Sommaire

I. Introduction

Rappel du cahier des charges

La serre doit pouvoir réguler automatiquement les trois grandeurs physiques qui sont :

- l'hygrométrie
- la luminosité
- la température.

L'utilisateur a le droit de choisir parmi deux modes de fonctionnement : le mode profil et le mode manuel. Pour ce faire, une carte de gestion (Raspberry) accueille le système et une carte Arduino fait le lien entre la carte de gestion et les actionneurs. Le système doit savoir quel est le mode utilisé par la serre :

- le mode profil : le système récupère une valeur dans la base de données en fonction de l'heure, des profil en cours, ainsi que la grandeur physique voulue. On dit que c'est un mode automatique.
  Exemple d'un profil :

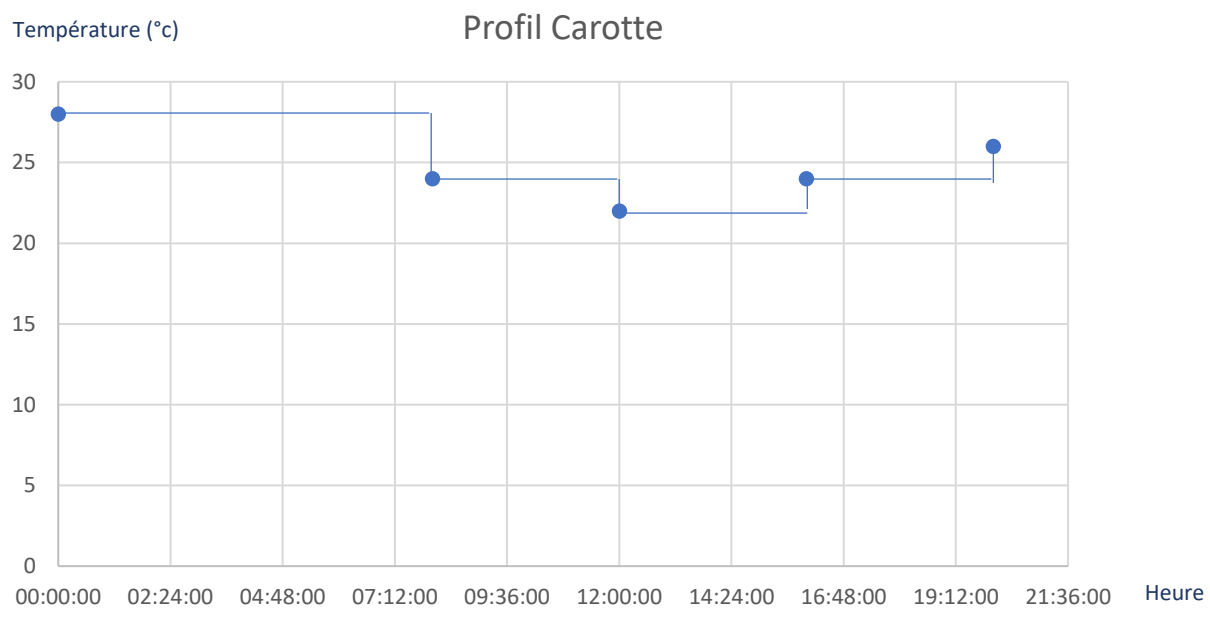


Image 1: Graphique du profil Carotte pour la température

- le mode manuel : ce dernier récupère une valeur dans la base de données en fonction de la grandeur physique.

II. Réalisation du projet

A. Rappel de la tâche de l'étudiant

Aperçu des tâches réalisées

Dans le système de régulation, ma partie de développement consiste à créer un système permettant :

- d'activer les différents actionneurs
- de récupérer des valeurs depuis la base de données
- de créer une connexion en RS232 avec la carte Arduino
- de journaliser toutes les actions du système.

Le langage python est utilisé avec différentes librairies, comme pyserial qui permet de créer une communication en série, mysql-connector-python pour la connexion avec la base de données.
De plus, une application graphique en python a été créée. Elle permet de simplifier la lecture des logs du système pour l'utilisateur.

Contraintes liées au développement

Le système doit être développé sur une carte Raspberry, mais par manque d'outil, l'application PyCharm qui propose divers outils comme le débuggage est utilisée. Il a été aussi rajouté une carte Arduino car il était plus facile de piloter les actionneurs avec cette-ci.

B. Communication entre les différents éléments

Synoptique

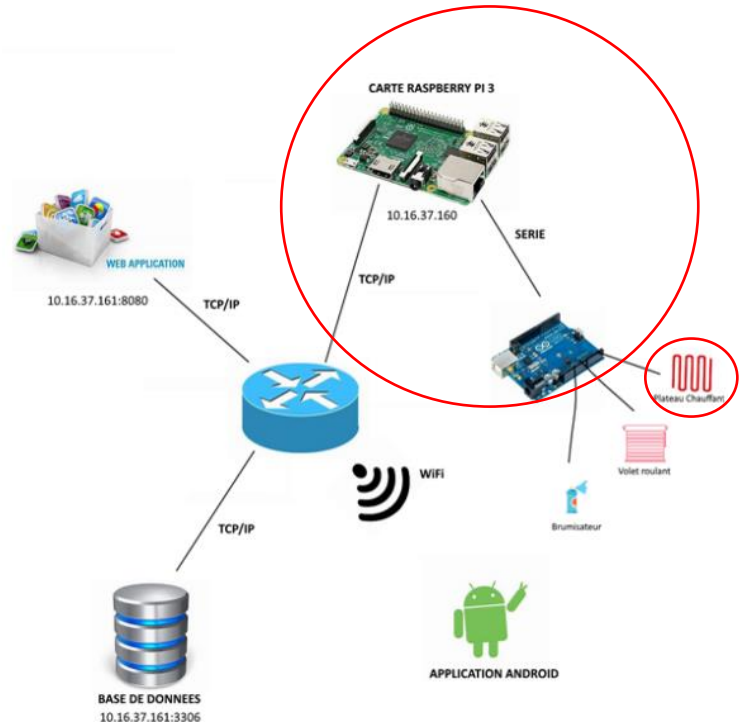


Image 2: Schéma du système de régulation

La partie du projet « Carte de gestion » est entourée en rouge. Cette tâche consiste en la création d'un système de régulation permettant l'activation et la désactivation du plateau chauffant. Dans le cahier des charges, la carte de développement Arduino ne m'était pas imposée mais elle a été ajoutée pour faciliter l'utilisation des actionneurs.

