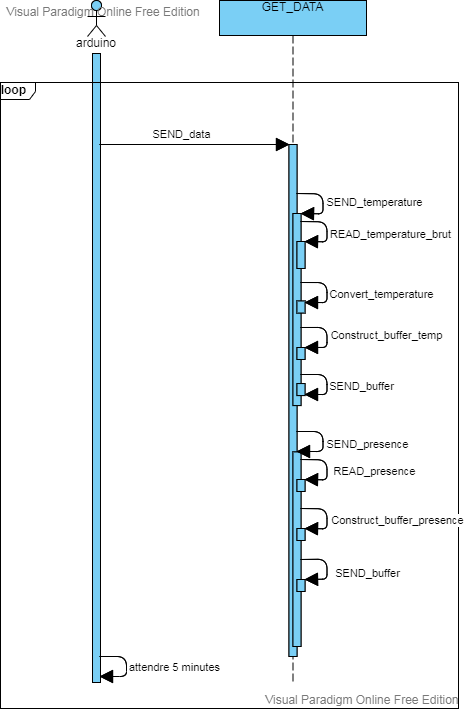
**Diagramme de séquence**

****

La fonction Appliquer Consigne permet de récupérer et de lire les consignes qui seront envoyés depuis la centrale.

Elle lit les données envoyer dans le buffer et les extraits pour les affecter à une variable.

Puis la carte programmable applique les consignes données

****

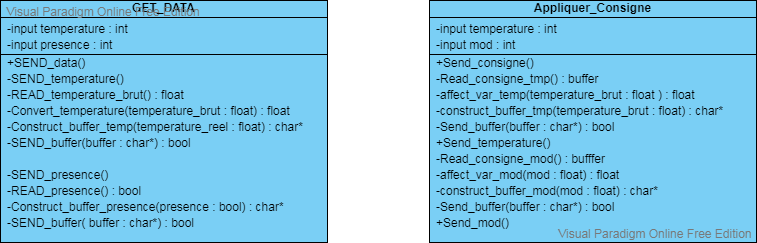
La fonction GET\_DATA récupèrera les informations données par les capteurs.

La fonction tourne dans une boucle qui se renouvelle tous les 5 minutes.

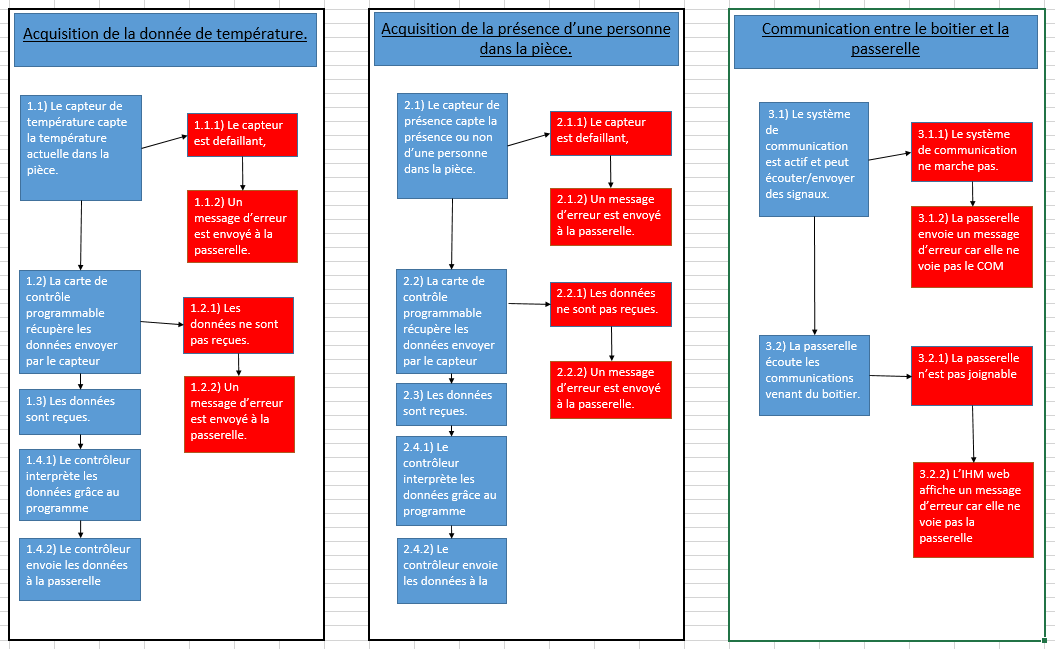
Après avoir récupérer les données, la fonction les traiteras pour transformer les valeurs bruts (V) en valeurs lisibles par l’homme.

Ensuite les valeurs sont envoyés à la passerelle dans 2 buffer distinct.

**Diagramme de classe**

****

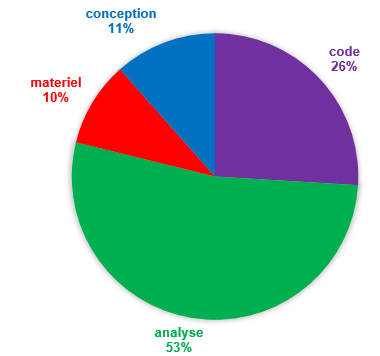
**Scenario**

****

*Suivie du projet*

**

Capture d’écran du Compte Rendu D’activités montrant la répartition des différents types de tache fait en fonction des heures de projet.

