Simon Jourdenais
&
William Bouchard
Technologies du génie électrique
Groupe 2317

Manuel Technique du Projet Environnement Contrôlé

Travail présenté à
M. Benoit Beaulieu,
M. Richard Cloutier et M. Julien Bosco
Département du génie électrique
Pour le cours de
Projet de fin d'études

Cégep de Sherbrooke
Mai 2021

Table des matières

1 Description du projet

1.1 Introduction

Dans le but de rendre le jardinage plus accessible et un peu plus adapté à nos situations, notre équipe avons décidé de créer un système permettant de monitorer l'état d'une serre et de contrôler le climat interne. Ce système permettrait donc d'automatiser une petite serre, pour que quiconque puisse récolter des légumes directement chez eux. De cette façon, si on est trop occupé pour s'occuper du potager, il n'y aurait aucun problème pour notre système. Même y pour faire pousser des plantes plus exotiques ou des champignons ne serait pas difficile.

1.2 Description du projet

Le produit fait donc l'acquisition des données météorologiques et contrôle le climat d'une serre. Il comporte un boitier qui sert de contrôleur qui doit être placé près de la serre, ainsi qu'un autre plus petit boitier, servant comme boitier protecteur pour le circuit de capteurs. Le produit, en faisant le monitorage de la température, l'humidité, la concentration en dioxyde de carbone , peut contrôler le degré d'humidité et la température de la serre à l'aide des éléments reliés au boitier de contrôle. Toutes ces données sont affichées sur un écran sur le boitier dans lequel le projet est monté et sont à la fois sauvegardées les données en ligne.

La sauvegarde des données se fait sur la plateforme infonuagique ThingSpeak afin de rendre disponible un historique de données simple accessible en tout temps.

2 Fonctionnement du produit

2.1 Fonctionnement général

Pour expliquer un peu plus en détail, le produit comporte une boite de contrôle et un bloc de capteurs, appelé "Boite de contrôle" et "Sensorblock" respectivement. Ces deux pièces sont le minimum pour faire fonctionner le projet.

Une fois le bloc de capteurs branché au Raspberry Pi de la boite de contrôle avec un câble mini USB, il pourra capter les données de température, d'humidité, de concentration en dioxyde de carbone et l'indice de qualité d'air et les envoyer à la boite de contrôle.

Sur l'écran de la boite, le programme d'interface usager affiche un menu interactif comportant quatre onglets. C'est dans l'onglet "Monitorage" que les données captées dans la serre par le bloc de capteurs sont affichées. L'onglet "Consignes" permet de voir et changer les consignes de température et d'humidité. Le troisième onglet, "Paramètres" permet de configurer le compte ThingSpeak sur lequel les données sont enregistrées. La boite au bas de la fenêtre affiche des informations de débogage permettant la configuration du projet. Le dernier onglet de l'interface usager permet de tester les différents éléments branchés à la boite de contrôle et de voir leurs états respectifs, soit allumés ou éteints.

Le produit fonctionne de la manière suivante : une fois installé et configuré, l'usager entre les consignes de température et d'humidité que le contrôleur doit respecter. Aussitôt entré, le programme vérifie si les données de température et humidité respectent leur consigne respective, puis agit en activant l'élément nécessaire, soit le tapis chauffant, soit l'aération, soit l'humidificateur.

Chaque deux secondes, une nouvelle mesure de données de la serre est prise, puis à chaque 15 secondes, une sauvegarde sur Thingspeak est effectuée.

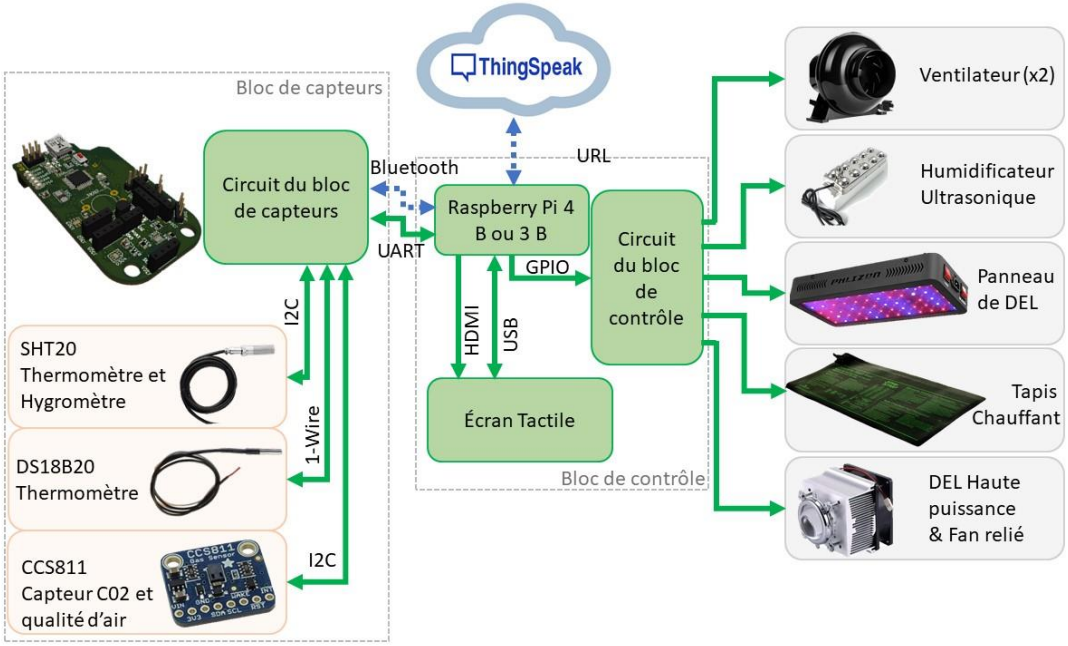


Schéma 1 Schéma de fonctionnement du projet

2.2 Fonctionnement détaillé

Le schéma 1 démontre les liens entre les éléments généraux du projet et apporte plus de détails au propos énoncé plus tôt.

Le bloc de capteurs est la partie du projet qui permet de faire l'acquisition. C'est un des deux éléments clés du projet. Il effectue la lecture des capteurs à chaque deux secondes, le premier thermomètre, le DS18B20, par protocole 1-Wire, le deuxième, le SHT20, un hygromètre et thermomètre en un par protocole I2C et finalement le capteur de gaz CCS811 par protocole I2C aussi. Il envoie ensuite par communication série par l'entremise du fil USB, ou d'une connexion Bluetooth configurée préalablement, les données reçues des capteurs.

Le bloc de contrôle reçoit alors les données, les vérifie, puis les affiche si elles sont valides. Une fois l'affichage fait, il vérifie si la température ou l'humidité n'est pas en règle selon la consigne définie par l'usager. Le microcontrôleur au cœur du bloc de contrôle active une de ses sorties tout usage pour activer le relais à semi-conducteurs relié à l'élément nécessité, comme le tapis chauffant, si la température venait à descendre sous la consigne de température.

Une fois aux quinze secondes, le programme sauvegarde les données sur la base de données infonuagique Thingspeak par l'entremise d'une requête URL et dépend d'une clé entrée durant la configuration du produit.

3 Contenu matériel du projet

Le schéma ci-dessous démontre les connexions matérielles du projet afin de mieux comprendre le contenu matériel du projet. Comme dans le schéma de fonctionnement, on y retrouve le bloc de capteurs et le bloc de contrôle.

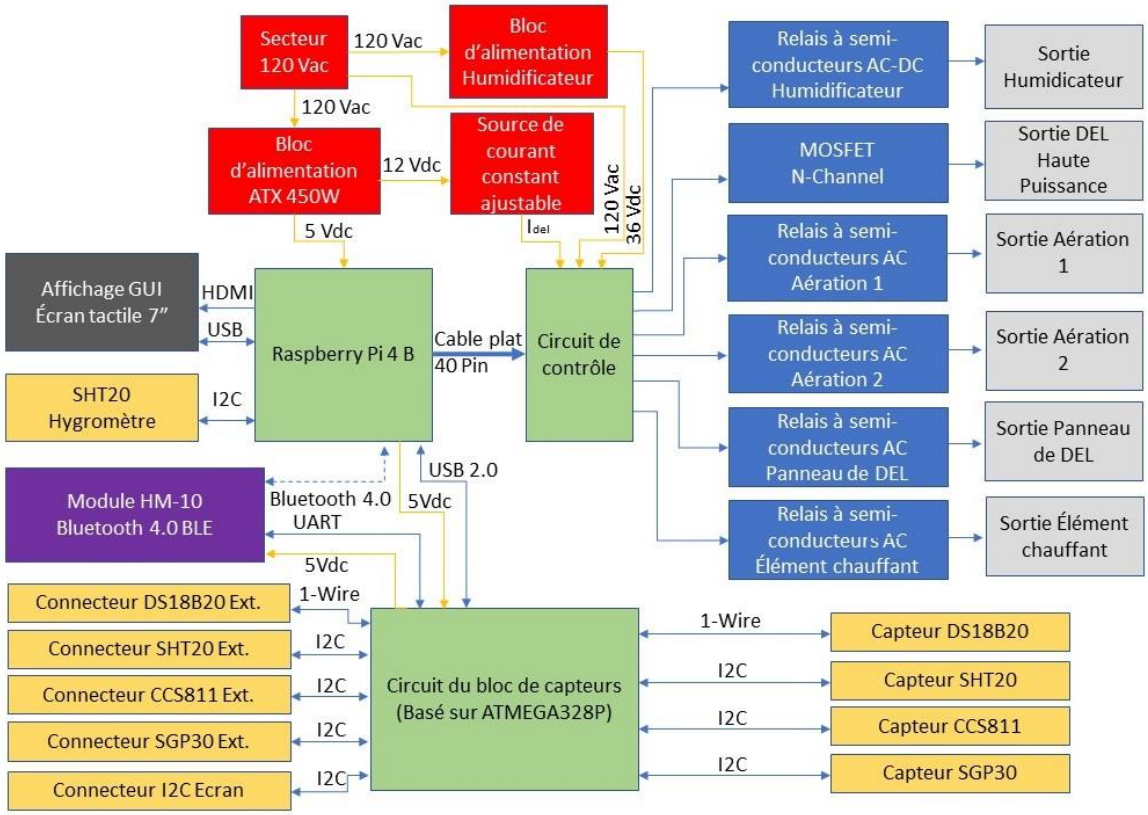


Schéma 2 Schéma bloc du projet environnement contrôlé

3.1 Circuit de contrôle

Le circuit de contrôle est un circuit fait par notre équipe et produit par JLCPCB. Le schéma complet des circuits du projet est annexé à la fin de ce fichier, consultez la table des matières au besoin.

Pour contrôler les composantes, la carte prend les commandes du Raspberry Pi par l'entremise de son connecteur de 40 pins. Le circuit de contrôle comporte donc un "*header*" de 40 pins compatible avec Raspberry Pi. Ce circuit de contrôle est en fait un module d'extension ("Shield"), étendant les fonctionnalités des sorties toutes usages du Raspberry Pi.

Les sorties du Raspberry Pi peuvent contrôler jusqu'à quatre sorties de 120 volts en courant alternatif, une sortie de 60 volts de courant continu ou courant alternatif , trois sorties de 12 volts CC et une sortie dont la tension est variable (J411). Elle est dite variable dans le sens où c'est l'usager qui choisit la tension qui

y sera appliquée en injectant cette tension dans l'entrée J414. Elle doit cependant être du courant continu.

Les sorties à courant alternatif (120V) sont gérées par des relais à semi-conducteurs CPC1966B. Comme le montre la figure de droite, ces quatre composantes sont U204, U205, U206 et U207.
Leur fonctionnement est extrêmement simple. Injecter un courant de 5 milliampères sur son entrée permet de fermer le circuit relié à ses bornes de sortie. Le CPC1907B (U208) fonctionne exactement de la même manière, mais est limité à 60 volts. Il fonctionne toutefois en courant continu comme en courant alternatif. Il fait partie du circuit et a été spécialement choisi pour contrôleur l'humidificateur ultrasonique qui nécessitait plus d'ampérage que peuvent sourcer les CPC1966B. Ils sont limités à 3 ampères, alors que le CPC1907B fait jusqu'à 5 ampères.
Nécessitant très peu de courant, ces relais sont donc contrôlés directement par les GPIO du Raspberry Pi à 3.3 volts. Ils ne produisent aucun bruit et ne s'abiment pas comme les relais réguliers.

