INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# BE C++
# DomoHouse

Rapport sur le BE de C++

*Date : 19 mai 2021*

*4ème année AE-SE*


Étudiants :

- ➔ Choulot Romain
- ➔ Meyer Carole
- ➔ Pitault Léa

Enseignant :

- Gauchard David

# Introduction

L'objectif de ce Bureau d'Etude est d'utiliser un microcontrôleur ESP8266 programmable en Arduino afin de lui ajouter des capteurs et des actionneurs dans le cadre du cours de C++ de 4ème année du Génie Electrique et Informatique de l'INSA de Toulouse. Au cours de ce projet, nous avons dû développer une bibliothèque modulaire, extensible et capable de s'interfacer avec d'autres capteurs et actionneurs. Pour ce faire, l'utilisation du langage C++ est une évidence. L'utilisation de ce langage permettra d'utiliser le mécanisme d'héritage, la redéfinition d'opérateurs, la librairie STL et des exceptions. Pour le git, le lien est le suivant : https://github.com/carolemeyer/project-cpp.git

# I. Le projet DomoHouse dans son ensemble

Le projet DomoHouse est basé sur une carte ESP8266 dotée d'une module Wi-Fi et d'un shield Grove associé permettant de brancher des capteurs et actionneurs Grove sur la carte. Ce projet est composé de trois fichiers seuls et de cinq modules *.h* et *.cpp*. Voici à quoi ces fichiers correspondent :

- **domohouse.ino** : c'est l'équivalent du main.cpp, ce fichier permet de contrôler tout le projet, toutes les variables et fonctions que l'on souhaite utiliser,
- **classes.h** : définit les classes de base pour les capteurs et les actionneurs, qu'ils soient analogiques ou numériques,
- **includes.h** : chargé d'inclure toutes les bibliothèques nécessaires au projet afin de ne pas toutes les redéfinir à chaque fois dans chaque fichier, et nous permet de définir les numéros de pin et d'autres valeurs utiles au projet,
- **module 'kozy'** : regroupe des classes de plusieurs niveaux, à savoir les classes de base *Light*, *Led* et *Speaker*. Ce module propose aussi des classes dédiées à notre projet utilisant les classes de base, comme *MyAlarm* et *MyMood*. La classe *MyFridge*, qui n'hérite pas d'autres classes, est aussi implémentée ici,
- **module 'zedoor'** : regroupe les classes de base *MonTouchSensor* et *MonServo*, qui sont ensuite utilisées par leurs classes dédiées *MaPorte* et *DoorProject*.
- **module 'weblink'** : s'occupe de la gestion de la connectivité wifi en proposant la mise en place d'un serveur web pour accéder à la page web dédiée de la DomoHouse,
- **module 'IFTTTFeed'** : contient le code nécessaire pour se connecter, souscrire et suivre le fil d'information du voisinage, un feed, qui nous avertira de toute intrusion chez nos chers voisins.

Voici la liste des capteurs et actionneurs utilisés dans ce projet :

| **Capteurs** | | **Actionneurs** | |
|---|---|---|---|
| **Analogique** | **Digital** | **Analogique** | **Digital** |
| Température | Lumière | | LEDs |
| | Toucher | | Haut-parleur / Buzzer |
| | | | Servo-moteur |

## II. Les différents modes de la DomoHouse

La DomoHouse peut fonctionner selon deux modes distincts : le mode *Normal* ou le mode *Domotik*. L'utilisateur peut choisir l'un ou l'autre des modes en modifiant la ligne *"#define MODE 0"* située dans le fichier *domohouse.ino*, dont voici les caractéristiques :