Communication

La communication entre la base de données et la carte de gestion fonctionne grâce au protocole TCP, sachant que la donnée est interprétée par MySQL Server.
En ce qui concerne la communication entre la carte Raspberry et la carte Arduino, celle-ci se fait en série grâce au protocole RS232.
Enfin, le programme est écrit de telle sorte que le technicien puisse ajouter facilement un autre protocole de communication, comme Ethernet, et que l'utilisateur puisse choisir entre ces différents protocoles. Pour cela il existe une classe « ComSerial » (Classe fille) qui hérite d'une classe mère « Communication ». Pour rajouter un nouveau protocole de communication, il suffit de créer une nouvelle classe exemple : « Ethernet » et elle doit hériter de la classe mère Communication.

C. Information de connexion au serveur

Lors des phases de tests du programme, le réseau SN du secteur Informatique est utilisé.
Le serveur est configuré de la manière suivante :

- Adresse IP : 10.16.37.161
- Nom de la base de données : BDD_Serre_Automatique
- Nom utilisateur : sfl6
- Mot de passe : sfl6db

La carte de régulation est configurée de la manière suivante :

- Adresse IP : 10.16.37.162:2222
- Nom de l'utilisateur : pi
- Mot de passe : password
- Le chemin du système : /home/pi/regulation

D. Structure du protocole de communication

Pour communiquer avec la carte Arduino un protocole bien spécifique est créé. La donnée envoyée est sous le format : NOM_DE_L'ACTIONNEUR:ACTION:PORT

Prenons l'exemple du plateau chauffant, le port du plateau est 6.
Il peut y avoir deux cas :

- Activé : PLATEAU CHAUFFANT:ACTIVATE:6
- Désactivé : PLATEAU CHAUFFANT:DESACTIVATE:6

III. Plateau chauffant

A. Schéma électrique du câblage

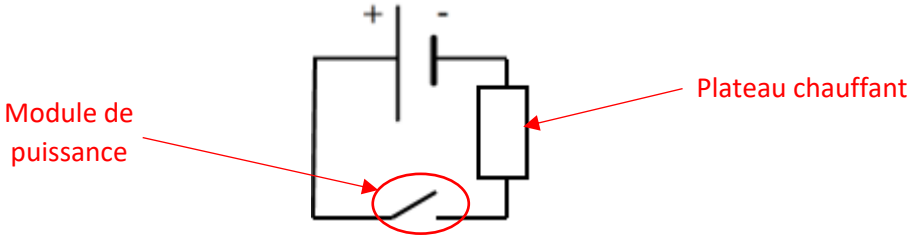


Image 3: Schéma de câblage du système (Plateau chauffant)

Pour activer le plateau chauffant, il suffit de lui envoyer du courant. Ce plateau chauffant fonctionne avec un courant de 18 V. Un module de puissance est un ensemble de semi-conducteurs de puissance interconnectée entre eux dans le même boîtier. Il permet de commander un élément de puissance à courant continu exemple : Moteur, résistance, ampoule. Le module de puissance utilisé dans ce système est le K_AP_MPWR. Il possède un MOSFET (Metal Oxide Semiconductor Field Effect Transistor) IRL520N, pour changer son état, il suffit d'envoyer une tension. On peut donc dire qu'il fonctionne comme un interrupteur ON/OFF. Il permet de gérer le passage du courant selon l'état du MOSFET.

B. Évolution de la température

Lors de l'activation ou la désactivation des actionneurs (ici plateau chauffant), un temps d'attente débute. Pendant ce temps, toute action est impossible. Cette attente est stockée dans la base de données pour chaque actionneur. La colonne qui concerne cette période s'appelle « Periode_Action » Pour déterminer le temps d'attente, des tests ont été établis.

Présentation du test

La tige du thermomètre est placée en contact avec le plateau chauffant. Le but de ce test est d'évaluer l'évolution de la température. L'image ci-dessous représente le test :

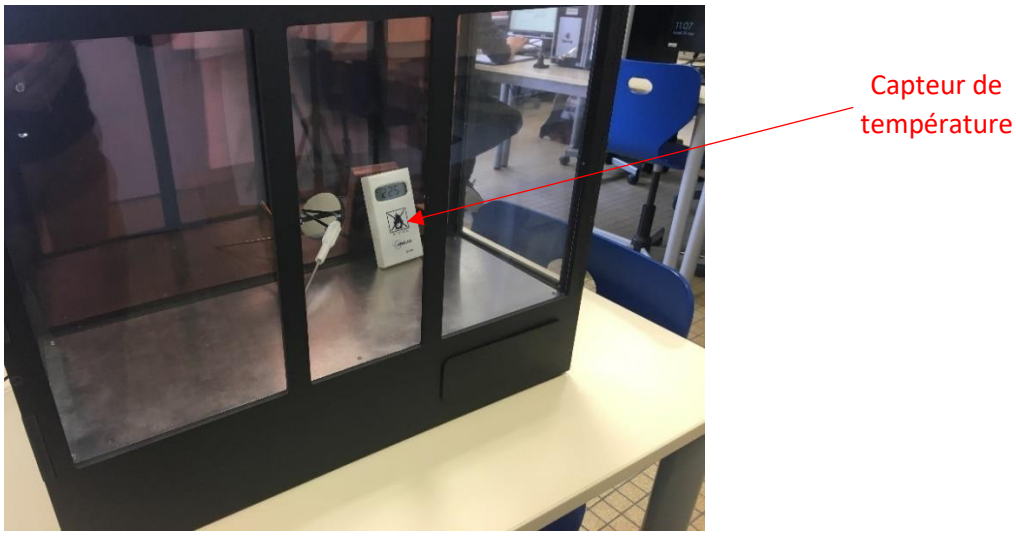


Image 4: Illustration du test : « évolution de la température »

Plateau chauffant actif

La courbe ci-dessous représente l'évolution de la température lorsque le plateau chauffant est activé (donc chauffe).