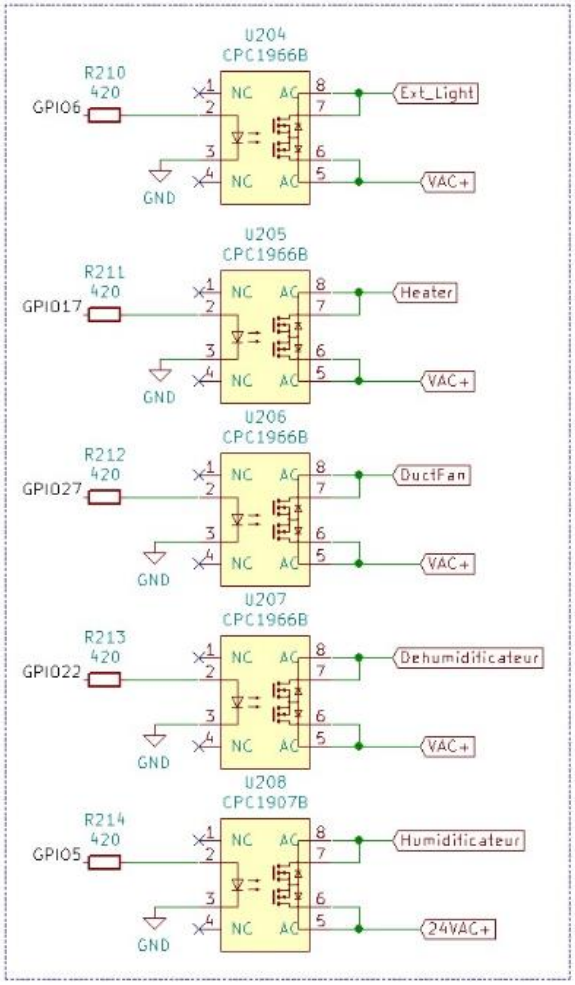


Figure 1 Relais à semi-conducteurs

Note : Il a été choisi puisque le vendeur de l'humidificateur revendiquait que son produit nécessitait 24 volts de courant alternatif. Toutefois, il nécessite 36 volts et aurait pu être contrôlé par un MOSFET. Par chance, le CPC 1907B fait aussi pour du courant continu.

Ce qui fait que les sorties que gèrent les relais CPC1966B ont une puissance maximale de 360 watts chacune et le CPC1907B 300 watts. Cependant, il n'est pas conseillé d'utiliser des appareils de plus de 200 watts, car le prototype n'a pas été testé à ce point!

Les sorties J401, J402, J403 et J411 sont tous contrôlés par des MOSFET. Ceux qui ont été utilisés dans le projet sont de type N, numéro de modèle FQP30N06L. La raison derrière le choix de cette composante est le fait que c'est un MOSFET

fonctionnant à niveau de logique. C'est-à-dire qu'il s'active avec un état logique haut, parfaitement convenable avec les sorties de 3.3 volts d'un Raspberry Pi, sans avoir besoin de convertir la tension à un niveau plus élevé.

Les MOSFET sont utilisés pour gérer les sorties à courant continu. Ces transistors contrôlent le ventilateur du boitier(U202), le ventilateur du dissipateur de chaleur de la DEL haute puissance(U201), le ventilateur de l'humidificateur (U203) et la DEL haute puissance (U209).

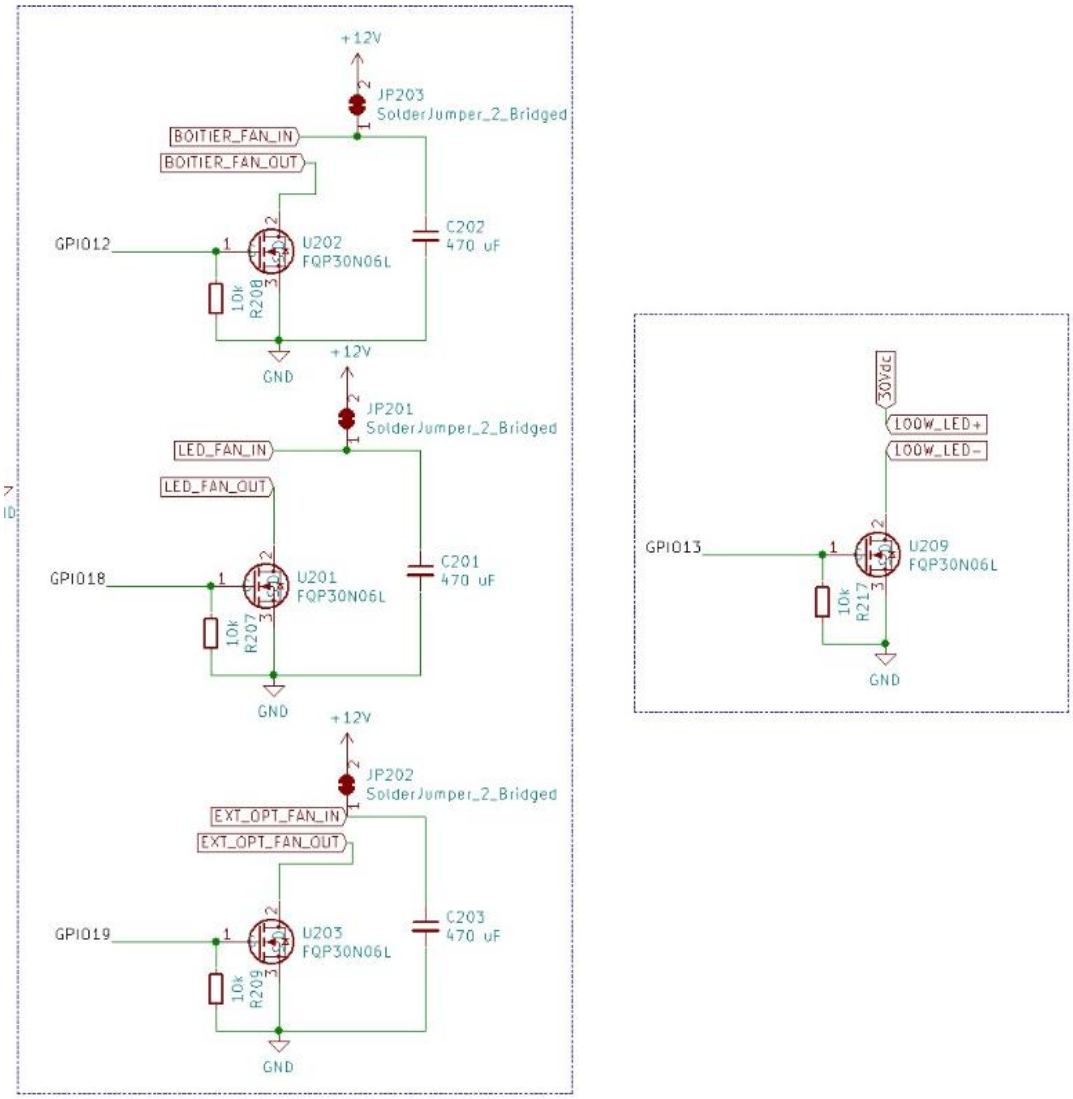
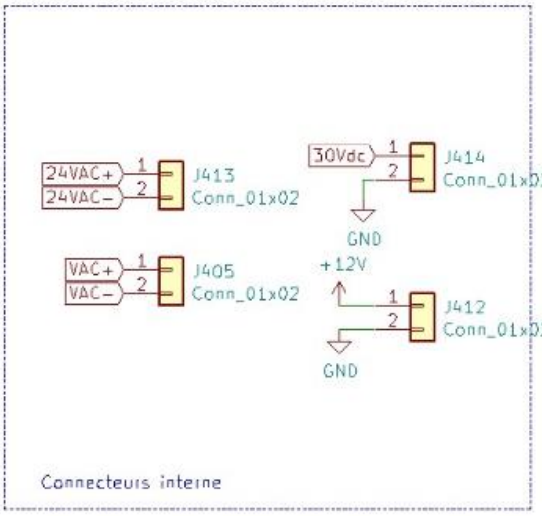


Figure 2 MOSFET du circuit

Comme indiqué précédemment, la tension de la sortie J411 est variable.



La tension que reçoit U209 vient de l'entrée J414. Elle peut donc être variée pour accommoder la DEL branchée sur la sortie DEL haute puissance. Le circuit et ses composantes sont ceux qui dictent la limite de la tension/courant qui peut être injectée dans cette entrée.
Le MOSFET qui gère la sortie ne peut dépasser la tension maximale de 60 volts (courant continu).

Dans le cas du projet, la tension qui est injectée dans cette entrée est environ 30 volts, soit la tension nécessitée par la diode électroluminescente de haute puissance.

3.2 Circuit de capteurs

Le circuit de capteurs est basé sur un ATMEGA328P-AU puisqu'il est simple d'utilisation et la grande communauté Arduino assure que le produit est compatible avec un maximum de capteurs.

Le circuit comporte toutes les traces des capteurs, malgré le fait que nous n'utilisons pas la version en puce sur le prototype. Il est possible d'obtenir les composantes en ligne et de les souder soi-même à l'aide de la liste de matériel prévue à cet effet au chapitre portant ce nom.

Les deux dernières et majeures parties du circuit sont le régulateur de tension 3.3 volts et les transistors servant à convertir les niveaux de tensions. Les capteurs fonctionnent à un niveau de tension de 3.3 volts et le ATMEGA328P-AU fonctionne à un niveau de tension de 5 volts, d'où la nécessité des convertisseurs de niveau.

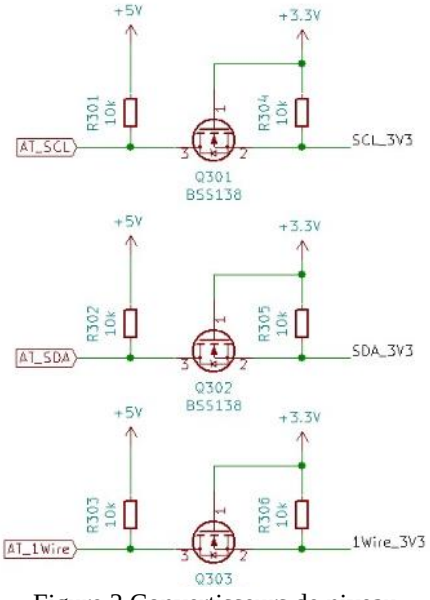


Figure 3 Convertisseurs de niveau

Certains connecteurs et traces resteront inutilisés puisque certaines parties du projet ont dû être omises par manque de temps. C'est le cas pour la partie Bluetooth, mais elle sera tout de même brièvement abordée. Comme indiqué dans le schéma de fonctionnement, le bloc de capteurs peut envoyer des données par Bluetooth, lorsque le module y est connecté. Le code n'est pas prévu à cet effet encore, mais il peut être ajouté par un adepte de programmation.

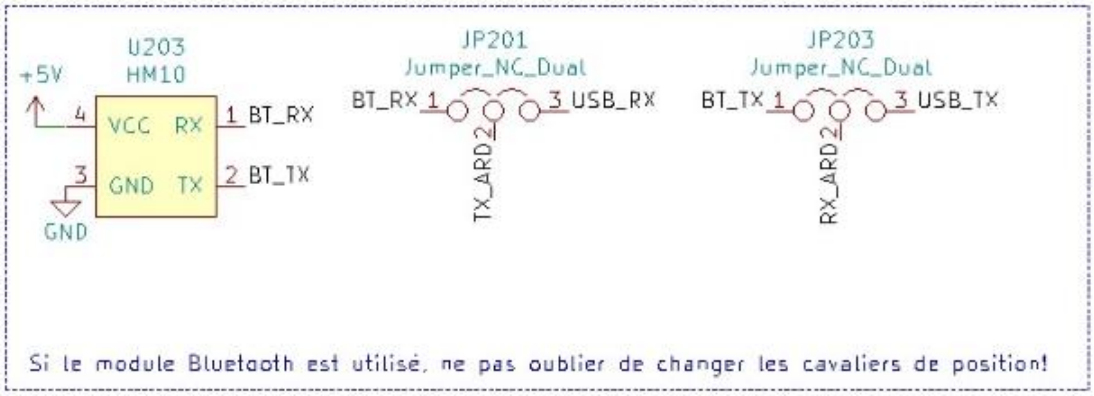


Figure 4 Cavaliers de communication

Avant de pouvoir transmettre des données, l'usager devra cependant déplacer le les cavaliers JP201 et JP203 en position 1-2, sous quoi le câble USB ne devient que la source d'alimentation du bloc de capteurs. En position 2-3, l'échange de données est effectué par l'entremise du câble USB.