**

Pour le moment, sur un total de 108 heures sur les 200 allouer à ce projet :

- 53% ont été consacrées à l’analyse du projet et à la rédaction des diagrammes SYSML

- 26% au code des classes

- 10% à la recherche du matériel pour faire fonctionner le boitier

- 11% pour la conception du boitier

*Conception du boitier*



Un bornier recevra l’alimentation 220V du réseau électrique local et redirigera celui-ci en deux :

- alimentation du radiateur (220V)

- alimentation de la carte (5V)

Un transformateur 220V – 5V permettra d’alimenter la carte programmable

Des relais de puissance permettront de relever l’intensité des sorties analogiques afin de contrôler les fils pilote du radiateur.

*Matériel*

Carte Programmable (annexe page ..)



**Caractéristiques**

* Tension d'alimentation (recommandé) :7-12V
* Entrées/sorties digitales : 14 (dont 6 pouvant être utilisées comme sorties PWM)
* Entrées Analogiques : 6
* Mémoire Flash  : 32 KB (ATmega328) dont 0.5 KB utilisé par le bootloader
* EEPROM :1 KB (ATmega328)

Systèmes de communication (annexe page ..)



**Caractéristiques**

* Fréquence :

2.4 GHz

* Portée :

4000 ft

* Data Rate :

250 kbps

* Sensibilité :

- 100 dBm

* Tension d’alimentation :

2.1 V to 3.6 V

Capteur (annexe page ..)



Capteur de température :

* Tension d’alimentation : 3,3 à 5,5V
* Étendue de mesure : -40 à +125°C
* Précision de mesure : ± 0,25°



Capteur de présence :

* Tension d’alimentation : 5V – 20V
* Tension de sortie : 3V
* Température de fonctionnement : -15 à +70 °C

Actionneur (annexe page ..)



Diode fil pilote :

* Tension : 230VAC
* Le sens de branchement de la diode permet de choisir les modes Arrêt ou hors-gel

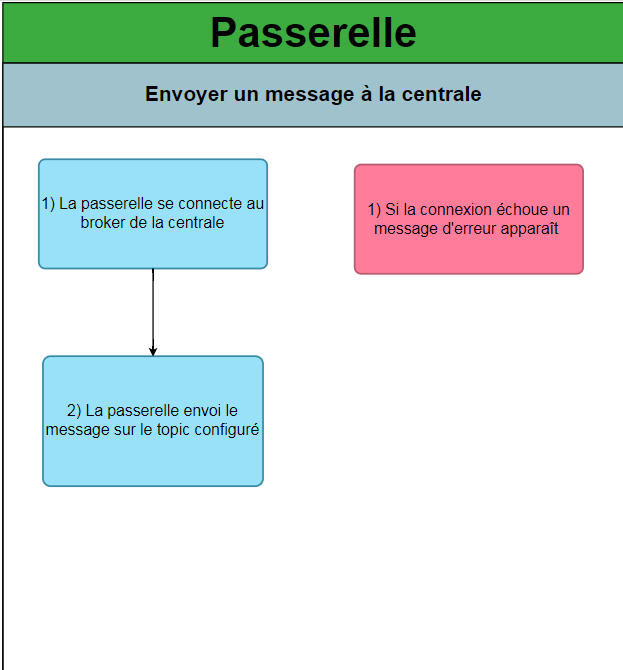
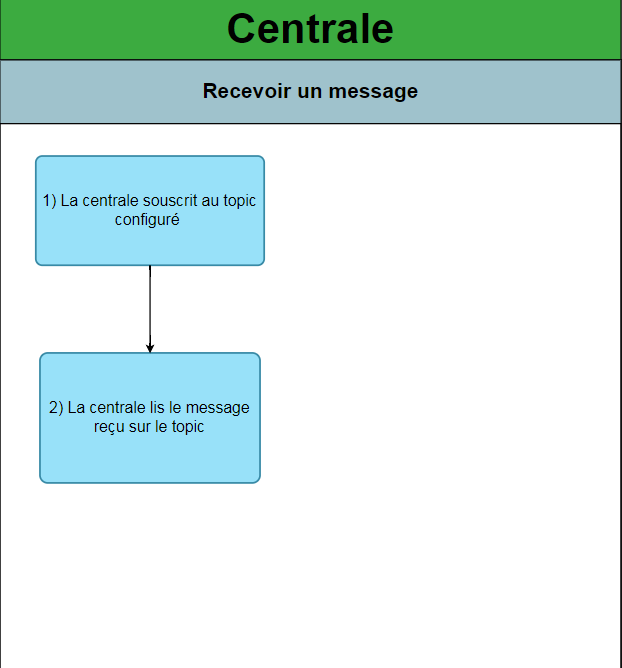
Alimentation du boitier (annexe page ..)