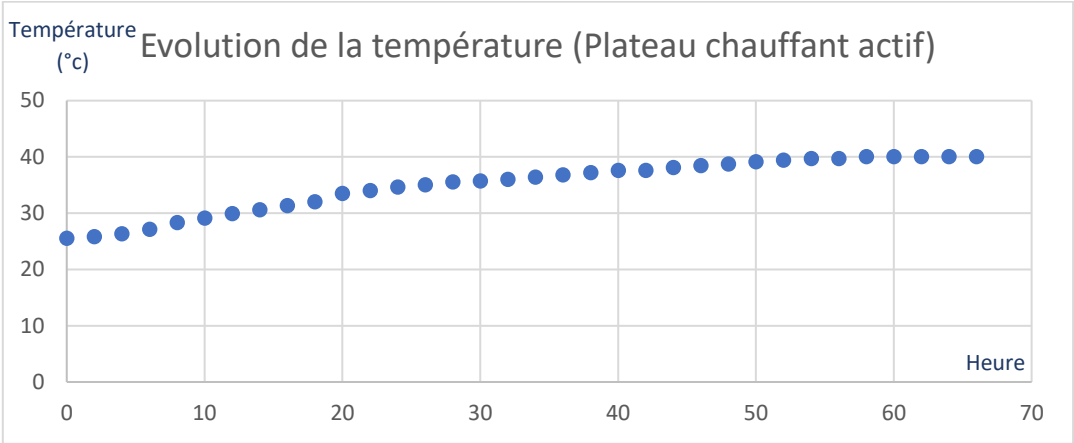


Image 5: Graphique de l'évolution de température (mode actif)

En conclusion, le plateau chauffant a atteint sa température max (40°c) en 60 minutes soit 1 heure.

Plateau chauffant inactif

Après que le plateau chauffant ait atteint sa température maximum, des relevés ont été pris afin de connaître en combien de temps le plateau chauffant atteint 35 °c.

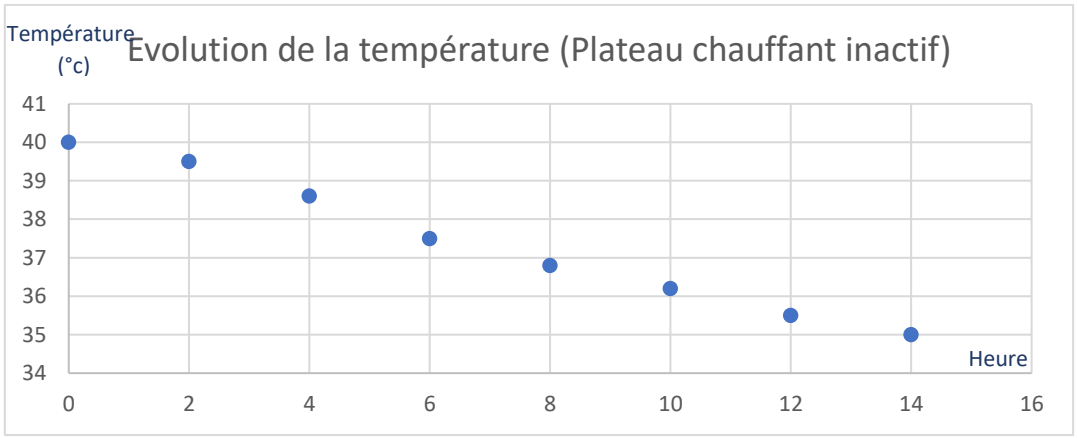


Image 6: Graphique de l'évolution de la température (mode inactif)

On constate qu'en 14 minutes le plateau chauffant perd 5°c.

Conclusion

Après les deux tests établis, j'ai décidé de prendre une période d'action de 20 minutes soit 1200 secondes. La période d'action est fixée pour la maquette, l'utilisateur ou le technicien peut modifier cette valeur via l'application WEB. De plus, dans le système réel, la température sera régulée par des tuyaux d'eaux chaudes. Un autre test devra donc être réalisé pour déterminer une nouvelle période d'action. Attention la période d'action doit être en seconde.

IV. Réalisation du programme de régulation

A. Carte Raspberry

Base de données

La classe SQLConnection permet de créer un objet composé de différentes méthodes pour récupérer ou modifier la base de données.

Mise en place de la connexion

La méthode connection() permet d'établir une connexion entre la base de données et le système de régulation

```
def connection(self):
    try:
        self.conn = mysql.connector.connect(host=self.m_hostname, user=self.m_user,
                                            password= self.m_password,
                                            database=self.m_database) #Créer la connection à la BDD
        self.cursor = self.conn.cursor() #Creation du cursor
        self.log.info("BDD Connection !") #Ecrit dans le fichier log
    except ValueError:
        self.log.error("BDD Connection fail !")
```

Image 7: Code de la méthode « connection »

La méthode « connect » (voir programme ci-dessus) qui appartient à la librairie mysql-connector permet de faire cette connexion. Elle demande différents arguments :

- host : Qui correspond à l'adresse IP du serveur,
- user : Qui correspond à l'utilisateur MySQL du serveur,
- password : Qui correspond au mot de passe de l'utilisateur,
- database : Qui pointe sur la bonne base de données,

Ces valeurs se renseignent dans le constructeur de SQLConnection. Lorsqu'on appelle cette méthode, il peut arriver que la connexion échoue. Les raisons de cet échec sont diverses :

- les informations pour la base de données (host, user, password et database) peuvent être erronées,
- le serveur peut être éteint,
- le serveur ou le système ne sont pas sur le même réseau.

Méthode Query et Write

Ces deux méthodes permettent d'optimiser le code. Elles sont en visibilité privée car seulement la classe SQLConnection peut y accéder.