3.2.1 Capteurs

3.2.1.1 DS18B20

Le capteur de température DS18B20 est abordable et simple d'utilisation, le rendant un choix évident lorsque vient le temps de prendre des mesures de température. Plusieurs libraires Arduino sont disponibles pour effectuer la lecture de ce capteur 1-Wire. Il est disponible en format TO-92 et en un format plus étanche comportant un embout en métal pour mieux conduire la chaleur ambiante au capteur. Malgré sa précision moindre (±0.5 °C), il reste un très bon capteur pour avoir une lecture valide de la température.

Le circuit imprimé du bloc de capteurs porte une empreinte pour placer directement un DS18B20 en format TO-92 ainsi qu'un connecteur pour y brancher la version plus robuste. Cette empreinte additionnelle permet d'ailleurs d'avoir deux DS18B20 et ainsi assurer une certaine redondance.

3.2.1.2 SHT20

Le capteur SHT20 est un capteur thermique et hygrométrique fonctionnant par protocole I2C ayant une précision de ±3%, ce qui est relativement précis en termes de capteur d'humidité.

3.2.1.3 CCS811

Le capteur de gaz CCS811 est un capteur peu dispendieux (sous 10 dollars) qui lit la concentration en dioxyde de carbone ambiante ayant une portée de 400 à 16383 parties par million. Cette mesure est de plus effectuée par approximation et est donc plus ou moins précise. À défaut d'avoir plus de fonds, le capteur CCS811 devra faire l'affaire en termes de capteur de concentration en CO2. Sa portée de lecture limitée en fait un choix boiteux, mais les capteurs de CO2 de qualité nécessitaient davantage de fonds, chose que mon équipe n'avait pas.
Tout comme les autres capteurs du circuit de capteurs, le circuit imprimé comporte une trace faite pour la puce du capteur ainsi que ses composantes secondaires nécessitées pour son fonctionnement, de manière qu'il soit possible d'avoir deux capteurs CCS811. Le cavalier de soudure (JP301) doit être sectionné pour changer l'adresse du capteur de gaz s'il advenait que deux soit connectés à la fois, ou si un autre capteur à la même adresse.

Un bon remplacement serait le SCD30 d'Adafruit, qui effectue une vraie lecture de la concentration de dioxyde de carbone environnante, avec un maximum détectable de 65535 parties par millions.

3.3 Boitier de contrôle:

Le boitier utilisé pour le projet est un boitier Hammond Manufacturing EJ12108. Le boitier idéal aurait eu des dimensions de 14 pouces longueur par 10 pouces de largeur par 8 pouces de profondeur. Le boitier est un du type industriel, mais il a été modifié pour qu'il puisse répondre à nos besoins.

Un ventilateur y a été apposé de manière à forcer l'air chaud à s'évacuer. De plus, pour s'assurer que le niveau d'humidité introduit dans le boitier ne soit dommageable pour le projet, un capteur d'humidité SHT20 peut être soudé à même le circuit, sinon un connecteur permet d'y brancher la version câblée.

3.3.1 Raspberry Pi et affichage:

Le projet utilise un Raspberry Pi comme microcontrôleur auquel l'écran tactile de 7 pouces est connecté par HDMI et un fil micro-USB. L'écran tactile est attaché au panneau avant de la boite de contrôle grâce à un contour d'écran imprimé en 3D. Deux autres pièces, des supports de plastiques, ont aussi été créées à l'aide des imprimantes 3D du Cégep et même celle d'un de nos collègues.

L'écran était en fait dans un boitier de plastique peu commode pour notre projet, nous avons donc décidé de retirer l'écran et son circuit du boitier, nous laissant donc d'autres choix que de faire notre propre contour pour l'écran.
Les deux autres supports en question sont ceux qui fixent le Raspberry Pi et le circuit de contrôle à la boite de contrôle.

3.3.2 Le bloc d'alimentation:

Initialement, le projet était supposé avoir son propre circuit pour générer les tensions nécessitées à partir de la tension du secteur, comme le 12 volts et le 5 volts CC pour le circuit de contrôle et le Raspberry Pi, par exemple, mais afin de pouvoir obtenir un prototype fonctionnel plus rapide, nous avons opté pour un bloc d'alimentation d'ordinateur de 450W.

La puissance du bloc n'était pas vraiment un choix, nous avons plutôt pris ce que nous avions a porté et parce que c'était gratuit.

De ce bloc d'alimentation, le projet utilise la tension de 12 volts CC et celle de 5 volts CC. La tension de 12 volts est injectée dans le circuit de contrôle pour activer les divers ventilateurs ainsi que comme tension d'entrée pour la source de courant.

3.3.3 Ventilateur du boitier:

La ventilation du boitier est effectuée par un ventilateur de 120 millimètres d'ordinateur, fonctionnant avec une tension de 12 volts et consommant moins d'un demi- ampère. Elle envoie de l'air directement sur les 3 composantes cibles, c'est-à-dire le circuit de contrôle, le Raspberry Pi et le dissipateur de chaleur de la source de courant constante. L'air chaud s'évacue ensuite vers le haut de la boite de contrôle. Le flot de l'air ne suit pas l'échappement naturel de l'air chaud, mais c'était la seule direction possible vu toutes les contraintes de positionnement des autres composantes.

3.3.4 Prises de courant de secteur:

Comme le projet devait être capable d'accepter les appareils des usagers préexistants, des prises de courant de secteur ont été choisies comme connecteurs. Ces prises servent donc à connecter les éléments de contrôle.

Les différentes prises sont identifiées pour que l'usager puisse bien connecter les éléments aux bonnes prises pour éviter les problèmes. Ces prises sont situées sur le côté gauche du boitier de contrôle.

3.3.4.1 ATTENTION :

Sur le prototype actuel, la prise humidificateur est aussi une prise de courant régulière, mais devra être changée à tout prix! Elle laisse croire que l'on peut y brancher un appareil de 120 volts CA, alors que cette sortie donne une tension de 36 volts DC!
Elle a été utilisée à défaut d'avoir un connecteur différent, car ce problème n'avait pas été vu sous cet angle et donc aucun connecteur n'a été prévu à cet effet.

3.4 Éléments de contrôle

Les éléments de contrôle sont à la discrétion de l'usager. Il peut choisir de n'utiliser aucun des éléments de contrôle et de ne faire que du monitorage avec le produit. Comme indiqué précédemment, les appareils utilisés peuvent aller jusqu'à 360 watts, mais le produit n'a pas été testé à cet effet encore.

4 Programmes

4.1 Programmes principaux

Pour que le projet fonctionne, trois programmes fonctionnent en même temps. Ces programmes importants sont :

"Projet_EC_GUI.py"
"Projet_EC_Logic.py"
"Projet_EC_LectureData.ino"

Le premier, "Projet_EC_GUI.py", se charge de l'affichage de l'interface usager du projet tandis que le deuxième, "Projet_EC_Logic.py", agit comme intermédiaire, donc communique avec le programme d'acquisition de données des capteurs que le microcontrôleur du bloc de capteurs exécute "Projet_EC_LectureData.ino" et les envois au premier programme, celui qui s'occupe du *GUI* ou « *Graphical User Interface* » (interface usager).

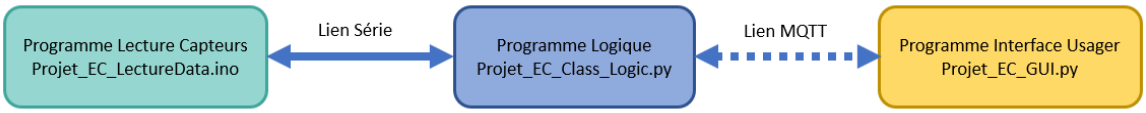


Schéma 3 Lien entre les programmes

Comme le démontre le graphique ci-dessus, le programme python de logique, "Projet_EC_Logic.py", lie les trois programmes. Du programme Arduino, il reçoit une trame de données générées qu'il doit décoder et analyser pour vérifier sa validité. Une fois terminé, il transmet les valeurs décodées au programme d'affichage du GUI par MQTT. Le programme d'interface usager peut aussi envoyer des informations à programme de logique par différent topic MQTT au besoin.

Pour comprendre le fonctionnement du projet et de ses programmes, les sections ci-dessous sont dédiées à l'explication de ces trois codes. Pour aider à la compréhension du fonctionnement de chaque programme, le schéma des classes suivant démontre les liens entre ces programmes, leurs classes et leurs fichiers respectifs.

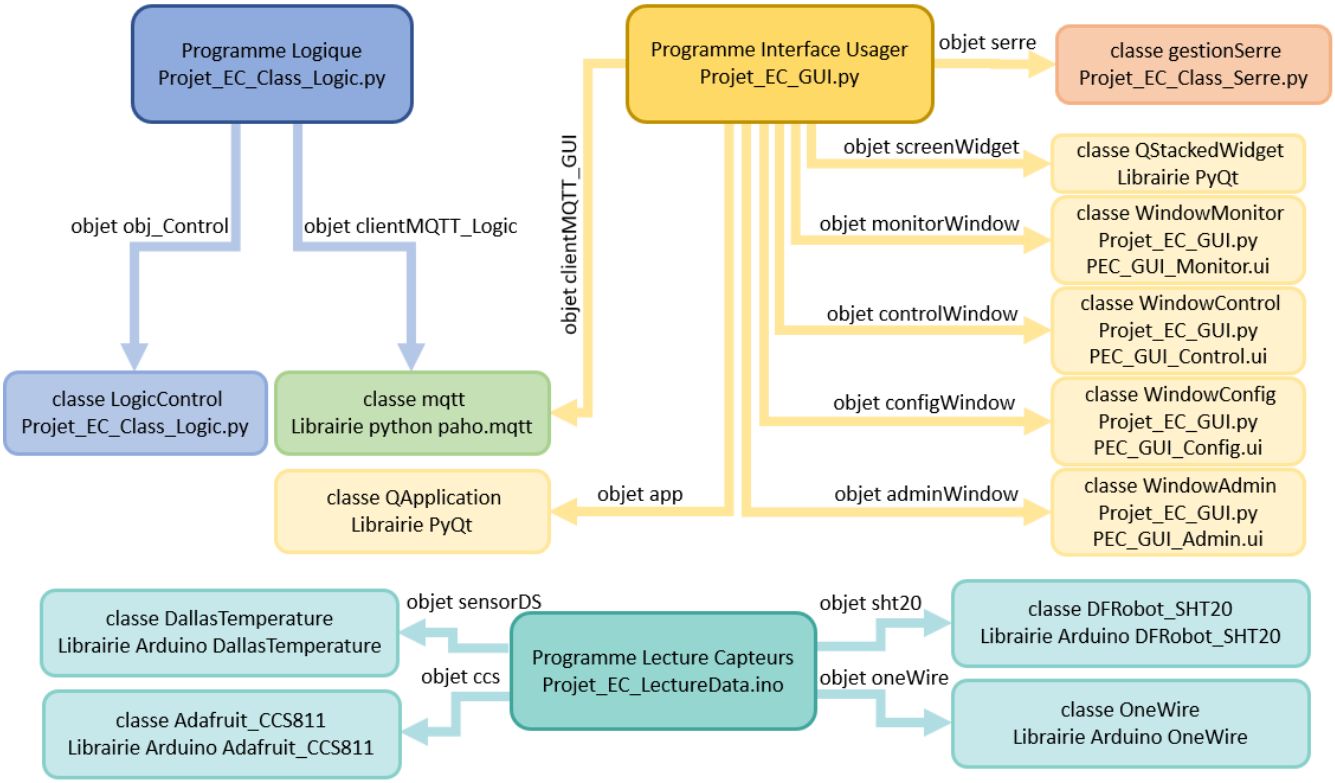


Schéma 4 Schéma des classes

4.2 Projet_EC_GUI

Le programme d'interface usager ou *GUI* en abrégé est en fait bien plus qu'un seul fichier. Le fichier principal est le "Projet_EC_GUI.py" et il utilise 5 autres fichiers, soit "Projet_EC_Class_Serre.py", "GUI_PEC_Monitor.ui", "GUI_PEC_Control.ui", "GUI_PEC_Config.ui" et "GUI_PEC_Admin.ui". Afin de comprendre le fonctionnement du programme, il faut d'abord savoir que sont les fichiers ".ui".