*Code*

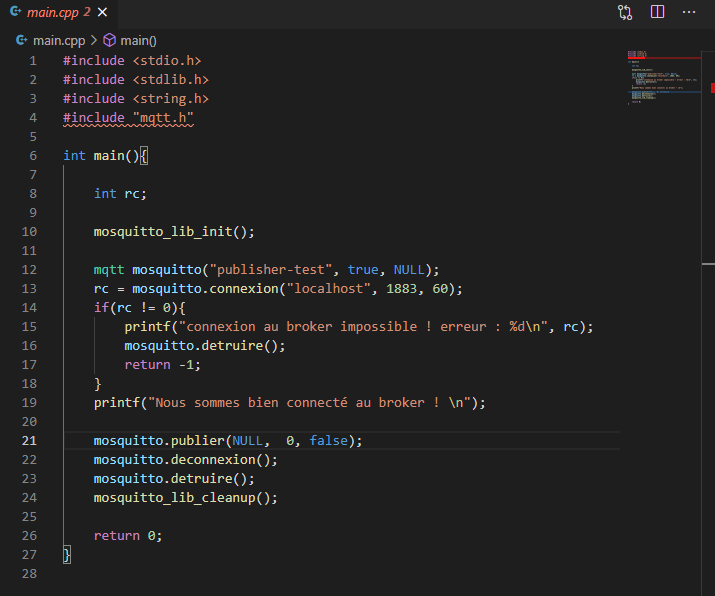
PARTIE 2 : Partie individuel de Wantelez Florian