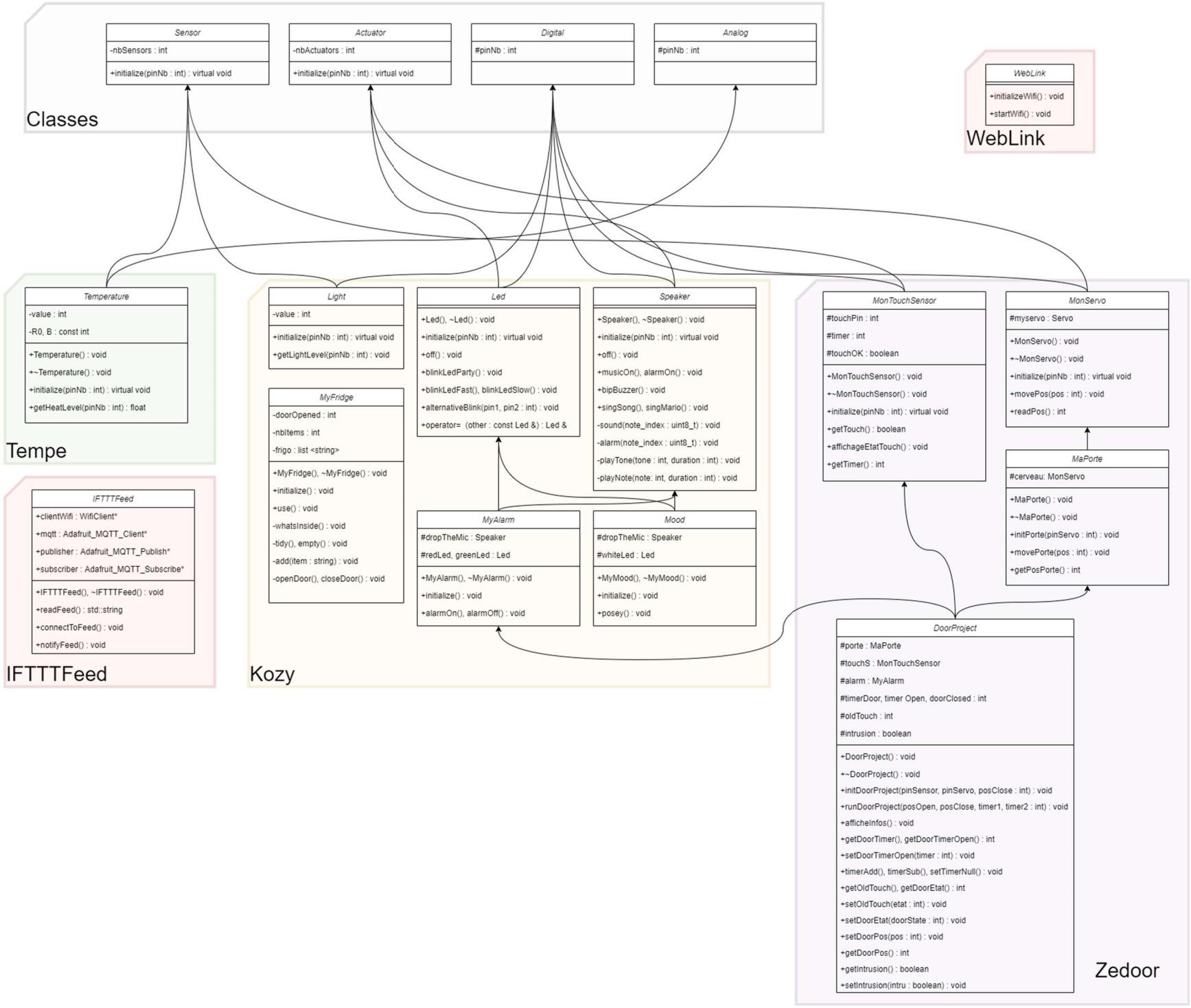
- Dans le mode *Normal* (*Mode* = 0), vous pourrez :
  - Utiliser et ajouter des choses dans le frigo 🍕
  - Activer les lumières et jouer une jolie musique en faisant monter la température (il faut prendre le module entre ses mains 🥵)
  - Appuyer sur le capteur de toucher :
    - Une seconde environ pour ouvrir la porte
    - Plusieurs secondes pour déclencher l'alarme
  - Choisir l'angle d'ouverture de la porte, selon deux modes différents :
    - Mode 1 : La porte s'ouvrira avec un angle de 90°
    - Mode 2 : La porte s'ouvrira avec un angle de 45°
    - Toute autre valeur rentrée par l'utilisateur lèvera une exception

- Dans le mode *Domotik* (*Mode* = 1) :
  - Vous pourrez accéder à la maison depuis un site internet, et voir ce qui se passe dedans. Pour ce faire :
    - Activez un hotspot wifi (partage de connexion) sur votre téléphone ou ordinateur
    - Entrez l'identifiant et mdp dans le fichier weblink.cpp de cette sorte :
      - const char* ssid = "name-of-your-wifi-network"
      - const char* password = "password-of-your-wifi-network"
    - Ouvrez le moniteur série, qui vous donnera une adresse IP
    - Entrez cette même adresse IP sur votre navigateur favori
    - Enjoy !
  - Si une intrusion est détectée dans la maison d'une autre équipe, l'information sera transmise à la DomoHouse, ce qui déclenchera l'alarme