4.2.1 Qt Designer 5

Le programme python utilise des fichiers ".ui" pour générer les différentes fenêtres du *GUI*. Ces fichiers ont été générés par le programme Qt Designer 5, qui est un programme de design d'interface usager sous forme graphique.
Le programme est simple et permet de générer des fenêtres en glissant et déposant les éléments désirés. En sauvegardant un fichier dans Qt Designer, le fichier créé est aussitôt enregistré avec l'extension ".ui". Les fichiers ".ui" qu'il génère sont en fait du code XML modifié, qui dicte l'emplacement et le placement des différents éléments sur la fenêtre comme les boutons, des étiquettes, etc.

Ce programme ne gère toutefois pas le code lié aux éléments de la fenêtre, il ne fait que générer les éléments graphiques et nécessite donc un code externe pour leur donner vie. C'est pourquoi ils sont chargés dans le programme python du *GUI*. Cependant, afin d'implémenter du code événementiel et interagir avec le *GUI*, on doit d'abord charger les fichiers avec un module de *PyQt5* ou convertir le code XML en code Python pour accéder aux différents éléments de l'interface, leurs propriétés, événements, etc. Les deux façons pour pouvoir accéder aux propriétés des éléments générés par Qt Designer sont les suivantes.

4.2.1.1 Conversion du code XML Qt

Pour convertir le fichier ".ui" en fichier python, la commande suivante est utilisée.

```
>pyuic5 Projet_EC_Monitor.ui -o Projet_EC_Monitor.py
```

Figure 5 Exemple de conversion

L'utilitaire permettant la conversion est le module uic de *PyQt5*, d'où le "pyuic5". On fournit le fichier à convertir en premier argument, dans cet exemple le fichier "Projet_EC_Monitor.ui" sera converti, puis on spécifie le nom du fichier qui sera généré avec l'argument "-o" puis le nom du fichier. En exécutant cette ligne, tout le code XML est converti en code dans un fichier python, nommé "Projet_EC_Monitor.py" dans cet exemple-ci, contenant toutes les propriétés de tous les éléments contenus dans la fenêtre.

```python
def setupUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(1087, 663)

    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
    self.gridLayout.setObjectName("gridLayout")
    self.verticalLayout = QtWidgets.QVBoxLayout()
    self.verticalLayout.setObjectName("verticalLayout")
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setObjectName("label")
    self.verticalLayout.addWidget(self.label)
    self.lcdNumber_DSB = QtWidgets.QLCDNumber(self.centralwidget)
    self.lcdNumber_DSB.setObjectName("lcdNumber_DSB")
    self.verticalLayout.addWidget(self.lcdNumber_DSB)
    self.label_2 = QtWidgets.QLabel(self.centralwidget)
    self.label_2.setObjectName("label_2")
    self.verticalLayout.addWidget(self.label_2)
    self.gridLayout.addLayout(self.verticalLayout, 0, 0, 1, 1)
    self.verticalLayout_3 = QtWidgets.QVBoxLayout()
```

Figure 6 Exemple de code converti

Le code obtenu est toutefois difficilement lisible, car les lignes de code sont générées automatiquement et chaque élément porte des dizaines de propriétés, créant un code de presque mille lignes pour une fenêtre très simple. De plus, si des modifications sont faites, le fichier doit être régénéré et le code additionnel fait doit être copié dans le nouveau programme. Encore pire, il est possible que le fichier généré écrase par erreur le fichier codé par l'usager lors d'une correction mineure, causant la perte de tout avancement.

4.2.1.2 Chargement du code XML Qt

L'autre option est de charger le fichier ".ui" dans le code et elle permet de ne pas avoir des centaines de lignes de code d'initialisation d'éléments graphique ainsi que de faire des ajustements rapides puisqu'on a seulement à modifier le fichier ".ui" et donc le code lié à l'interface n'a pas besoin d'être modifié ou copié dans le nouveau fichier généré.

```
class WindowMonitor(QMainWindow):


    def __init__(self):
        super(WindowMonitor,self).__init__()
        loadUi("GUI_PEC_Monitor.ui",self)
        self.btn_Monitor_Menu.setStyleSheet("background-color:#c5c5c6")
        self.btn_Outils_Menu.clicked.connect(goto_Outils)
        self.btn_Setpoints_Menu.clicked.connect(goto_Setpoints)
        self.btn_Param_Menu.clicked.connect(goto_Param)
        self.lcd_Temp.display("0.00")
        self.lcd_Humid.display("0.00")
        self.lcd_Dioxide.display("0.00")
        self.lcd_TVOC.display("0.00")
```

Figure 7 Exemple d'utilisation du loadUi[1] et d'accès aux propriétés des éléments

Cet exemple-ci est tiré directement du programme "Projet_EC_GUI.py" et montre que les éléments de la fenêtre de monitorage du GUI sont chargés et que les différents objets qui y sont contenus sont maintenant accessibles, comme le montre la figure suivante.

[1] Pour utiliser la fonction loadUi de cette façon, la fonction doit être importée directement, tel que :
From PyQt5.uic import loadUi
Sinon la fonction loadUi devra être appelé en précisant uic avant loadUi (exemple : uic.loadUi("ex.ui",self))

En voyant la différence entre la figure 2 et la figure 4, il est relativement clair pourquoi l'option du loadUi a été privilégiée, car la figure 2 n'est qu'une infime partie de l'initialisation de la fenêtre, alors que la figure 4 l'est en entièreté.

4.2.2 Objets et classes

La figure 4 démontre l'ordre d'instanciation des divers objets du programme gérant le GUI et celle qui doit être respectée. Comme dans le schéma 4, on voit que le programme instancie plusieurs objets. Le "main" du programme a été séparée en deux parties, la première étant dédié à l'instanciation de l'objet serre et du client MQTT ainsi que de son initialisation, puis les objets relatifs à l'affichage du GUI lui-même.
Comme indiqué dans la section 4.2.1.2, la génération des fenêtres de l'interface usager est beaucoup plus simple lorsque les fichiers d'instructions ".ui" ne sont pas convertis, mais seulement chargés avec la fonction loadUi. C'est donc ce que fait chacune des classes débutant par "Window". Leur contenu de base est donc très similaire d'une classe à l'autre, mais elles chargent toutes un fichier différent. La figure 3 est un bon exemple du contenu de ces classes.

```
if __name__ == "__main__":   #Debut du "Main"
    #Init
    serre = gestionSerre() #Instanciation d'un objet de la classe gestionSerre Elle serre a storer les données de la serre
    clientMQTT_GUI = mqtt.Client("clientMQTT_GUI") #client MQTT lié au échanges entre les données de la serre et l'affichage
    clientMQTT_GUI.connect(serre.BROKER_MQTT_ADDR)
    clientMQTT_GUI.subscribe([(serre.TOPIC_SERRE,0),(serre.TOPIC_GUI_FLAGS,0)])
    clientMQTT_GUI.on_message = on_message
    clientMQTT_GUI.loop_start()

    #Setup GUI
    app = QApplication(sys.argv)
    screenWidget = QtWidgets.QStackedWidget() #Widget qui contient toutes les fenetres et permet de switcher entre fenetres
    monitorWindow = WindowMonitor() #Instanciation d'un objet de chaque type de fenetres
    controlWindow = WindowControl()
    configWindow = WindowConfig()
    adminWindow = WindowAdmin()
    screenWidget.addWidget(monitorWindow)   #Ajoute les fenetres au Gestionnaire screenWidget
    screenWidget.addWidget(controlWindow)
    screenWidget.addWidget(configWindow)
    screenWidget.addWidget(adminWindow)

    #screenWidget.showFullScreen()         #Affiche en plein ecran ( PROJET MODE )
    screenWidget.showMaximized()         #Affiche le GUI en maximizé ( Bloquant )
    GPIO.cleanup()
    try:
        sys.exit(app.exec_())
    except:
        print("Exiting")
```

Figure 8 "Main" du programme de *GUI*

Une fois instanciés, ils sont ajoutés à l'objet qui gère les fenêtres, "screenWidget". Cet objet permet d'avoir un seul programme python qui gère l'affichage de

plusieurs fenêtres Qt à la fois, sans avoir recours à des signaux et simplifie davantage le code. Les fenêtres (en fait objets) qui y sont ajoutées sont enregistrées par index, de zéro à quatre. On peut d'ailleurs voir les fonctions utilisées pour changer l'index de l'objet de gestion d'affichage pour changer de fenêtre dans la figure 5.

```
WindowMonitor_Index = 0 #defines d'index pour les fenetres
WindowControl_Index = 1
WindowParam_Index = 2
WindowAdmin_Index = 3

def goto_Monitor(): #fonctions pour changer de fenêtres
    screenWidget.setCurrentIndex(WindowMonitor_Index)
def goto_Setpoints():
    controlWindow.updateSetpointsLCD()
    screenWidget.setCurrentIndex(WindowControl_Index)
def goto_Param():
    screenWidget.setCurrentIndex(WindowParam_Index)
def goto_Outils():
    screenWidget.setCurrentIndex(WindowAdmin_Index)


class WindowMonitor(QMainWindow):

    def __init__(self):
        super(WindowMonitor,self).__init__()
        loadUi("GUI_PEC_Monitor.ui",self)
        self.btn_Monitor_Menu.setStyleSheet("background-color:#c5c5c6")
        self.btn_Outils_Menu.clicked.connect(goto_Outils)
        self.btn_Setpoints_Menu.clicked.connect(goto_Setpoints)
        self.btn_Param_Menu.clicked.connect(goto_Param)
```

Figure 9 Fonctions d'affichage

Comme le montre la figure à gauche, on attache la fonction de changement de fenêtre comme fonction d'évènement lié aux boutons du menu. L'exemple montre la classe "WindowMonitor", mais les quatre classes d'affichage comporte ces fonctions, de manière à pouvoir naviguer le menu.

4.2.3 Classes d'affichage

Ce que l'on retrouve dans la fonction d'initialisation de ces classes est l'attachement des évènements des boutons de navigation du menu aux fonctions permettant l'affichage des différentes pages, ainsi que d'autres attachements d'évènements propres à chaque page.

Pour chacune de ces classes, une image de la fenêtre générée sera fournie en plus d'une courte description des évènements programmés.