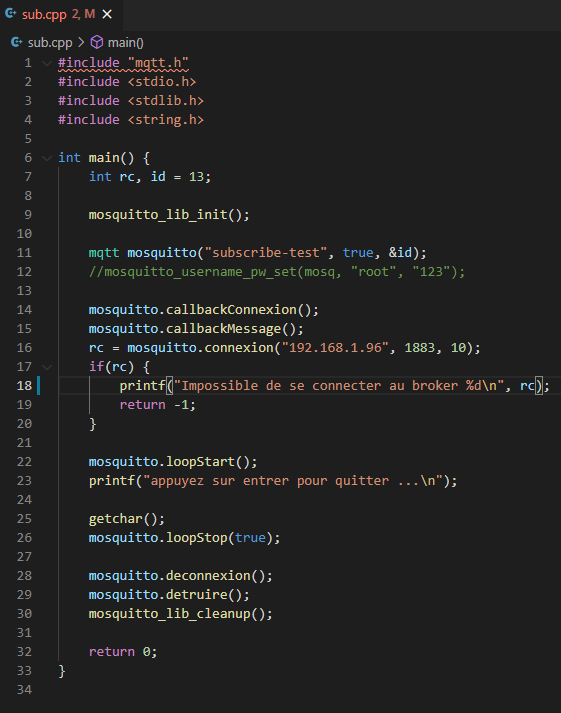
*Analyse*

**Cas d’utilisations**



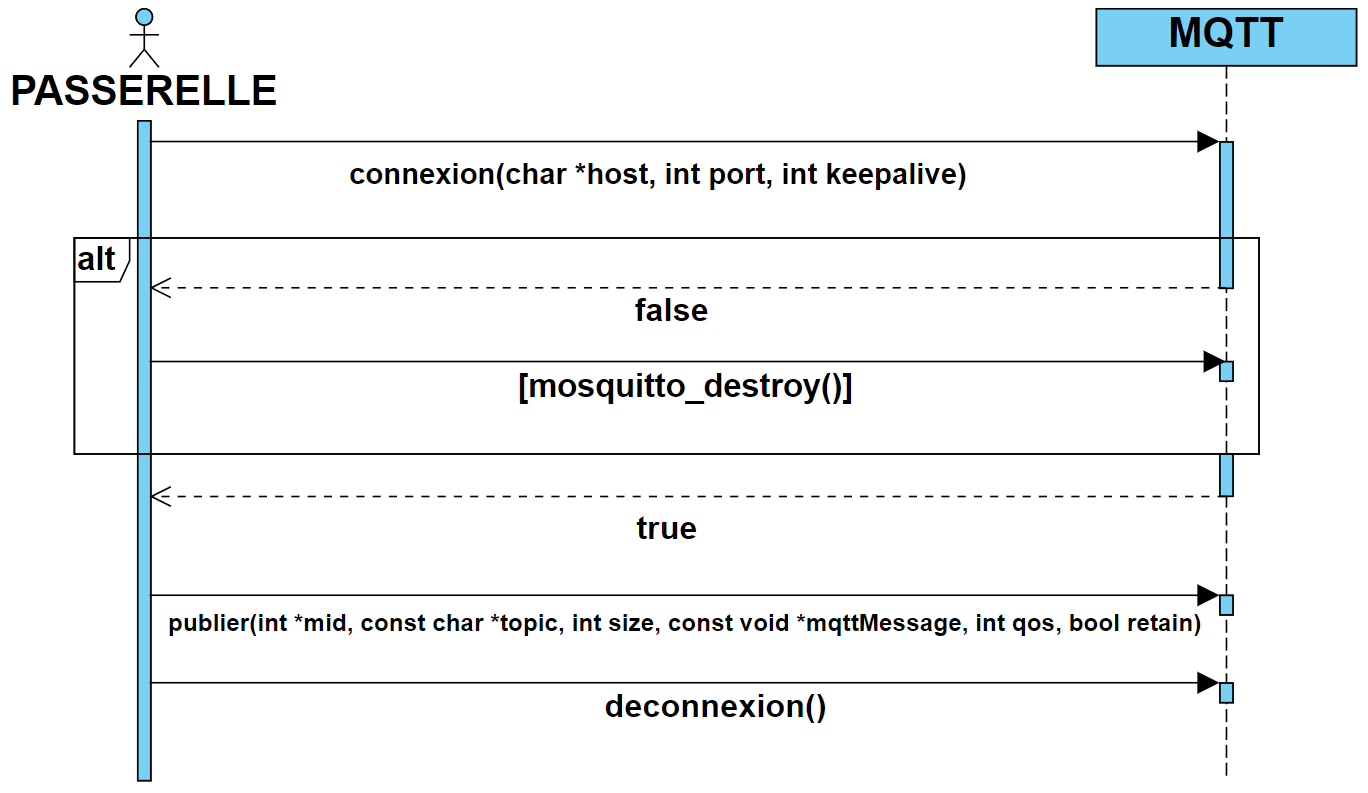
**Code :**

****

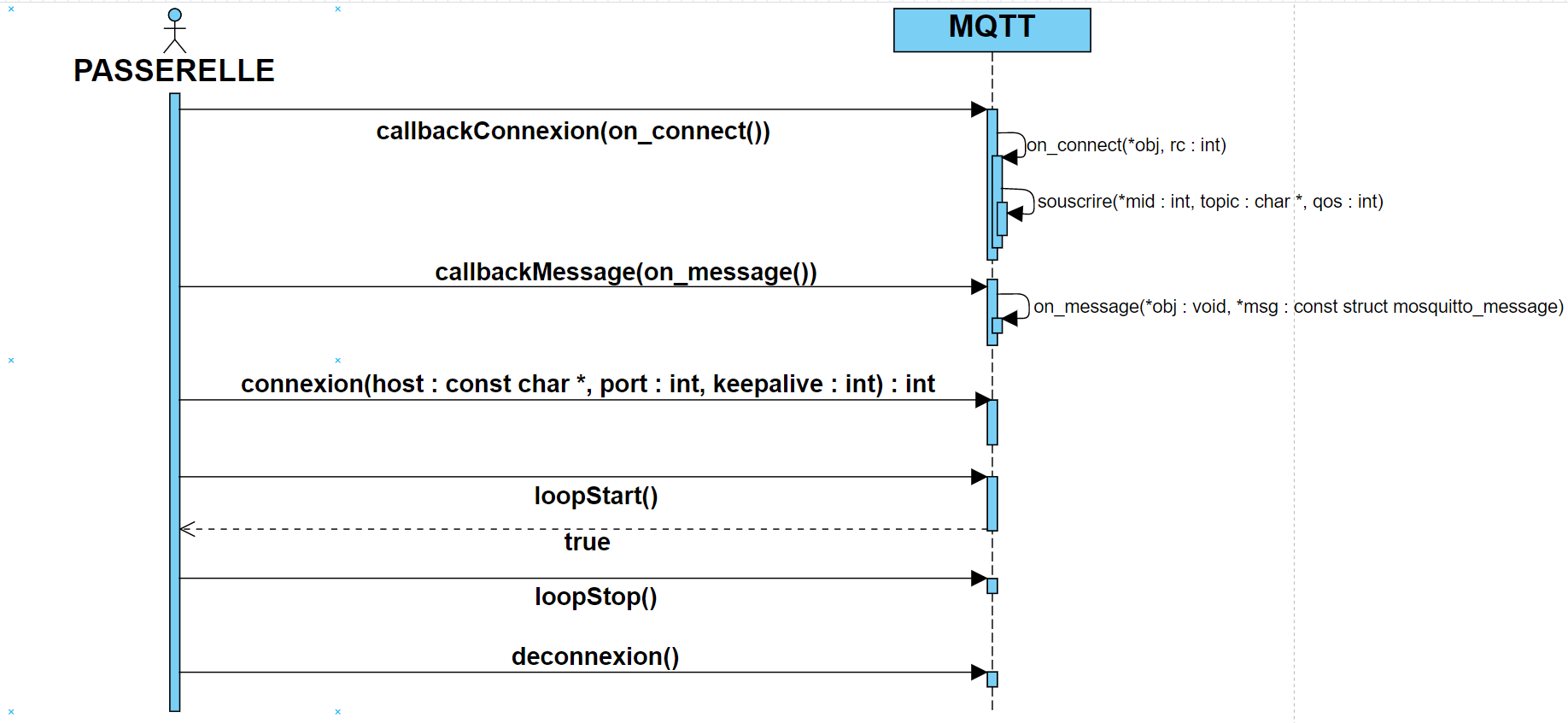


**Séquence :**

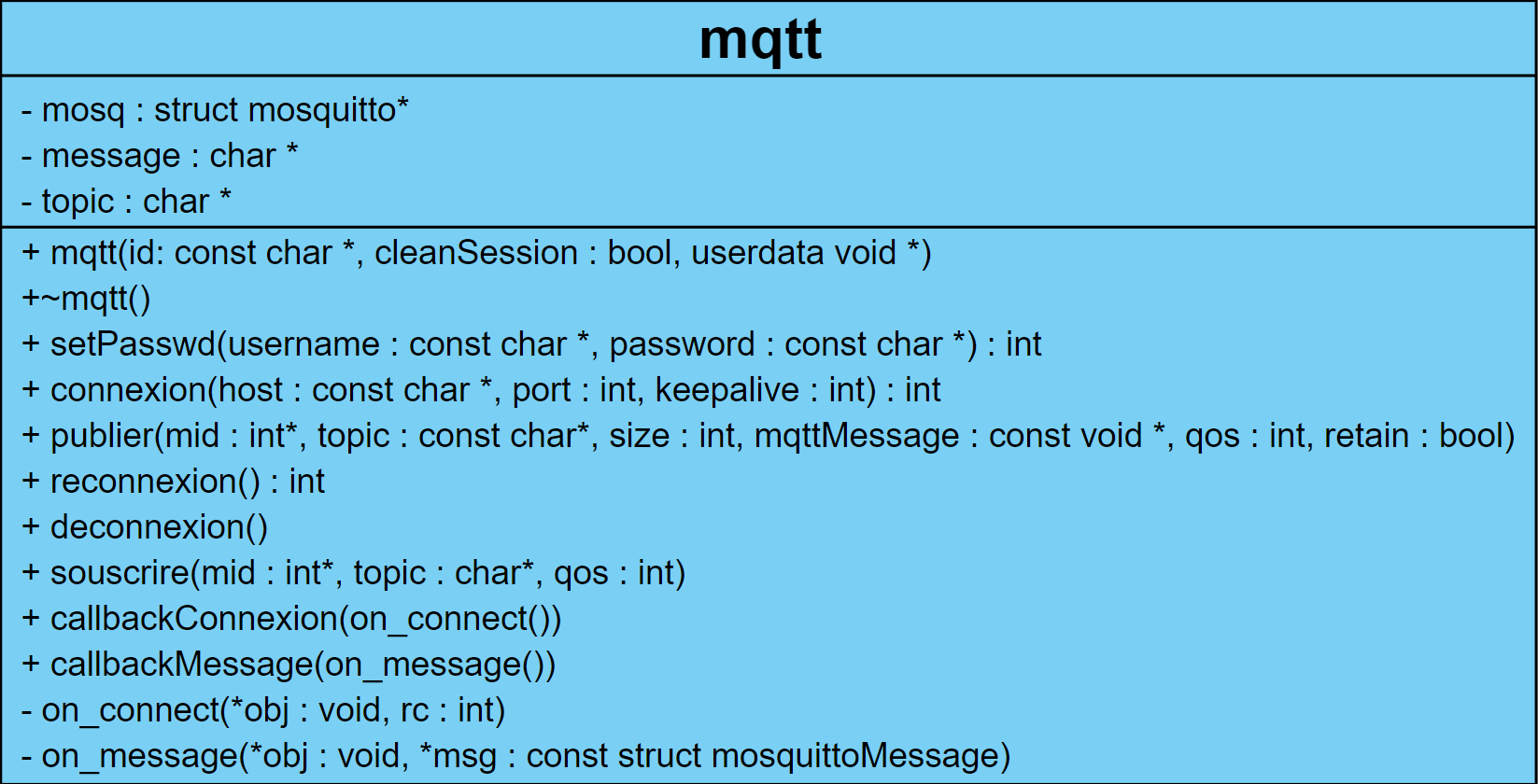
Envoyer un message à la centrale



Recevoir un message de la centrale

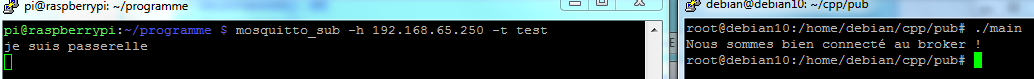


**Classe :**

****

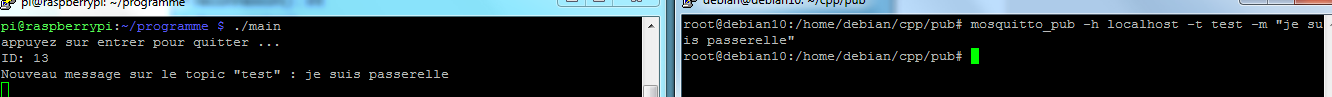
**Test :**

Envoyer un message à la centrale

****

Pour ce test j’utilise la fonction mosquitto\_sub disponible dans le paquet mosquitto (screen de gauche) pour écouter les messages MQTT reçus sur le topic « test » du broker puis je lance mon programme main.cpp avec la commande ./main qui va s’occuper d’envoyer lui-même le message prédéfinis, on peut voir qu’il arrive belle et bien à se connecter au broker et que le message est bien envoyé puisqu’on le reçoit dans la console de gauche

Recevoir un message de la passerelle

****

Pour ce test j’utilise la fonction mosquitto\_pub disponible dans le paquet mosquitto (screen de gauche) pour envoyer un message en MQTT sur le topic « test » du broker puis je lance mon programme sub.cpp avec la commande ./sub qui va s’occuper d’écouter les messages reçu sur les topic configurés, ici le topic prédéfinis est « test » on peut voir qu’il arrive belle et bien à recevoir le message.