Query :

La méthode « query » permet d'interroger le serveur et de renvoyer une donnée selon la commande.

```
def __query(self, queryCommand):
    self.cursor.execute(queryCommand)
    result = self.cursor.fetchall()
    return result
```

Image 8: Code de la méthode « query »

Write :

La méthode « write » permet d'envoyer une commande à la base de données.

```
def __write(self, command):
    self.cursor.execute(command)
    self.conn.commit()
```

Image 9: Code de la méthode « write »

Récupération des actionneurs

```
def getActuators(self):
    try:
        self.log.info("Get actuators !") #Ecrit dans le fichier log
        return self.__query("SELECT * FROM Actionneur") #Retourne sous forme de tableau les actionneurs
    except ValueError:
        self.log.error("Get actuators fail !")
```

Image 10: Code de la méthode « getActuators »

La méthode « getActuators() » permet de récupérer les différentes informations de chaque actionneur. Elle permet tout simplement de rendre le système automatique car si l'utilisateur crée un nouvel actionneur, alors celui-ci sera pris en compte dans le système.
Le technicien ou l'utilisateur peuvent ajouter un actionneur via l'application web développé par l'étudiant Goulven Perrin.

Réalisation de la connexion RS232

La connexion entre la carte Raspberry et la carte Arduino s'effectue grâce à la classe ComSerial.

Le constructeur ComSerial() :

```
def __init__(self, log, m_port=None):
    self.log = log
    try:
        if m_port is not None:
            self.port = m_port
        else:
            self.port = self.__getInstrumentByName("Arduino")
            if self.port is None:
                self.port = self.__getInstrumentByType("USB")
        if self.port is not None:
            try:
                self.ser = serial.Serial(port=self.port, baudrate=9600)
                self.log.info("Instrument found : {}".format(self.port))
            except:
                self.log.error("Watch if the COM port of the instrument is not opened by another application!")
        else:
            self.log.error("Not instrument founded !")

    except TypeError:
        self.log.error("Not found instrument in port {}".format(m_port))
```

Image 11: Code du constructeur de la classe « ComSerial »

Il permet de réaliser la connexion entre la carte Raspberry et la carte Arduino de manière automatique. Pour ce faire, le système utilise deux méthodes :

- getInstrumentByName() : Elle permet de chercher sur le port série tous les instruments ayant le nom Arduino.

- getInstrumentByType() : Si la première méthode ne trouve pas en fonction du nom ici « Arduino » alors le système cherche en fonction du type de l'instrument ici « USB »

La classe ComSerial dispose d'autres méthodes comme :

- write() : permet d'écrire sur le port com
- read() : permet de lire sur le port com et retourne la valeur
- query() : permet d'écrire et lire sur le port com et retourne la valeur

Régulation

La régulation est lancée lors du démarrage du système. Pour réguler, le programme utilise la méthode « run() » qui se trouve dans la classe « Actuator » (Voir Annexe 1). Elle appelle différentes méthodes présentées en page suivante :

Récupérer le mode actuel

Pour récupérer le mode actuel, la méthode « getMode() » appartenant à la classe SQLConnection est utilisée. Elle envoie une requête query à la base de données, et elle nous retourne le mode actuel. Code ci-dessous :

```
def getMode(self):
    try:
        data = self.__query("SELECT mode FROM ConfigRegulation") #Récupère le mode actuel dy système
        self.m_modeCurrent = self.parseTabData(data) #Permet de tranformer un tableau en entier
        return self.m_modeCurrent #Retourne le mode actuel du système
    except ValueError as err:
        self.log.error(err)
```

Image 12: Code de la méthode « getMode() »

Récupérer la valeur voulue

Pour ce faire, le système appelle une méthode en fonction du mode actuel.

Dans le premier cas, il appelle la méthode « getWantedValueProfil() » si le mode est égal à « Profil ».

```
def getWantedValueProfil(self, physicalId):
    try:
        data = self.__query("SELECT `valeur` FROM `ValeurHoraire`, Grandeur_Physique WHERE heure <= NOW() "
                            "AND ValeurHoraire.Grandeur_Physique_id = Grandeur_Physique.id "
                            "AND Grandeur_Physique.id = {} ORDER BY `heure` DESC "
                            "LIMIT 1".format(physicalId))
        if len(data) != 0: #Vérifie si la donnée retourné n'est pas vide
            return self.parseTabData(data) #Permet de tranformer un tableau en entier
    except ValueError:
        return None
```

Image 13: Code de la méthode « getWantedValueProfil » pour le mode profil

Cette méthode récupère dans la colonne « valeur » depuis la table « ValeurHoraire » une valeur en fonction de deux paramètres :

- L'heure : Elle correspond à la valeur actuelle du système
- La Grandeur physique : Elle demande l'ID de la grandeur physique

Dans le deuxième cas, il appelle la méthode « getWantedValueManual() » si le mode est égal à « Manual ».

```
def getWantedValueManual(self, physicalId):
    try:
        data = self.__query("SELECT valeur FROM ConfigManuelle WHERE Grandeur_Physique_id = \"{}\"".format(physicalId))
        if len(data) != 0: #Vérifie si la donnée retourné n'est pas vide
            return self.parseTabData(data) #Permet de tranformer un tableau en entier
        return None
    except ValueError as err:
        pass
```

Image 14: Code de la méthode « getWantedValueManual() »

Cette méthode récupère, dans la colonne « valeur » depuis la table « ConfigManuelle », une valeur en fonction de la Grandeur Physique.

Récupérer la valeur actuelle

Pour récupérer la valeur de actuel pour chaque Grandeur Physique, la méthode « getActualValue() » est alors appelée. Voir code ci-dessous :

```
def getActualValue(self, physicalId):
    try:
        data = self.__query("SELECT valeur FROM Releve,Capteur,Grandeur_Physique WHERE Releve.Capteur_id = Capteur.id "
                            "AND Capteur.Grandeur_Physique_id = Grandeur_Physique.id "
                            "AND Grandeur_Physique.id = {} ORDER BY `Date` DESC LIMIT 1".format(physicalId))
        if len(data) != 0:  #Vérifie si la donnée retourné n'est pas vide
            return self.parseTabData(data)  #Permet de tranformer un tableau en entier
    except ValueError as err:
        print(err)
        pass
```

Image 15: Code de la méthode « getActualValue() »

La requête récupère une valeur dans la colonne « valeur » depuis la table « Releve ». Cette table correspond à toutes les valeurs enregistrées depuis les capteurs du projet SFL5 « Serre Automatique Système d'acquisition ». Le procédé « ORDER BY 'Date' DESC LIMITE 1 » permet, dans un premier temps, de trier du plus grand au plus petit selon la date et ensuite de définir une limite de 1 pour récupérer la dernière valeur en fonction de la Grandeur Physique.