4.2.3.1 Classe WindowMonitor

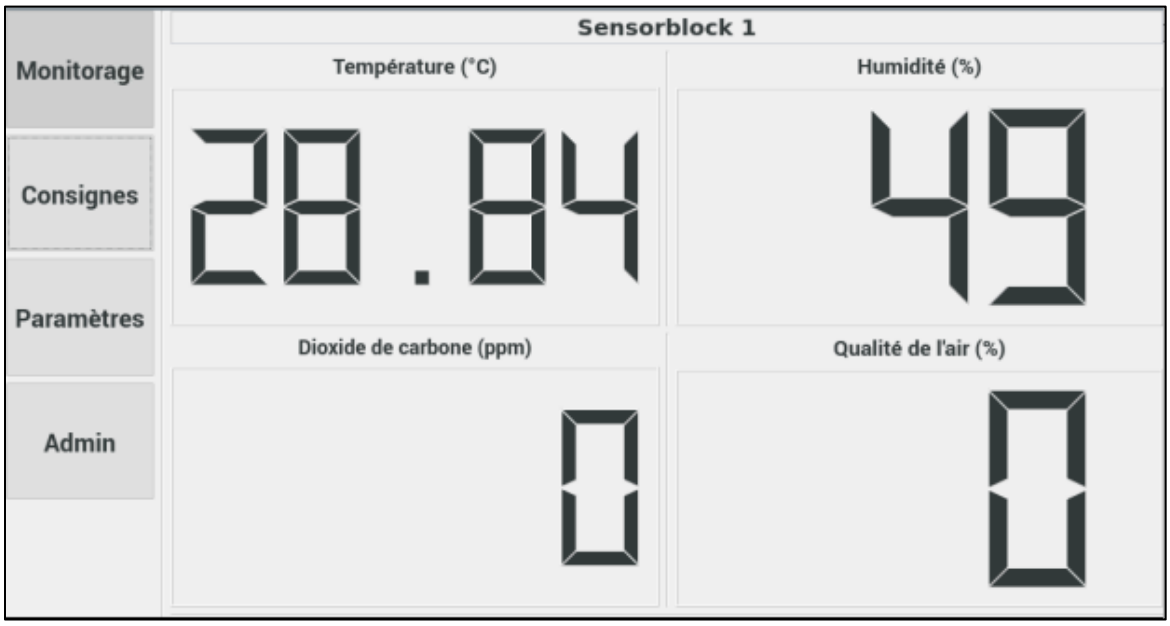


Figure 10 Page "Monitorage" de l'interface usager

Sur cette fenêtre, seulement les boutons du menu ont des évènements liés et c'est le changement de fenêtres, comme indiqué précédemment. Les valeurs affichées sont modifiées en temps réel aussitôt qu'elles arrivent.

4.2.3.2 Classe WindowControl

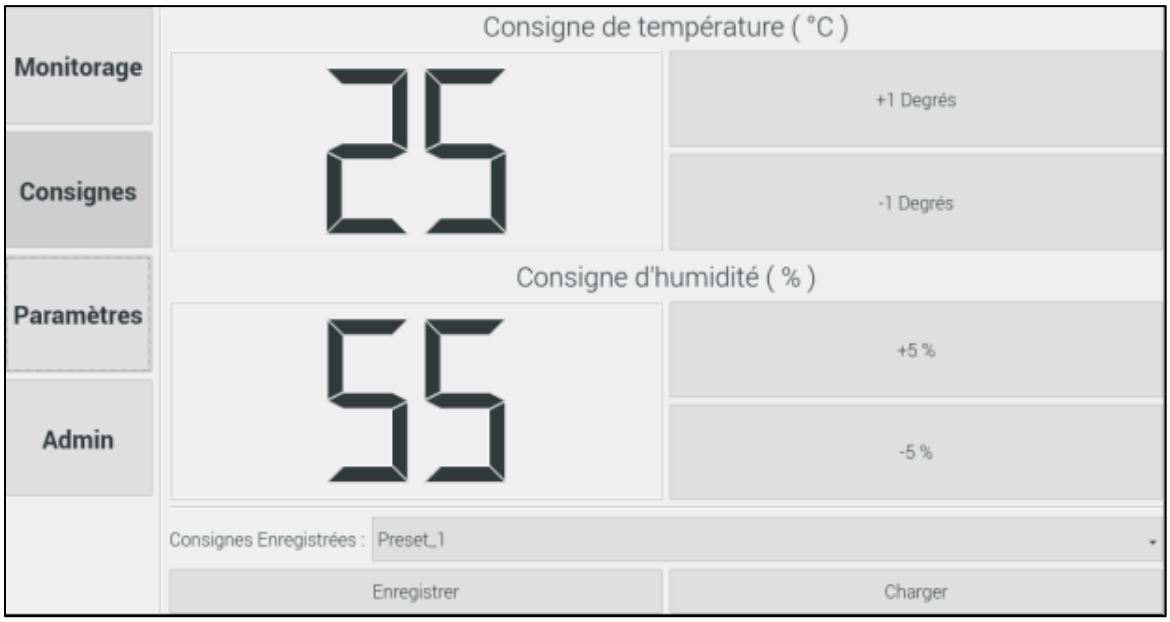


Figure 11 Page "Consignes" de l'interface usager

La fenêtre de la classe "WindowControl" est la page de contrôle des consignes de température et d'humidité. Elle permet de visualiser, changer, enregistrer et

charger des consignes. La plupart des fonctions de la classe sont liées à l'ouverture, la lecture et l'enregistrement de valeurs de consignes. Ces fonctions sont :

4.2.3.2.1 Fonction btn_Decrement_Temp_Clicked

Cette fonction est appelée lors d'un clic sur le bouton pour décrémenter la consigne de température. Elle diminue la consigne de température enregistrée dans l'objet serre si elle est au-dessus de 0 degré, sinon le bouton ne fait rien. La consigne est ensuite envoyée par MQTT sur le topic "/Projet_EC/GUI" au programme de logique, puis fait une mise à jour de l'affichage des consignes. Pour finir, la nouvelle consigne est sauvegardée dans un fichier en appelant la fonction "saveSetpoints" de l'objet serre.

4.2.3.2.2 Fonction btn_Increment_Temp_Clicked

Cette fonction est appelée lors d'un clic sur le bouton pour incrémenter la consigne de température. Elle incrémente la consigne de température enregistrée dans l'objet serre si elle est sous 30 degrés, sinon le bouton ne fait rien. La consigne est ensuite envoyée par MQTT sur le topic "/Projet_EC/GUI" au programme de logique, puis fait une mise à jour de l'affichage des consignes. Pour finir, la nouvelle consigne est sauvegardée dans un fichier en appelant la fonction "saveSetpoints" de l'objet serre.

4.2.3.2.3 Fonction btn_Decrement_Humid_Clicked

Cette fonction est appelée lors d'un clic sur le bouton pour décrémenter la consigne d'humidité. Elle diminue la consigne d'humidité enregistrée dans l'objet serre si elle est au-dessus de 0%. Sinon le bouton ne fait rien. La consigne est ensuite envoyée par MQTT sur le topic "/Projet_EC/GUI" au programme de logique, puis fait une mise à jour de l'affichage des consignes. Pour finir, la nouvelle consigne est sauvegardée dans un fichier en appelant la fonction "saveSetpoints" de l'objet serre.

4.2.3.2.4 Fonction btn_Increment_ Humid _Clicked

Cette fonction est appelée lors d'un clic sur le bouton pour incrémenter la consigne d'humidité. Elle incrémente la consigne d'humidité enregistrée dans l'objet serre si elle est sous 95%, sinon le bouton ne fait rien. La consigne est ensuite envoyée par MQTT sur le topic "/Projet_EC/GUI" au programme de logique, puis fait une mise à jour de l'affichage des consignes. Pour finir, la nouvelle consigne est sauvegardée dans un fichier en appelant la fonction "saveSetpoints" de l'objet serre.

4.2.3.2.5 Fonction btn_Save_Set_Clicked

Cette fonction d'événements permet de sauvegarder les consignes entrées par l'usager. La consigne enregistrée sélectionnée dans la boite-liste des consignes dans le bas de la fenêtre sera alors remplacée par celle qui a été entrée par l'usager. De cette façon, l'usager peut entrer des consignes personnalisées. La fonction ouvre le fichier dans lequel sont sauvegardées les consignes, modifie la ligne choisie par l'usager puis l'enregistre.

4.2.3.2.6 Fonction btn_Load_Saved_Set_Clicked

Très similaire à la fonction précédente, excepté qu'à la place de sauvegarder, elle charge la consigne enregistrée et l'affiche. Les valeurs de consignes sont envoyées par MQTT sur le topic "/Projet_EC/GUI" au programme de logique.

4.2.3.2.7 Fonction getPresets

Cette fonction est appelée lors de l'initialisation de l'objet de la classe gérant cette fenêtre. Elle lit la liste des consignes et peuple la boite-liste de la fenêtre "Consignes"

4.2.3.2.8 Fonction updateSetpointsLCD

Cette fonction est très simple, elle ne fait qu'actualiser l'affichage des deux valeurs de consignes sur la page avec les valeurs qui sont contenues dans les variables "setpoint_Temp" et "setpoint_Humid" de l'objet "serre".

4.2.3.3 Classe WindowConfig

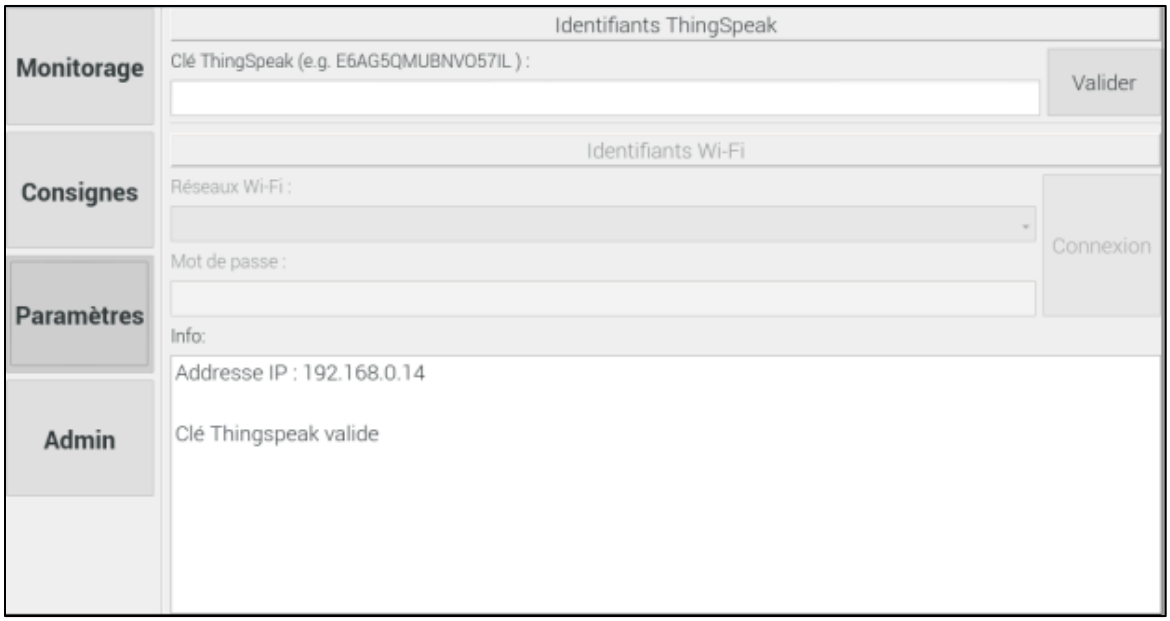


Figure 12 Page "Paramètres" de l'interface usager

Cette fenêtre est séparée en 3 parties. La première est dédiée à la connexion à ThingSpeak, pour l'enregistrement des données en ligne, la section du milieu permettrait normalement de se connecter aux réseaux wifi, mais puisque le temps manquait, cette partie a été omise. Quant a la dernière partie, elle affiche des données utiles pour débugger et pour configurer le projet.

Note : La configuration du compte ThingSpeak dans le programme nécessite forcement au moins une connexion à distance ou de brancher un clavier sur le Raspberry Pi, car l'usager doit entrer sa clé d'API personnelle. Cependant, le programme pourrait très bien fonctionner sans connexion à ThingSpeak.

Cette classe-ci ne comporte qu'une seule fonction et elle est liée au bouton valider dans la section de connexion à ThingSpeak.

4.2.3.3.1 btn_Thingspeak_Cle_Clicked

En appuyant sur "Valider" lors de la configuration du ThingSpeak, ce programme envoi la clé par MQTT sur le topic "/Projet_EC/GUI" au programme de contrôle "Projet_EC_Logic.py" pour qu'il puisse tester la connexion puis ce dernier envoi une confirmation sur le topic "/Projet_EC/GUI/Flags/API" pour confirmer si

la connexion a bel et bien fonctionné, si oui la clé API est sauvegardée dans un fichier nommé "Projet_EC_API_Thingspeak.txt".