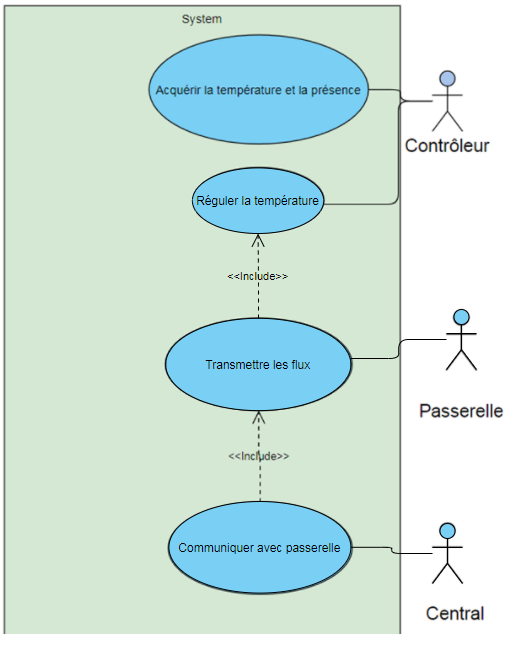
PARTIE 2 : Partie individuel de Malter Alexandre

*Analyse*

# PARTIE 2 : Partie individuel de Garcia Florian

## Analyse

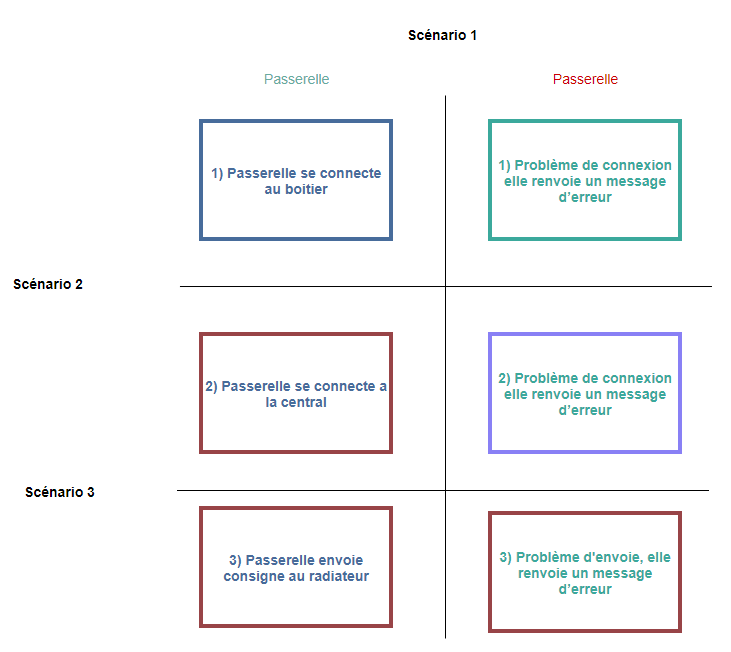
### Cas d’utilisations

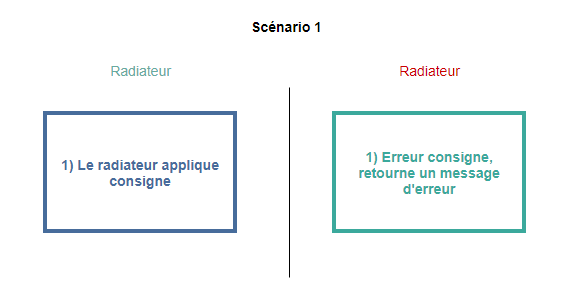


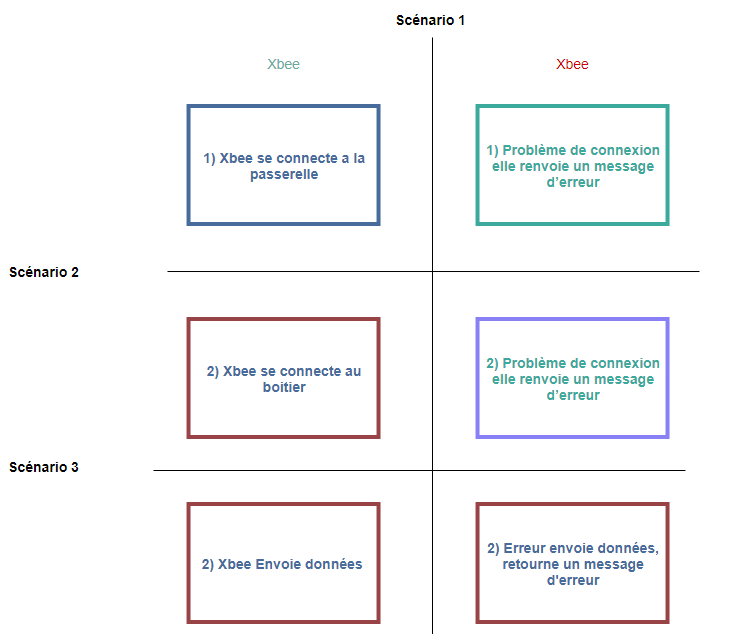
### Diagramme de séquence

### Diagramme de classe

### Scenario





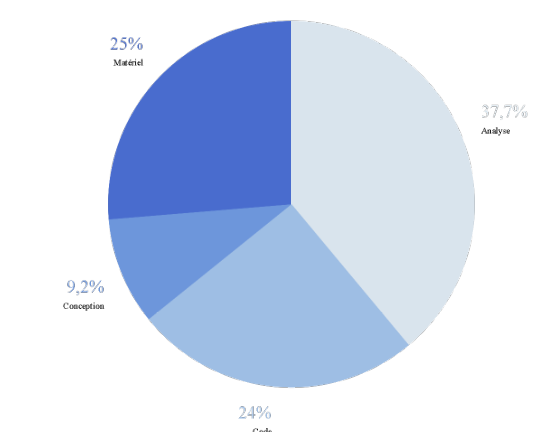


# Suivie du projet

## Compte Rendu D’activités



Le compte rendu d’activités a été effectué sur Excel, les taches sont répartit par heure et date ainsi que par catégorie : Analyse, Matériel, Conception, Code, Prototypage.

**

Sur un total de 108 heures sur les 200:

- 37.7% ont été consacrées à l’analyse du projet et à la rédaction des diagrammes.

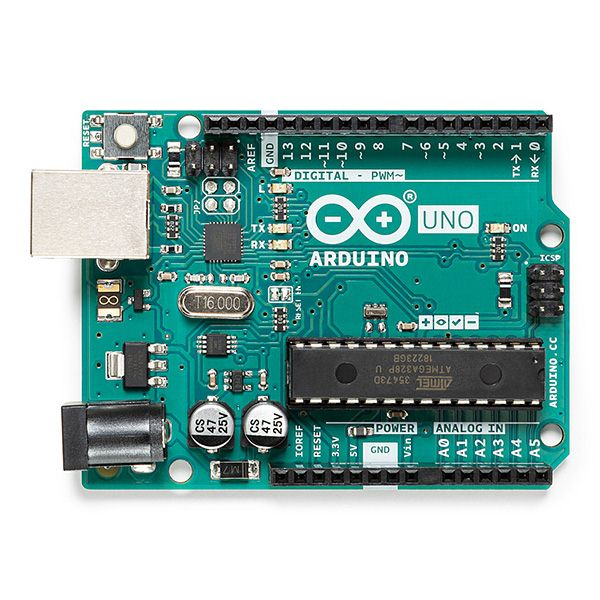
- 24% au code des classes

- 25% à la recherche du matériel.

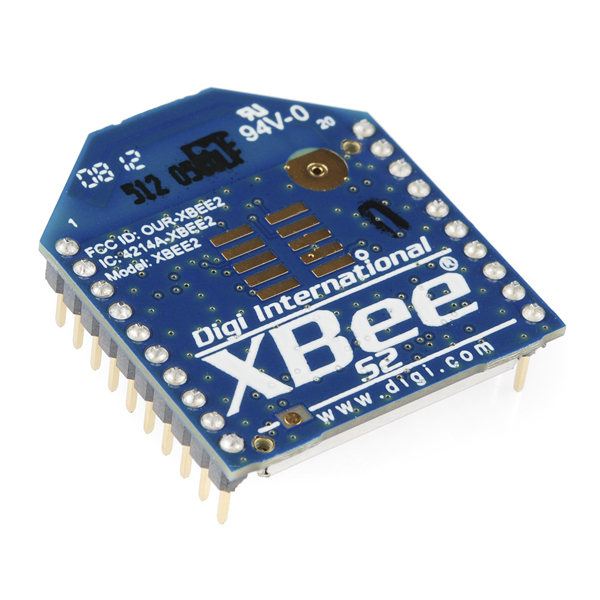
- 9.2% pour la conception.