Période d'action

La méthode « getIntervalBetweenActualTimeToLastTimeUseActuator() » est une méthode qui retourne deux valeurs dans un tuple. Un tuple est un type de liste en python avec lequel il est impossible de modifier la taille et les valeurs à l'intérieur. Code ci-dessous.

```
def getIntervalBetweenActualTimeToLastTimeUseActuator(self, physicalId):
    data = self.__query("SELECT heure_derniere_utilisation FROM Actionneur WHERE Grandeur_Physique_id = {}".format(physicalId))
    data = self.parseTabData(data)

    if data is None:
        self.__write("UPDATE Actionneur SET heure_derniere_utilisation = NOW() WHERE Grandeur_Physique_id = {}".format(physicalId))
        data = self.__query("SELECT heure_derniere_utilisation FROM Actionneur WHERE Grandeur_Physique_id = {}".format(physicalId))
        data = self.parseTabData(data)
        self.log.info("Update time Actuator")

    actualDateIsDifferentActuatorDate = False
    if data < datetime.datetime.now():
        actualDateIsDifferentActuatorDate = True

    resActuator = data.time()
    resActual = datetime.datetime.now()

    secondsActual = self.convertHourToSecond(resActual)
    secondsActuator = self.convertHourToSecond(resActuator)

    """print("Actuator {}:{}\n Now {}:{}".format(resActuator, secondsActuator, resActual, secondsActuator))
    print("Actual : {}".format(secondsActual))
    print("Actuator : {}".format(secondsActuator))"""

    return (actualDateIsDifferentActuatorDate, secondsActual - secondsActuator)
```

Image 16: Code de la méthode « getIntervalBetweenActualTimeToLastTimeUseActuator() »

La première valeur de la liste correspond à un Boolean qui soit est « Vrai », dans ce cas la date récupérée depuis la requête SQL est inférieure à la date actuelle du système. Sinon elle retourne « Faux ».
La deuxième valeur de la liste correspond à une soustraction entre deux valeurs :

- La première valeur correspond à l'heure actuel en seconde
- La deuxième valeur correspond à l'heure récupérée par la requête SQL en seconde (Cette valeur récupérée correspond à la dernière utilisation de l'actionneur depuis le système).

B. Carte Arduino

Connexion RS232

Sur Arduino, une connexion en RS232 est effectuée en utilisant la librairie SoftwareSerial. Les méthodes suivantes sont appelées :

- begin() : permet de définir les baud rates. Les baud rates sont des débits de symbole.
- write() : permet d'écrire sur le port série
- read() : permet de lire sur le port série
- readString() : permet de lire une chaîne de caractères sur le port série

Comment fonctionne le système ?

Il faut savoir que Arduino a son propre langage de programmation. La carte Arduino est utilisée seulement pour piloter les actionneurs. Elle reçoit différentes commandes depuis la carte Raspberry qui sont présentées dans la partie « **Structure du protocole de communication** ». Après avoir reçu cette donnée, le programme découpe la donnée en fonction d'un délimitateur « : » (elle remplace la méthode Split).

- La première case du tableau correspond au nom de l'actionneur
- La deuxième case du tableau correspond à l'action « ACTIVATE | DESACTIVATE »
- La dernière case du tableau est tout simplement le port de l'actionneur

Grâce à ces trois informations, le système est capable de piloter le bon actionneur sur le bon port avec la bonne action.

C. Journalisation

Le cahier des charges impose de journaliser tous les évènements du système pour que l'utilisateur et le technicien sache ce qui se passe. Le système utilise la librairie « Logging » qui permet de créer un fichier et d'y écrire. Trois types de message existent :

- INFO : Ce message indique l'action que fait le système
- WARNING : Ce message indique qu'il y a une anomalie dans le système
- ERROR : Ce message indique qu'il y a eu une erreur dans le système

Format de message

```
logging.basicConfig(level=logging.DEBUG,
                    filename="system.log",
                    format="[%(asctime)s]::[%(levelname)s]::%(message)s",
                    datefmt='%m/%d/%Y %I:%M:%S')
```

Image 17: Code de la création du format de log

L'image ci-dessus montre le format du message :

- level permet de définir le type de message
- filename permet de créer un fichier pour enregistrer les messages
- format permet de définir le format de message
    - asctime permet d'afficher le temps
    - levelname permet d'afficher le type du message
    - message permet d'afficher le message

V. Réalisation de l'application « Journalisation »

Pour que l'utilisateur s'informe des différentes actions faites par le système de régulation, l'application de journalisation lui permet d'ouvrir un fichier de type « .log ». Après l'ouverture de ce fichier, il peut choisir la date avec un calendrier et le type de message (INFO | WARNING | ERROR).
L'utilisateur peut :

- Ouvrir un fichier (.log)
- Fermer le fichier
- Choisir la date
- Choisir le type de message (INFO | WARNING | ERROR)

Les fonctions à venir :