4.2.3.4 Classe WindowAdmin

La dernière fenêtre permet de tester chacune des sorties contrôlées par le programme. Encore une fois dût au manque de temps, nous n'avons pas réussi à tout implémenter les fonctionnalités voulues et certaines sont désactivées (en grisé) sur cette page de l'interface utilisateur.

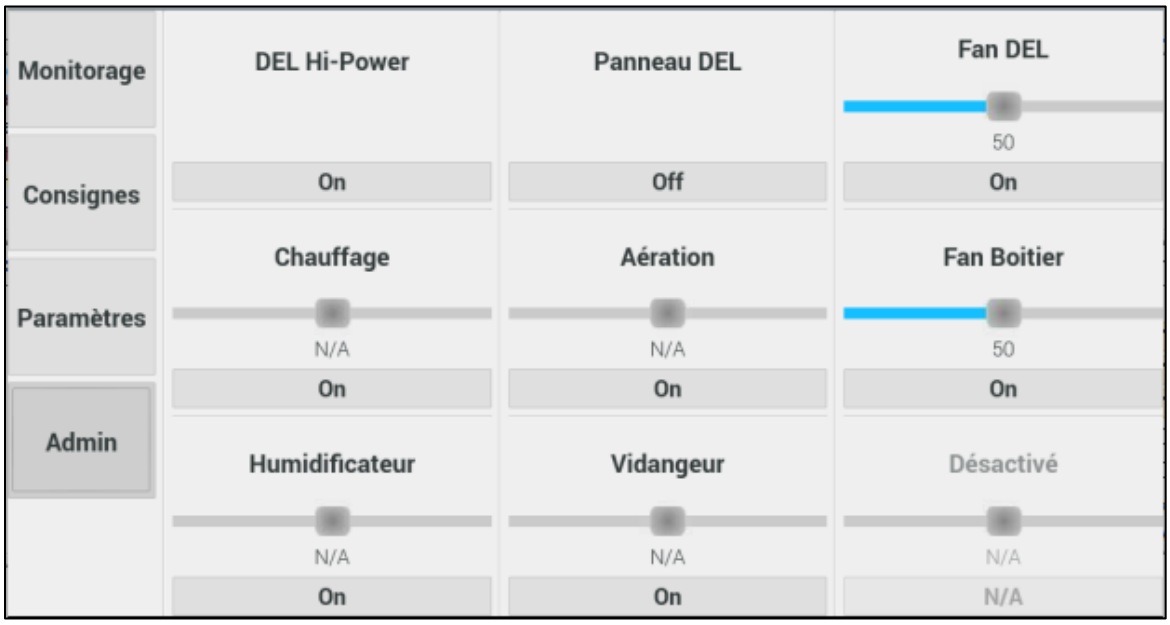


Figure 13 Page "Admin" de l'interface usager

Les fonctions de cette classe ne seront pas définies, car elles sont toutes la même chose. Encore une fois, les fonctions sont des fonctions d'événement sur clic ou changement d'intensité pour les deux ventilateurs ("*Fan*" DEL et "*Fan*" Boitier). Décrire chacune de ces fonctions serait répétitif et ne ferait qu'alourdir ce document.

La plupart des fonctions d'événement clic des boutons ne contient qu'un appel a la fonction "toggle" de l'objet "serre", lui envoyant seulement le numéro du GPIO de l'élément à activer ou désactiver, défini par une macro dans l'objet "serre" aussi.

Les fonctions d'évènements des glissières des ventilateurs envoi leur valeur (entre 0 et 100) par MQTT sur le topic "/Projet_EC/GUI" et les fonctions d'évènements des boutons de ces deux ventilateurs font de même, mais envoient soit 1 ou 0.

4.2.4 Classe gestionSerre

Le premier objet instancié par le programme est l'objet "serre" de la classe "gestionSerre". Cette classe est définie dans le fichier "Projet_EC_Class_Serre.py". Comme le commentaire le montre dans la figure 5, elle sert à enregistrer les données de la serre, donc les consignes, les mesures affichées sur le GUI, des constantes et l'état des sorties. C'est aussi cette classe qui s'occupe de faire l'enregistrement des consignes et la lecture des consignes dans un fichier texte externe. Les fonctions de cette classe sont les suivantes.

4.2.4.1 Fonction setup_GPIO

La fonction "setup_GPIO" est une fonction d'initialisation des GPIO du Raspberry Pi et n'est appelée qu'une seule fois. Elle change la direction des GPIO relatifs au projet en sortie.

4.2.4.2 Fonction toggle

La fonction "toggle" est la plus utilisée de cette classe. Elle permet d'activer ou de désactiver une des GPIO qui contrôle les éléments branchés au projet. Cette fonction et la fonction "setup_GPIO" utilisent toutes deux la libraire RPi.GPIO.

4.2.4.3 Fonction initSetpointList

Cette fonction permet de vérifier si une liste de consignes sous le nom de "Projet_EC_Setpoint_List.txt" existe déjà. Si oui, elle est lue et enregistrée dans la liste de consignes enregistrées de l'objet serre.
Sinon, une liste de consignes est créée, enregistrant dix lignes contenant "Preset_X 0 0", où X représente le chiffre de 1 à 10.

4.2.4.4 Fonction getSetpoints

Cette fonction-ci permet de retrouver les dernières consignes utilisées en tentant de lire le fichier "Projet_EC_Current_Setpoint.txt". Si aucune consigne n'est trouvée, donc le fichier n'existe pas, cette fonction appelle la dernière de cette classe, "saveSetpoints".

4.2.4.5 Fonction saveSetpoints

Cette fonction sauvegarde les consignes actuelles. Si aucune consigne n'a été entrée ou trouvée au démarrage du programme, les consignes sauvegardées sont de 0 et 0, ce qui veut dire que les consignes sont désactivées.

4.2.4.6 Fonction getApi

Cette fonction est appelée lors de l'initialisation du programme. Elle lit le fichier "Projet_EC_API_Thingspeak.txt" pour y extraire la clé ThingSpeak entrée précédemment. Si aucun fichier n'existe, il est créé.

4.2.5 Client MQTT

Pour que le programme de GUI puisse recevoir et afficher les données en temps réel, un client MQTT écoute constamment le topic sur lequel y sont publiées les données captées de la serre "/Projet_EC/Sensorblock_1/Readings". Ce client écoute aussi le topic "/Projet_EC/GUI/Flags/API" pour recevoir la confirmation de connexion à ThingSpeak.

C'est ce client qui assure la communication entre le programme de logique et le programme de *GUI.*

4.3 Projet_EC_Logic

Ce programme est démarré en arrière-plan lorsque le projet fonctionne. C'est ce programme qui assure la lecture du port série, donc la réception des données du bloc de capteurs, vérifie l'intégrité de la trame de données puis envoie les données au programme du *GUI*. De plus, il s'assure qu'aux 15 secondes, si le compte ThingSpeak est bien configuré, une sauvegarde des données soit faite par requête URL.

Le programme instancie aussi un objet MQTT de manière à pouvoir envoyer les données et d'écouter pour une demande quelconque du *GUI* sur leur topics respectifs.

4.3.1 Classe LogicControl

Dans le fichier "Projet_EC_Logic.py", la classe "LogicControl" est une classe enfant de la classe "gestionSerre" du fichier "Projet_EC_Class_Serre.py". De cette manière, la classe peut hériter des fonctions et des constantes de la classe mère et obtenir les mêmes valeurs de macros.

```
class LogicControl(gestionSerre):

    def __init__(self):
        super().__init__()

    #Defines
    API_KEY_LENGTH = 16
    #EndDefines
    WRITE_API = ""
    BASE_URL = ""

    Flag_Thingspeak_Connected = False
```

Figure 14 Héritage de la classe "gestionSerre"

4.3.1.1 setAPI_URL

Cette fonction est utilisée en tant que "*setter*" pour la variable WRITE_API et BASE_URL. Ce sont les variables qui contiennent la clé ThingSpeak de l'usager.

4.3.1.2 resetAPI_URL

Réinitialise les valeurs des variables contenant la clé ThingSpeak de l'usager.

4.3.1.3 verifTrame

Cette fonction vérifie la trame. Elle commence par vérifier le "*checksum*" de la trame de bytes reçue en paramètres ; il calcule le checksum puis le compare au checksum reçu. Il vérifie la longueur de la trame et si sa première byte est un SOH (équivalent à 1). Si c'est le cas, la fonction retourne "vrai", sinon elle retourne "faux".

4.3.1.4 read_Serial

Cette fonction vérifie si le port série contient des données et si oui elle le lit. Elle décode le "String" reçu, sépare les valeurs et les ajoute à une liste, qu'elle envoie à la fonction verifTrame. Si cette fonction retourne "vrai", c'est que la trame est valide. Elle est donc reconstituée et chaque valeur est enregistrée dans les variables membres de la classe, telles que "temp_DS", "temp_SHT", "humid_SHT", etc.

4.3.1.5 sendFlagThingspeakConnected

Cette fonction envoie un "Flag" vrai ou faux sur le topic MQTT "/Projet_EC/GUI/Flags/API" pour confirmer la connexion ThingSpeak dépendamment de la valeur reçue en paramètre, soit "True" ou "False".

4.3.1.6 publishDataToGUI

Cette fonction publie les valeurs enregistrées dans les variables membres de la classe au *GUI* par MQTT sur le topic "/Projet_EC/Sensorblock_1/Readings".

5 Procédure de développement

Pour assurer le fonctionnement du projet, on doit d'abord configurer le Raspberry Pi. Le projet peut être testé sans le bloc de contrôle. Il suffit d'avoir un bloc de capteurs de branché au Raspberry Pi ainsi qu'un écran et une connexion internet pour s'assurer de pouvoir configurer le tout.

5.1 Procédure pour la configuration des librairies pour le projet

5.1.1 Étapes de l'installation de l'image du Raspberry Pi

1) Télécharger l'image de Raspberry Pi Buster Desktop
2) "Flasher" une carte SD d'au moins 8Go avec cette image. Nous recommandons Balena Etcher pour cette étape
3) Faites la configuration initiale du Raspberry Pi. Lorsqu'il vous est demandé de redémarrer le Raspberry Pi, faites-le.

5.1.2 Activation des interfaces

1) Pour activer les interfaces nécessitées par le projet, dans un terminal, entrez la commande : sudo raspi-config
2) Dans le menu de configuration, sélectionnez l'index 5, "*Interfacing Options*"
3) Activez ''*SSH*'', ''*VNC*'', ''I2C'', ''*1-Wire*'' et ''*Serial*''. Ne pas activer le ''*login shell*'' depuis le serial. Acceptez lorsqu'on vous demandera d'activer le ''*Serial port hardware*''
4) De retour sur le bureau, cliquez sur le logo Raspberry Pi dans le coin haut gauche de l'écran, puis aller sur "*Preferences*" et "*Raspberry Pi Configuration*". Dans l'onglet "Display", appuyez sur "*Set Resolution*..." et sélectionnez "DMT mode 9 800x600 60Hz 4:3"
5) Redémarrez le Raspberry Pi pour qu'il effectue les changements.

5.1.3 Configuration au serveur de temps

Cette partie peut ne pas être nécessaire, mais est tout de même suggérée, afin d'assurer un serveur de temps fiable et toujours disponible.

1) Ouvrez le fichier /etc/systemd/timesyncd.conf
2) Modifiez les lignes ''NTP'' et ''FallbackNTP'' pour que leur valeur soit : time.apple.com
3) Entrez la commande : sudo systemctl restart systemd-timesync.service
4) Entrez la commande : sudo systemctl daemon-reload
5) Faites une mise à jour des logiciels et du système avec la commande : sudo apt-get update && sudo apt-get upgrade

Une fois le Raspberry Pi bien configuré, il faut ajouter les bonnes librairies pythons dans le Raspberry Pi pour que les codes puissent bien fonctionner.

La prochaine section couvre la marche à suivre pour l'installation de ces librairies.