# Matériel

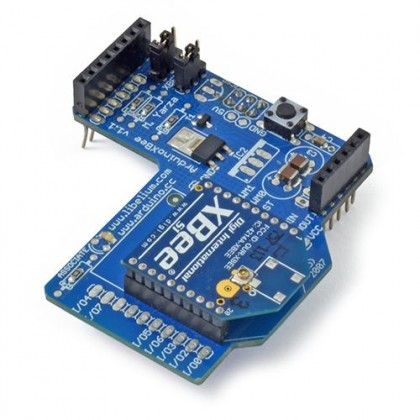
## Carte Programmable

**

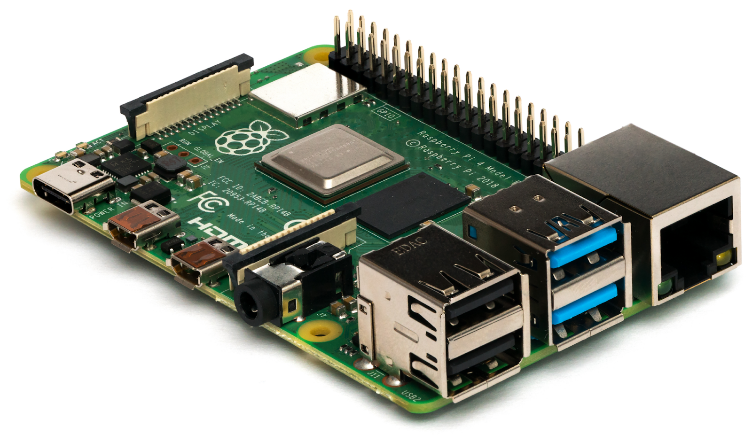
## Systèmes de communication

**

## Xbee shield

**

## Interface graphique

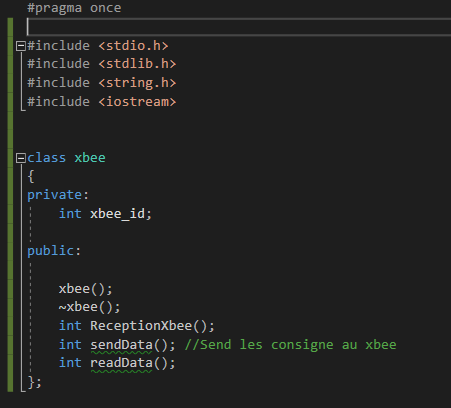
**

# Module de test 1

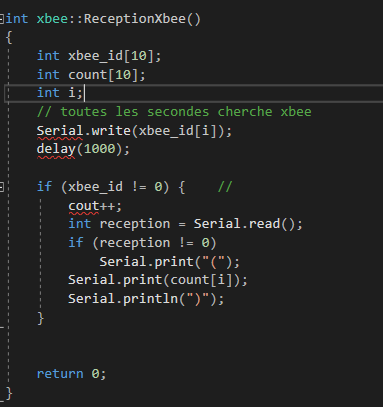


Pour transmettre les flux nous utiliserons carte arduino, module xbee ainsi qu’une raspberry.

## Classe xbee :



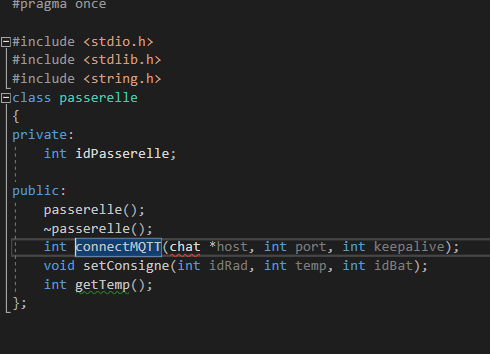
## Xbee.cpp



Tous les une seconde le programme cherche les xbee et permet de réceptionner l’id du xbee et le message.

## Module de test 2

## Passerelle.h



Cette classe permet d’envoyer les données a la central.

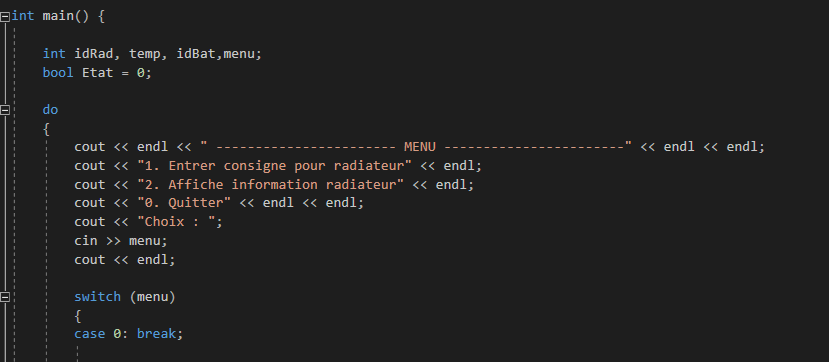
La fonction setConsigne()=> permet de donner une température au radiateur souhaité.

## Passerelle.cpp

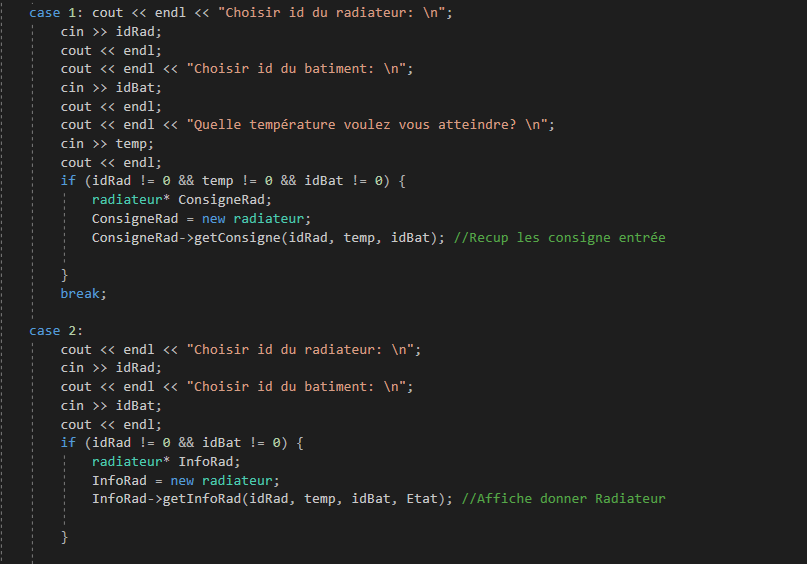
La fonction permet d’envoyer les consignes au xbee.

Main.cpp

## Menu



## Choix programme :



Le menu nous permet de choisir l’id du radiateur et du bâtiment pour appliquer une consigne (température).