- Se connecter à une base de données
- Lire les logs sur la base de données

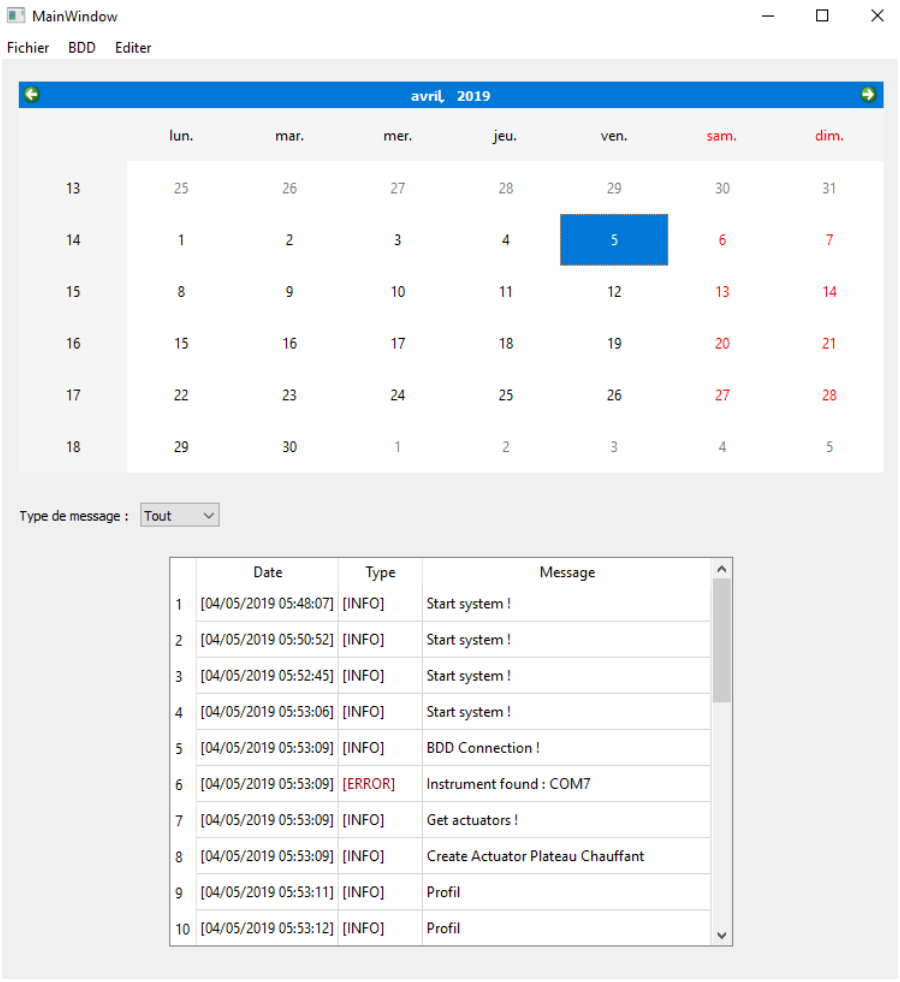


Image 18: Application graphique de la journalisation

Attention le fichier ouvert ne peut être modifié par l'application !

VI. Problèmes rencontrés

Durant le projet, j'ai rencontré plusieurs problèmes :

- Dans la partie analyse, il était difficile de comprendre ce qu'était le mode Profil. Pour mieux comprendre nous avons beaucoup discuté avec notre professeur référent Monsieur ANGIBAUD pour savoir ce que caractérisait un profil.

- Toujours dans la partie analyse, j'ai dû élaborer un diagramme de classe du système de régulation et par la suite des diagrammes de séquences. La partie diagramme de séquence a été compliqué, car j'avais du mal encore à comprendre comment le système pouvait fonctionner.

- Lors de la phase conception, j'ai commencé dans un premier temps à faire fonctionner mon plateau chauffant. Au départ, le plateau chauffant fonctionnait en continue (il était actif). Mais lorsqu'on envoyait 5V sur la carte de puissance, le MOSFET, qui je rappelle fonctionne comme un interrupteur, ne changeait pas d'état (le courant passait toujours et donc le plateau chauffant était actif). Pour cela, avec le professeur de physique Monsieur BRIDONNEAU, nous avons décidé de lire la documentation technique de la carte. En regardant de plus près, nous en avons conclus qu'il y avait un sens pour le câblage. Nous avons donc changé le sens de câblage en changeant juste le sens de la carte de puissance. Nous avons ensuite renvoyé 5V sur la carte et nous nous sommes aperçus que le plateau chauffant changeait d'état.

- Le dernier problème que j'ai rencontré était de récupérer la dernière valeur voulue pour le mode profil. Pour cela, j'ai écrit sur une feuille le chemin que devais faire ma requête SQL pour parvenir à la bonne valeur. J'ai donc, grâce à cette technique, pu faire face à cette difficulté rapidement.

VII. Test unitaire

Pour garantir la fiabilité du programme, un test unitaire a été mis en place. Il permet de récupérer une valeur dans la base de données en fonction d'une grandeur physique et d'une valeur horaire.

Elément testé :	Valeur voulue – Fichier : « unittest.py »		
Objectif du test :	Récupération de la valeur voulue pour le mode profil en fonction de l'heure actuelle et de la grandeur physique		
Nom du testeur :	Baptiste MAGONI	**Date :**	24/04/2019
Moyens mis en œuvre :	Logiciel : CarteDeGestionRegulation	Matériel : Carte Raspberry, Base de données (MySQL)	Outil de développement : Python
Procédure de test			
Description du vecteur de test	**Résultat attendu**	**Résultat obtenu**	**Validation**
Une fois que l'utilisateur est dans le répertoire du projet, on lance le fichier « unittest.py » avec la commande « python unittest.py »	Le programme se lance et affiche la valeur attendue avec la grandeur physique 1 et il était 14 : 50	`Valeur attendue: 56`	❑
Connexion avec l'utilisateur : « sfl6 » et le mot de passe « sfl6db » sur phpMyAdmin	Accueil de phpMyAdmin	Voir Annexe 4	❑
Se déplacer dans la table « BDD_Serre_Automatique » et ensuite aller dans « ValeurHoraire »	Liste de toutes les valeurs horaires	Voir Annexe 5	❑
Maintenant, il faut regarder dans la colonne valeur	Regarder en fonction de l'heure ici « 14 :50 » et la grandeur physique 1	56	❑
Conclusion du test			

VIII. Fiches recettes

Dans cette partie, trois fiches recettes sont destinées au client. Elles permettent de valider le fonctionnement du système.