5.1.4 Étapes pour l'installation des librairies Python

1) Installez Python : sudo apt-get install python3
2) Installez la librairie RPi.GPIO : sudo pip3 install RPI.GPIO
3) Installez la librairie du SHT20 : sudo pip3 install sht20
4) Installez le broker MQTT : sudo apt-get install mosquitto && sudo apt-get

install mosquitto-clients

5) Installez la librairie python pour MQTT : sudo pip3 install paho-mqtt
6) Installez les librairies PyQt5 : sudo apt-get install qt5-default pyqt5-dev pyqt5-dev-tools
7) Installez la librairie pour les PID : sudo pip3 install simple-pid

5.1.5 Étape pour l'installation des librairies Arduino

1) Installez la librairie du capteur Adafruit_CCS811:
    a. Ouvrez l'IDE Arduino
    b. Cliquez sur l'onglet "Croquis"
    c. Cliquez sur "Inclure une bibliothèque"
    d. Cliquez sur "Gérer les bibliothèques"
    e. Tapez "CCS811" dans la barre de recherche
    f. Installez la librairie "Adafruit CSS811 Library par Adafruit"

2) Installez la librairie DFRobot_SHT20:
    a. Ouvrez le lien https://github.com/DFRobot/DFRobot_SHT20 et télécharger le contenu en appuyant sur le bouton vert ou faites un "*pull*" de ce "*repo*"
    b. Déplacez le dossier DFRobot_SHT20-master dans votre dossier "C:\Program Files (x86)\Arduino\libraries"

3) Installez la librairie OneWire :
    a. Ouvrez l'IDE Arduino
    b. Cliquez sur l'onglet "Croquis"
    c. Cliquez sur "Inclure une bibliothèque"
    d. Cliquez sur "Gérer les bibliothèques"
    e. Tapez "OneWire"
    f. Installez la librairie "OneWire"

4) Installez la librairie DallasTemperature :
    a. Ouvrez l'IDE Arduino
    b. Cliquez sur l'onglet "Croquis"
    c. Cliquez sur "Inclure une bibliothèque"
    d. Cliquez sur "Gérer les bibliothèques"
    e. Tapez "DallasTemperature"
    f. Installez la librairie "DallasTemperature"

6 Liste de matériel et coûts

Digi-Key Part Number	Description	Quantité	Prix
HM410-ND	BOX STEEL GRAY 12"L X 10"W	1	$164.00
ATMEGA328P-AU-ND	IC MCU 8BIT 32KB FLASH 32TQFP	1	$3.31
XC2343CT-ND	CRYSTAL 16.0000MHZ 18PF SMD	1	$1.04
399-C0805Y104K4RACAUTOCT-N	CAP CER 0.1UF 16V X7R 0805	8	$0.68
FQP30N06L-ND	MOSFET N-CH 60V 32A TO220-3	4	$1.98
Q337-ND	PWR ENT RCPT NEMA5-15 PANEL QC	5	$1.63
CLA394-ND	SSR RELAY SPST-NO 6A 0-60V	1	$7.26
1189-1588-1-ND	CAP ALUM 470UF 20% 25V SMD	3	$0.98
CPC1966B-ND	SSR RELAY SPST-NO 3A 20-240V	4	$5.86
P16950CT-ND	RES SMD 470 OHM 5% 1/2W 0805	5	$0.61
RMCF0805FT10K0CT-ND	RES 10K OHM 1% 1/8W 0805	4	$0.14
1649-1004-1-ND	SENSOR HUMID/TEMP 3V I2C 3% SMD	1	$8.98
BSS138CT-ND	MOSFET N-CH 50V 220MA SOT23-3	3	$0.38
RMCF0805FT10K0CT-ND	RES 10K OHM 1% 1/8W 0805	10	$0.14
P16000CT-ND	RES 0.33 OHM 1% 1/3W 0805	3	$0.53
478-10504-1-ND	CAP CER 18PF 50V NP0 0805	2	$0.82
587-1284-1-ND	CAP CER 1UF 10V X7R 0805	1	$0.21
1276-2890-1-ND	CAP CER 10UF 25V X5R 0805	1	$0.31
AP2210N-3.3TRG1DICT-ND	IC REG LINEAR 3.3V 300MA SOT23-3	1	$0.61
768-1007-1-ND	IC USB FS SERIAL UART 28-SSOP	1	$6.56
Circuit bloc de capteurs	Circuit imprimé du projet	1	$4.00
Circuit bloc de contrôle	*idem*	1	$14.56
Total:			$224.59

Tableau 1 Liste de matériel

Comme vous pouvez le voir, ce tableau ne contient pas les éléments de contrôle. Bien qu'ils soient nécessaires au contrôle de la serre, il n'est pas assuré que le cout sera le même pour tous les usagers. De plus, le projet peut fonctionner sans les éléments de contrôle. Ce tableau représente alors plutôt le cout minimal pour recréer le projet.

Cependant, le tableau suivant comprend les éléments de contrôle utilisé avec le prototype, à titre informatif.

Item	Description	Source	Quantité	Prix
SunStream 24"x24"x56" Mylar Hydroponic Grow Tent	Serre tente avec intérieur en mylar	Amazon.ca	1	$69.99
600W LED Grow Light Double On/Off Switch Full Spectrum Grow Lamp	Panneau de DEL	Amazon.ca	1	$67.99
VIVOSUN 195 CFM 4" Inch Inline Ventilation Duct Fan	Ventilateur de 10 cm	Amazon.ca	2	$39.99
LOHAS® Ampoule LED 100 W Blanc froid Haute puissance	DEL sur puce	Amazon.ca	1	$14.31
Vivosun Tapis Chauffant	Élément chauffant en forme de tapis	Amazon.ca	1	$24.99
Ventilateur de refroidissement en aluminium avec dissipateur thermique	Dissipateur thermique en aluminium avec ventilateur de refroidissement pour DEL sur puce	Amazon.ca	1	$34.78
3 Head Ultrasonic Mist Maker Fogger Humidifier W/transformer Humidification	Humidificateur ultrasonique	Vevor.ca	1	$69.99
Total				$322.04