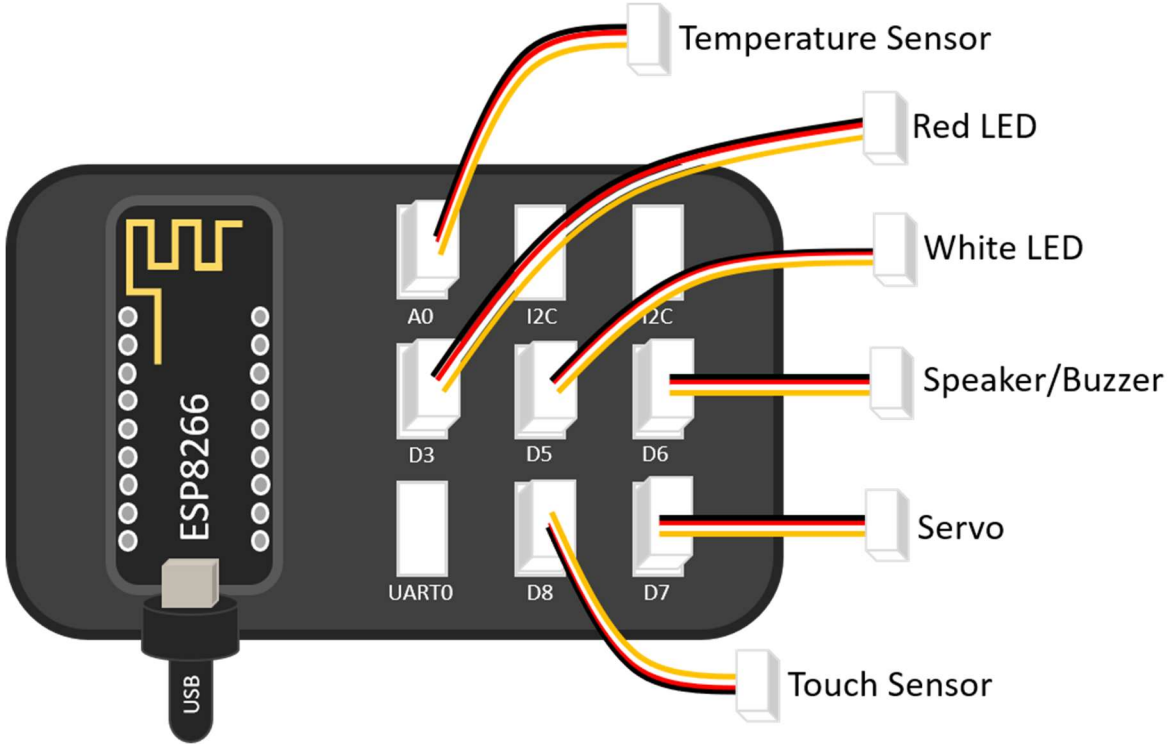
## III. Le cahier des charges

Durant ce BE, nous avions un cahier des charges que l'on devait respecter, voici donc les fichiers correspondant aux exigences :

- Création de plusieurs classes : *classes.h*, module *'kozy'*, module *'tempe'*, module *'zedoor'.*
- Utilisation du mécanisme d'héritage : module *'kozy'*, module *'tempe'*, module *'zedoor'*
- Redéfinition d'opérateur : dans la classe *Led* du module *'Kozy'* redéfinition de l'opérateur= pour les Leds par *"Led & operator= (const Led & other);".*
- Utilisation de la STL : dans la classe *MyFridge* du module *'Kosy'* car le contenu du frigo est stocké sous forme de liste.
- Utilisation des exceptions : exception levée sur le choix de l'angle d'ouverture de la porte par l'utilisateur à travers le moniteur série, dans le setup de *domohouse.ino*