A. Connexion Raspberry

Nom : Se connecter à la Raspberry	
Recette : Technique	
Objectif	Savoir si le compte utilisateur fonctionne
Elément à tester	Connexion SSH
Pré requis	Avoir au préalable téléchargé une application pour une connexion SSH comme PuTTY ou MobaXterm

Scénario				
Id	Démarche	Données	Comportement attendu	OK
1	Créer une nouvelle session SSH sur l'application et entrer l'adresse 10.16.37.161 et le port 22		Une nouvelle session s'est créée et vous arrivez sur un shell	❑
2	On s'authentifie au système	pi password	Vous êtes maintenant connecté au système de régulation	❑

Rapport de test	Testé par : L'Etudiant 1	Le : 24/04/2019
Conformité ❑ Excellente ❑ Moyenne ❑ Faible		
Commentaire :		Approbation :

B. Lancer le système de régulation

Nom : Lancer le système de régulation	
Recette : Technique	
Objectif	Lancer le système de régulation afin que le système puisse contrôler les différents actionneurs du système
Elément à tester	Le programme de régulation
Pré requis	Etre connecté au système en SSH (Voir Connexion Raspberry)

Scénario				
Id	**Démarche**	**Données**	**Comportement attendu**	**OK**
1	Se déplacer dans le bon dossier.	Entrez la commande : « cd Regulation »	Vous êtes à présent dans le dossier du système.	❑
2	On lance le système.	Entrez la commande : « python Main.py »	Le système est maintenant lancé	❑

Rapport de test	Testé par : L'Etudiant 1	Le : 24/04/2019
Conformité ❑ Excellente ❑ Moyenne ❑ Faible		
Commentaire :		**Approbation :**

C. Lecture des événements journalisés

Nom : Lecture des évènements journalisés	
Recette : Technique	
Objectif	Lancer l'application « AppLog » pour lire tous les évènements du système en fonction de la date et du type de l'évènement
Elément à tester	Le bon fonctionnement de l'application
Pré requis	Avoir lancé l'application

Scénario				
Id	**Démarche**	**Données**	**Comportement attendu**	**OK**
1	Dans l'onglet « Fichier ». Sélectionner « Ouvrir »		Vous êtes maintenant dans l'explorateur de l'ordinateur	❑
2	Sélectionner un fichier de type « .log »	Pour l'exemple choisissez le fichier dans le répertoire du système « system.log »	Maintenant le fichier est lu et vous pouvez choisir le type d'information et la date.	❑
3	Pour l'exemple, sélectionner la date du 3 ou 5 mai 2019.		En sélectionnant la bonne date, les messages log s'affichent dans le tableau en bas de l'application	❑
4	Sélectionnez dans « type de message »	Le type « WARNING »	Tous les message warning s'affichent dans le tableau	❑

Rapport de test	Testé par : L'Etudiant 1	Le : 24/04/2019
Conformité ❑ Excellente ❑ Moyenne ❑ Faible		
Commentaire :		**Approbation :**

IX. Conclusion

A. Communication de groupe

Durant tout le projet nous avons mis en place une cohésion de groupe. Pour cela, plusieurs outils et méthode ont été utilisés :

- Diagramme de Gantt pour se partager le travail et savoir quelle tâche nous devions traiter lors de notre arrivée en projet,
- La plateforme GitHub et le logiciel GitHub Desktop qui nous a permis d'échanger nos documents et programme,
- Toute les chartes graphiques (Word, PowerPoint) sont les mêmes avec les étudiants
- Des réunions hebdomadaires en début pour savoir ou en était rendu le projet.

Ces méthodes m'ont permis d'apprendre le travail en équipe et de renforcer un lien avec les autres étudiants. Ces échanges nous on permis de discuter sur des points techniques notamment sur la base de données qui est commune avec le groupe SFL5 « Serre Automatique Système d'acquisition ».
Je remercie les étudiants Goulven Perrin et Alexandre Stanisiere pour leur participation au projet.

B. Regard critique du projet

Le projet était très intéressant, le plateau chauffant utilisé par la maquette de mon point de vue ne peut pas être utilisé, car il met beaucoup trop de temps à chauffer ce qui ne suffit pas pour chauffer entièrement la serre.

C. Connaissances apportées

J'avais déjà des connaissances en python, car mon sujet de stage était de créer un protocole de communication entre un ordinateur et une carte électronique pour microcontrôleur. J'ai créé une application graphique pour utiliser ce protocole toujours en python. J'ai pu quand même approfondir les bases de la programmation objet en python. Je suis toujours passionnée par la programmation et aimerais en faire mon métier plus tard.

Durant le projet, je suis allé aussi m'informer au prêt des autres étudiant à propos de leur projet. J'ai aussi aidé des personnes sur le plan technique pour résoudre quelques problèmes comme :

- Dépanner un élève sur une erreur de compilation pour une application Android,
- De montrer comment communiquer entre une carte Raspberry et une carte Arduino,
- De créer des blocs dynamiquement à l'aide de la base de données.

Bien que ces aides étaient ponctuelles afin de ne pas restreindre mon travail sur mon projet, j'ai pu apprécier ces interventions.

D. Ce qui me reste à faire

Au moment de la rédaction de ce rapport, il me reste à :

- Tester le programme de régulation avec l'application WEB et l'application Android
- Réaliser la documentation de mon programme sous forme de site internet

- Créer la documentation utilisateurs pour l'installation du système

Et par la suite à améliorer le projet :

- Créer un nouveau protocole (Ethernet) de communication entre la carte Raspberry et la carte Arduino
- Finir mon application de journalisation
- Rendre mon application de journalisation plus esthétique
- Lors du lancement de la carte Raspberry faire en sorte de lancer le programme principal

E. Poursuite d'étude

À la suite des tests, ma candidature a été retenue par l'ENI (l'école nantaise d'informatique) pour une formation de « Concepteur et développeur d'application » pour 2 ans et par l'EPSI (École professionnelle des sciences informatiques) pour dans un premier temps un Bachelor en informatique. Aujourd'hui, j'ai pour objectif d'obtenir un bac +5 pour développer tout type d'application dans une entreprise avec des projets ambitieux et innovants. Je cherche une alternance dans le domaine de l'informatique plus particulièrement, dans le domaine du développement.