Tableau 2 Liste des éléments de contrôle du prototype

7 Annexe

7.1 Schéma

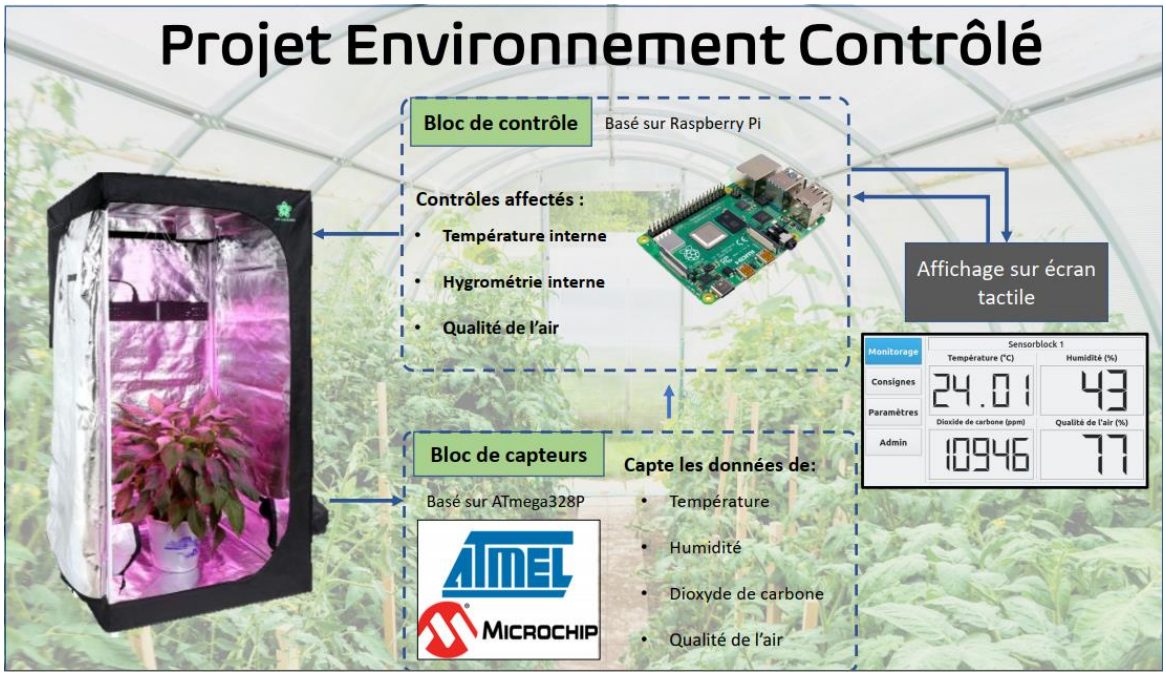


Figure 15 Schéma synoptique du produit

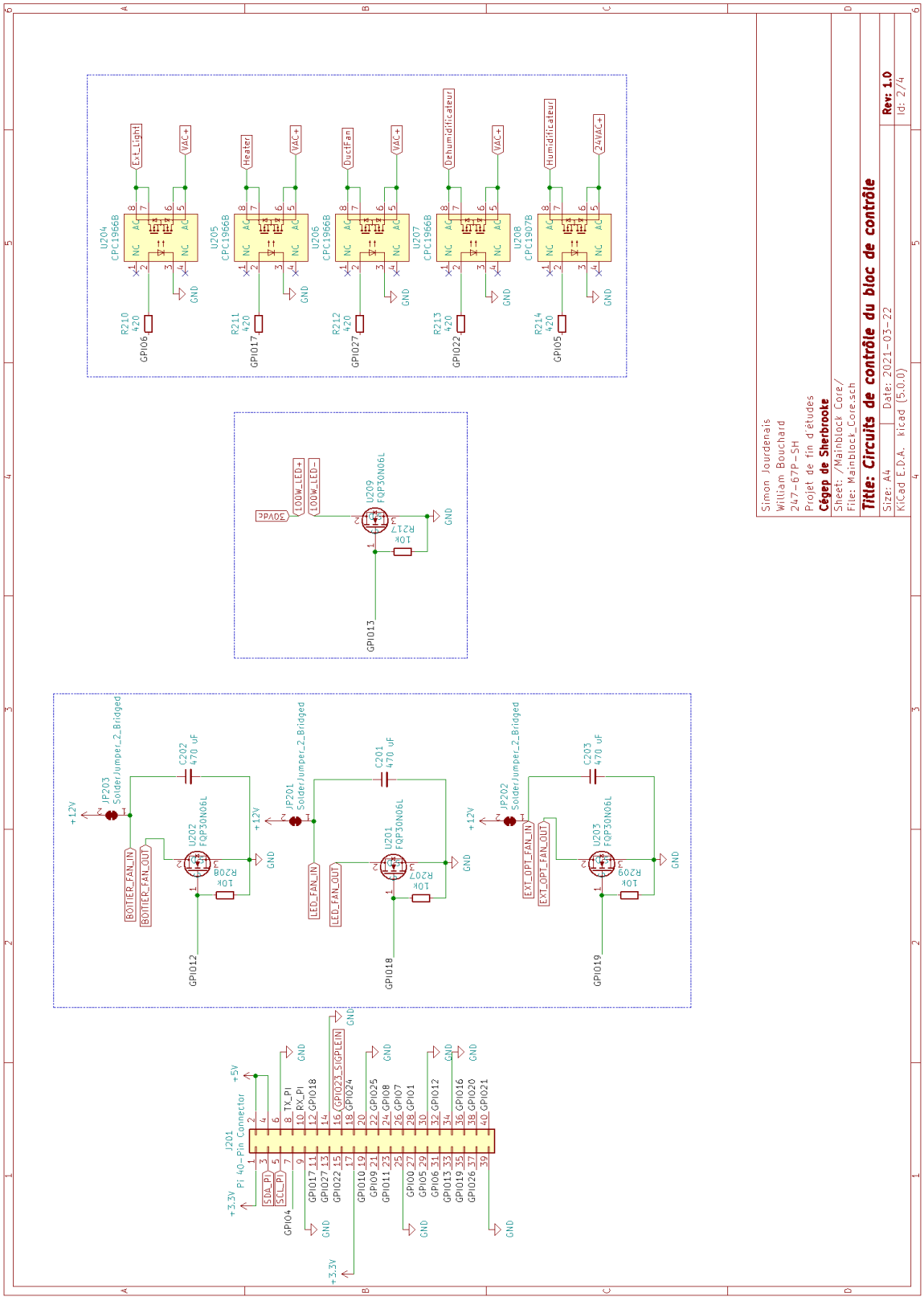


Figure 16 Schéma électrique du bloc de contrôle – Section IO

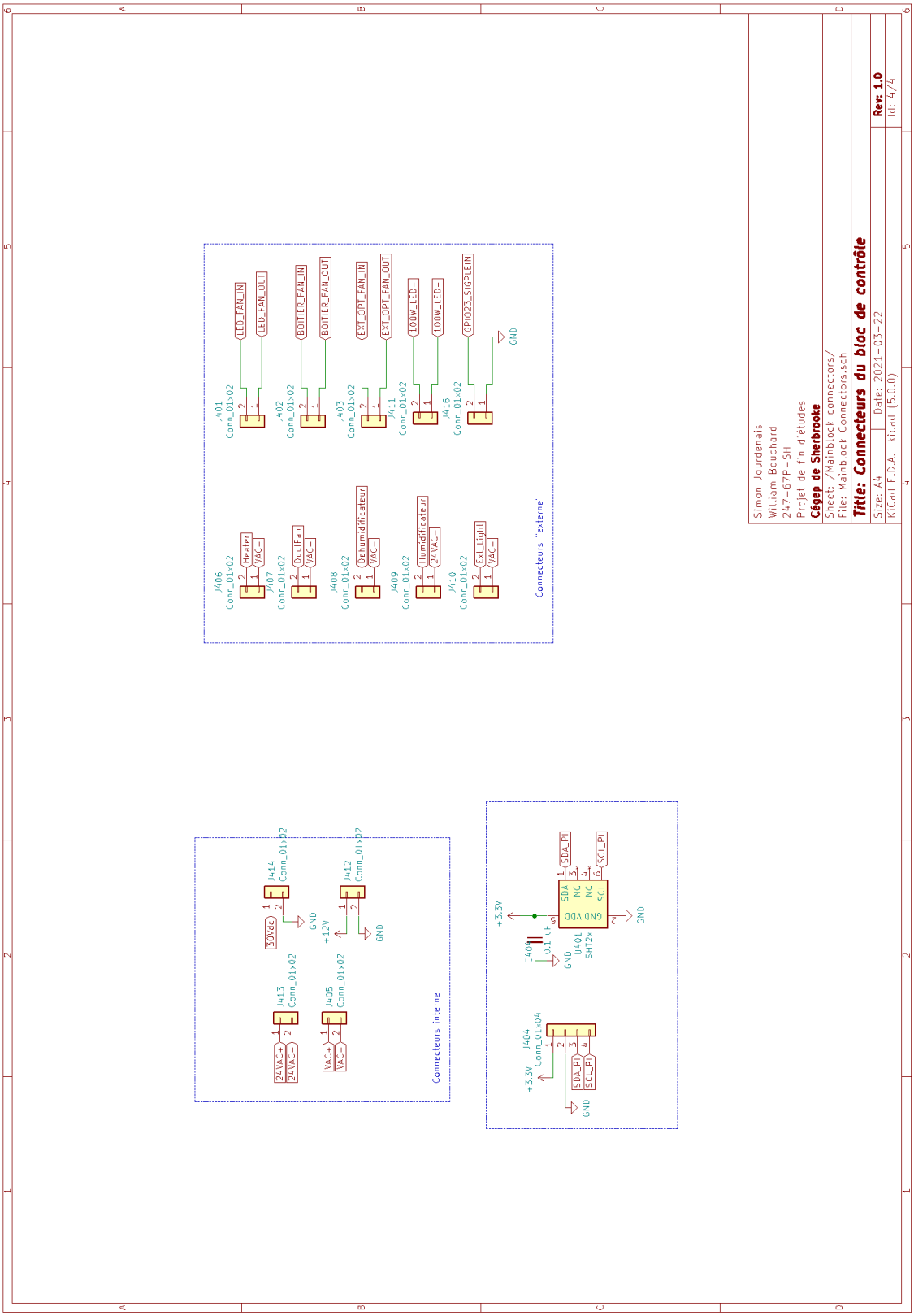


Figure 17 Schéma électrique du bloc de contrôle - Section connecteurs

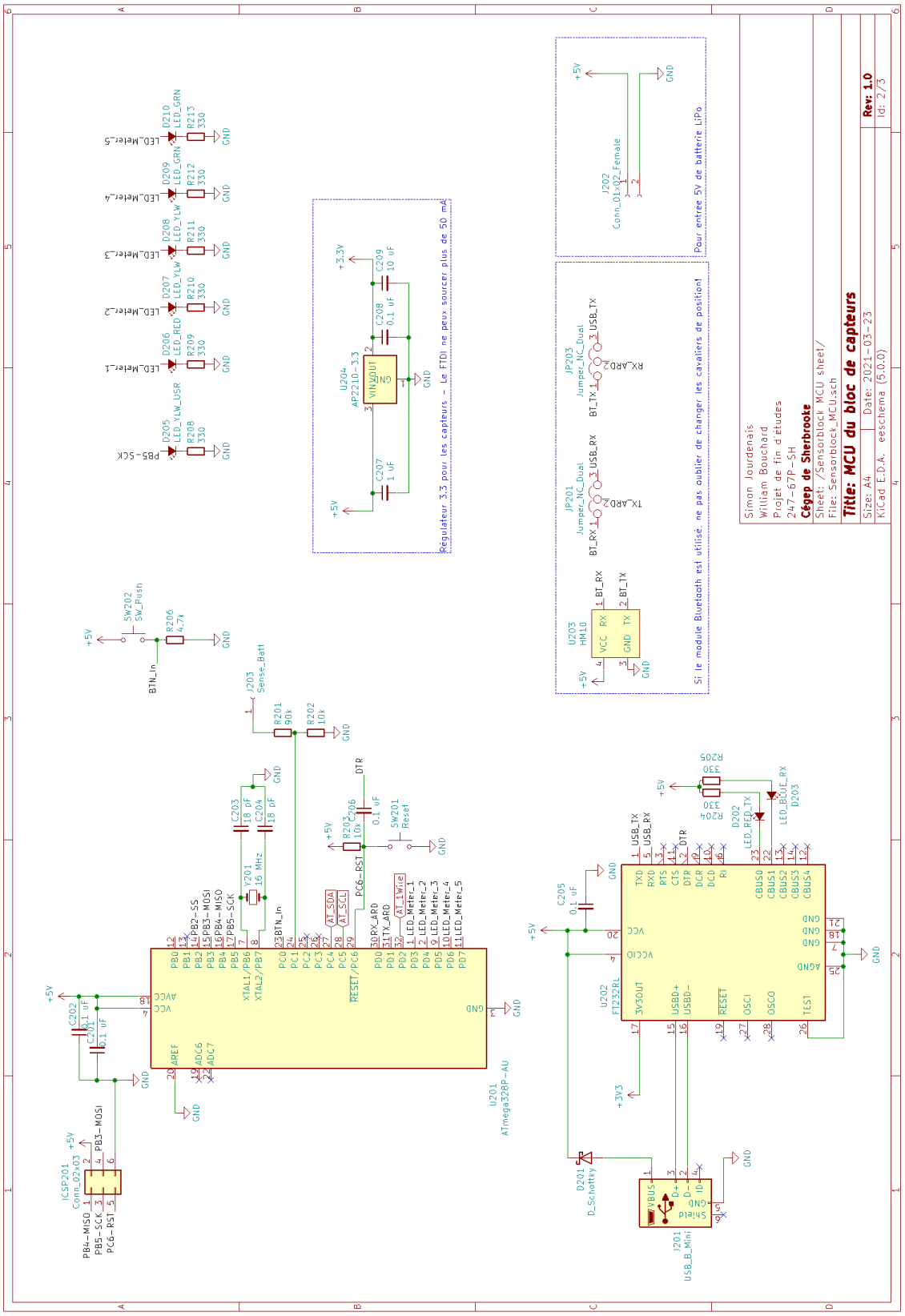


Figure 18 Schéma électrique du bloc de capteurs

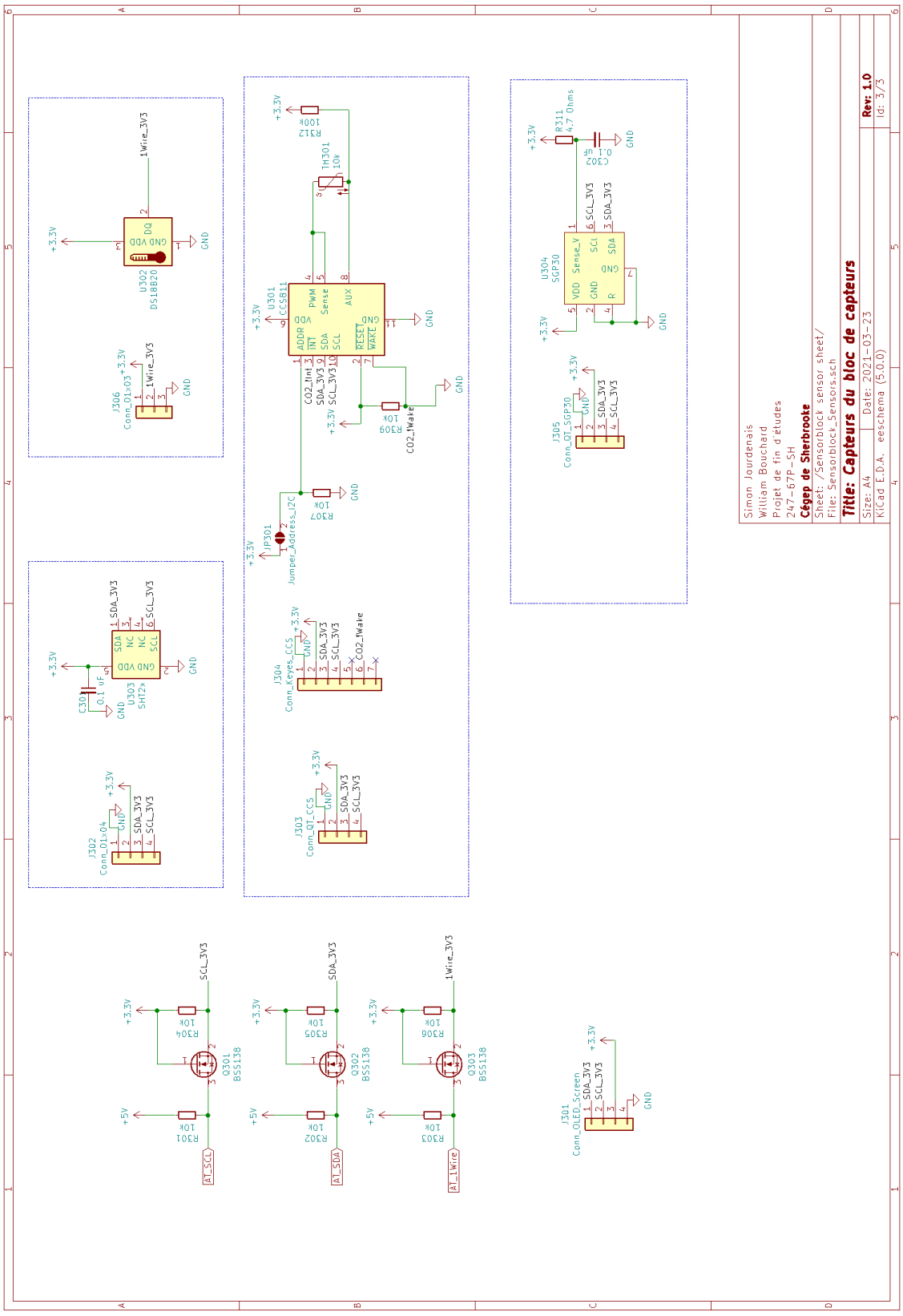


Figure 19 Schéma électrique du bloc de capteurs - Section connecteurs