Classes
Sensor
-nbSensors : int
+initialize(pinNb : int) : virtual void
Actuator
-nbActuators : int
+initialize(pinNb : int) : virtual void
Digital
#pinNb : int
Analog
#pinNb : int
WebLink
+initializeWifi() : void
+startWifi() : void
WebLink
Temperature
-value : int
-R0, B : const int
+Temperature() : void
+~Temperature() : void
+initialize(pinNb : int) : virtual void
+getHeatLevel(pinNb : int) : float
Tempe
IFTTTFeed
+clientWifi : WifiClient*
+mqtt : Adafruit_MQTT_Client*
+publisher : Adafruit_MQTT_Publish*
+subscriber : Adafruit_MQTT_Subscribe*
+IFTTTFeed(), ~IFTTTFeed() : void
+readFeed() : std::string
+connectToFeed() : void
+notifyFeed() : void
IFTTTFeed
Light
-value : int
+initialize(pinNb : int) : virtual void
+getLightLevel(pinNb : int) : void
MyFridge
-doorOpened : int
-nbItems : int
-frigo : list <string>
+MyFridge(), ~MyFridge() : void
+initialize() : void
+use() : void
-whatsInside() : void
-tidy(), empty() : void
-add(item : string) : void
-openDoor(), closeDoor() : void
Led
+Led(), ~Led() : void
+initialize(pinNb : int) : virtual void
+off() : void
+blinkLedParty() : void
+blinkLedFast(), blinkLedSlow() : void
+alternativeBlink(pin1, pin2 : int) : void
+operator= (other : const Led &) : Led &
Speaker
+Speaker(), ~Speaker() : void
+initialize(pinNb : int) : virtual void
+off() : void
+musicOn(), alarmOn() : void
+bipBuzzer() : void
+singSong(), singMario() : void
-sound(note_index : uint8_t) : void
-alarm(note_index : uint8_t) : void
-playTone(tone : int, duration : int) : void
-playNote(note: int, duration : int) : void
MyAlarm
#dropTheMic : Speaker
#redLed, greenLed : Led
+MyAlarm(), ~MyAlarm() : void
+initialize() : void
+alarmOn(), alarmOff() : void
Mood
#dropTheMic : Speaker
#whiteLed : Led
+MyMood(), ~MyMood() : void
+initialize() : void
+posey() : void
Kozy
MonTouchSensor
#touchPin : int
#timer : int
#touchOK : boolean
+MonTouchSensor() : void
+~MonTouchSensor() : void
+initialize(pinNb : int) : virtual void
+getTouch() : boolean
+affichageEtatTouch() : void
+getTimer() : int
MonServo
#myservo : Servo
+MonServo() : void
+~MonServo() : void
+initialize(pinNb : int) : virtual void
+movePos(pos : int) : void
+readPos() : int
MaPorte
#cerveau: MonServo
+MaPorte() : void
+~MaPorte() : void
+initPorte(pinServo : int) : void
+movePorte(pos : int) : void
+getPosPorte() : int
DoorProject
#porte : MaPorte
#touchS : MonTouchSensor
#alarm : MyAlarm
#timerDoor, timer Open, doorClosed : int
#oldTouch : int
#intrusion : boolean
+DoorProject() : void
+~DoorProject() : void
+initDoorProject(pinSensor, pinServo, posClose : int) : void
+runDoorProject(posOpen, posClose, timer1, timer2 : int) : void
+afficheInfos() : void
+getDoorTimer(), getDoorTimerOpen() : int
+setDoorTimerOpen(timer : int) : void
+timerAdd(), timerSub(), setTimerNull() : void
+getOldTouch(), getDoorEtat() : int
+setOldTouch(etat : int) : void
+setDoorEtat(doorState : int) : void
+setDoorPos(pos : int) : void
+getDoorPos() : int
+getIntrusion() : boolean
+setIntrusion(intru : boolean) : void
Zedoor

## V. Schéma de branchement du matériel




## Conclusion

Pour conclure, ce BE était vraiment intéressant, alliant Arduino et C++, qui sont tous les deux très utilisés par les ingénieurs. De plus, cela nous a permis d'avoir une vision plus globale pour notre code, car l'on doit prévoir les utilisations futures de nos classes et de nos fonctions. Celles-ci doivent pouvoir être utilisées dans de nombreuses configurations, donc notre code devait être le plus souple possible, tout en respectant notre cahier des charges imposé pour ce BE, et qui n'était pas non plus chose facile.

Les perspectives d'évolution de ce programme sont conséquentes. En effet, non seulement c'est du C++, qui est un langage connu par tous, mais notre cahier des charges était de faire en sorte que notre code puisse être évolutif, adaptatif et répétitif. Si l'on veut connecter plusieurs modules du même type, il n'est pas nécessaire de refaire des fonctions ou des initialisations, il suffit juste de déclarer un nouvel objet appartenant à cette classe et tout est bon !

Nous n'avons rencontré que très peu de problèmes et l'utilisation de StackOverflow nous a grandement aidé, par exemple pour trouver comment autoriser la gestion d'exception sur l'IDE.