Annexe

Annexe 1. Méthode run()

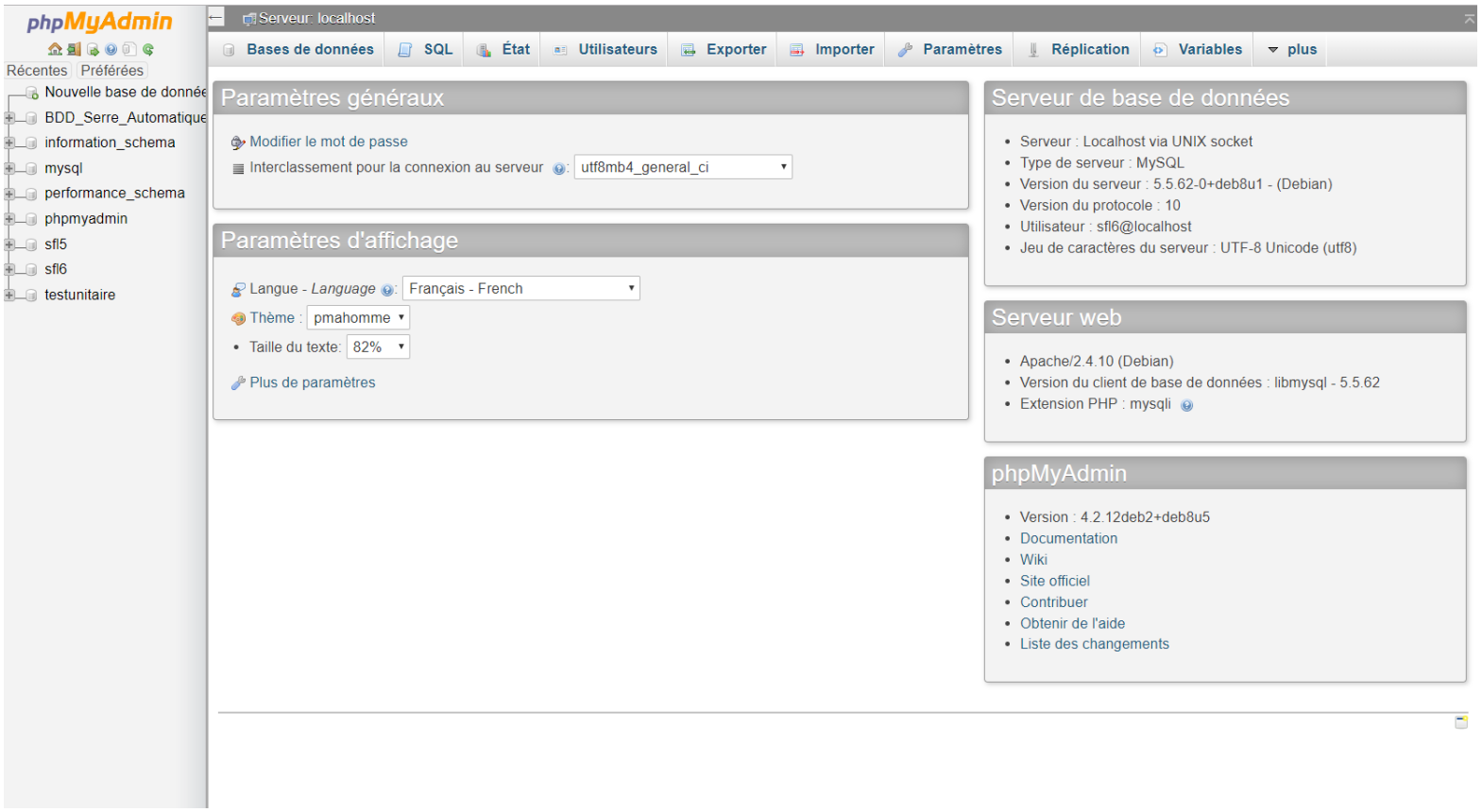
```
def run(self):
    if self.m_sql.getMode() == "Manual":
        self.log.info("Manual")
        self.wantedValue = self.m_sql.getWantedValueManual(self.m_physicalId)
    else:
        self.log.info("Profil")
        self.wantedValue = self.m_sql.getWantedValueProfil(self.m_physicalId)
    self.actualValue = self.m_sql.getActualValue(self.m_physicalId)

    res = self.m_sql.getIntervalBetweenActualTimeToLastTimeUseActuator(self.m_physicalId)
    if (self.actualValue is not None) and (self.wantedValue is not None):
        try:
            if res[0]:
                if self.timePeriod < res[1]:
                    self.actuatorProcedure()
                else:
                    self.log.info("{} : No action can be done".format(self.m_name))
            else:
                self.actuatorProcedure()
        except ValueError:
            self.log.warning("{} : ActualValue or WantedValue are probably None !".format(self.m_name))
```

Annexe 2. Séparation des éléments

```
for(int i=0;i<data.length();i++){
  if(data[i] != ':'){
    list[index] += data[i];
  }else{
    index += 1;
  }
}
```

Annexe 3. Interface phpMyAdmin



Annexe 4. phpMyAdmin vue « ValeurHoraire »

←T→	id ▲ 1	heure	valeur	Profil_id	Grandeur_Physique_id
Modifier Copier Effacer	2	00:00:00	200	1	2
Modifier Copier Effacer	3	00:00:00	25	1	3
Modifier Copier Effacer	5	09:00:00	300	1	2
Modifier Copier Effacer	6	08:00:00	22	1	3
Modifier Copier Effacer	8	17:00:00	450	1	2
Modifier Copier Effacer	11	21:00:00	250	1	2
Modifier Copier Effacer	12	21:00:00	20	1	3
Modifier Copier Effacer	14	00:00:00	550	2	2
Modifier Copier Effacer	15	00:00:00	27	2	3
Modifier Copier Effacer	16	15:00:00	60	2	1
Modifier Copier Effacer	17	15:00:00	700	2	2
Modifier Copier Effacer	18	15:00:00	32	2	3
Modifier Copier Effacer	20	22:00:00	900	2	2
Modifier Copier Effacer	21	16:00:00	1000	2	2
Modifier Copier Effacer	22	17:00:00	40	2	2
Modifier Copier Effacer	23	02:00:00	700	2	2
Modifier Copier Effacer	25	02:00:00	25	1	1
Modifier Copier Effacer	26	00:00:00	700	2	2
Modifier Copier Effacer	27	00:00:00	700	2	2
Modifier Copier Effacer	28	00:00:00	700	2	2
Modifier Copier Effacer	29	00:00:00	700	2	2
Modifier Copier Effacer	30	00:00:00	40	1	3
Modifier Copier Effacer	32	22:00:00	254	4	2
Modifier Copier Effacer	34	17:00:00	56	1	1
Modifier Copier Effacer	35	17:00:00	56	5	5