ECOLE POLYTECHNIQUE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Rapport intermédiaire

TRAN-H-201

« Le Thermostat Intelligent »

Étudiants :

Amzur Soufiane
Daubry Wilson
Sercu Stéphane
Verstraeten Denis

Superviseur :

Eppe Stefan

14 Décembre 2015

Abstract

"Smart thermostat using Arduino and Raspberry Pi", by Amzur Soufiane, Daubry Wilson, Sercu Stéphane and Verstraeten Denis, Université Libre de Bruxelles, 2015-2016.

The purpose of this project is to design a smart thermostat. The principle is simple : a Raspberry Pi 2 Model B with Python and a Wifi antenna communicates with « thin thermostats »thanks to the http protocol. These « thin thermostats »are microcontrollers supplied with a thermistor, a presence sensor and a module Wifi. Data collected can be gathered and stored in a SQLite database. As a first step, a simple reactive control has to be implemented in the central server. It consists of to open or to close thermostatic valve depending on whether it is too coold or too hot. A smarter and more predictive control algorithm will be then implemented using databases in order to learn the habits and the preferences of users. At the end, it will be necessary to set up a rigourous test protocol. The aim of those tests is to ensure the quality of the system as well quantitatively as qualitatively.

Key words : thermostat, microcontroller, smart, databases, server
[Word limit : 173]

"Thermostat intelligent à l'aide d'Arduino et d'un Raspberry Pi", par Amzur Soufiane, Daubry Wilson, Sercu Stéphane et Verstraeten Denis, Université Libre de Bruxelles, 2015-2016.

Le but de ce projet est de créer un système de thermostats intelligents. Un thermostat central doit être implémenté en Python sur un Raspberry Pi 2b équipé d'une antenne Wi-Fi au moyen de laquelle il communique suivant le protocole HTTP avec les « thin thermostats » . Ceux-ci sont composés d'un microcontrôleur Arduino Nano V2, d'un capteur de présence, d'une thermistance et d'un module Wi-Fi et sont montés sur une BreadBoard. Ils ne disposent d'aucune intelligence et servent à prendre des mesures et à les transmettre au Raspberry qui les stocke dans une base de données au format SQLite. Un algorithme de contrôle réactif doit être intégré dans un premier temps. Cette stratégie consiste juste à ouvrir la vanne s'il fait trop froid, et à la couper dans le cas contraire. Ensuite il faut développer un mécanisme de contrôle prédictif et intelligent qui va apprendre des habitudes et des préférences de l'utilisateur. Des stratégies de tests précises, rigoureuses et systématiques doivent être déterminées afin de déterminer qualitativement et quantitativement la qualité du système.

Mots-clés : thermostat, microcontrôleur, intelligence, base de données, serveur
[Limite de mots : 191]

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Actuellement, il y a une très forte augmentation de l'offre et de la demande d'objets connectés ou intelligents, car le désir d'une utilisation simplifiée et automatisée est grandissant. C'est dans cette optique que s'inscrit ce projet multidisciplinaire orienté vers l'informatique : « Thermostat intelligent ». En effet, le dispositif à concevoir se veut doté d'une interface d'utilisation simple ainsi que d'un modèle de prédiction adapté aux habitudes de l'utilisateur.
Le projet d'année des étudiants en bloc deux de l'école polytechnique consiste en la réalisation d'un thermostat intelligent. Ce dernier sera composé de deux types de périphériques, une unité centrale qui sera un Raspberry Pi 2 et des « Remote Devices » qui seront des Arduino Nano. Ce projet est découpé en quatre milestones distinctes. Dans ce rapport, seules les milestones une et deux sont abordées, elles concernent le côté réactif du thermostat. L'aspect intelligent de ce projet, ne pouvant être réalisé qu'avec un thermostat fonctionnel à un niveau basique, sera donc réalisé au second quadrimestre. Les deux premières parties peuvent être envisagées en parallèle, l'Arduino et le Raspberry étant deux dispositifs séparés.

Dans ce rapport, la conception globale du thermostat est abordée en premier. L'implémentation en language Arduino (proche du C) et son câblage suit, avant de passer au thermostat central. Ce dernier nécessite un serveur et une base de données, celle-ci sera utile par après lorsque le code devra interpréter les habitudes de l'utilisateur. Des tests réalisés afin de vérifier que ces dispositifs sont fonctionnels ainsi que les résultats de ces derniers sont explicités, avant de poursuivre avec le fonctionnement de groupe. Cette partie détaille la manière dont le travail à été réparti et organisé. Des perspectives d'évolutions sont également envisagées, celles-ci listant principalement le travail restant à effectuer.

2 Conception

Afin de commencer sur une bonne base et d'offrir une vue globale du projet, un diagramme (présenté en Annexe A) a été construit. Le but premier de celui-ci est d'offrir une vue globale et commune du développement à tous ses contributeurs. Cela permet principalement le partage efficace des différentes parties à développer entre les participants. Le rassemblement de tous les modules est également grandement simplifié. En effet, il est beaucoup plus facile d'assembler des parties du projet qui ont été construites en gardant en tête la place qu'elles occupent dans son ensemble et la façon dont elles sont sensées communiquer les unes avec les autres.

En plus de tous ces avantages pratiques, ce diagramme a un grand pouvoir documentaire puisqu'il offre une vision synthétique et schématique de l'ensemble du projet. Il est évident qu'il n'a pas été construit en une fois. En effet, il a été complété et corrigé lors de l'avancée du développement en fonction des discussions menées concernant certains problèmes et choix qui ont dû être opérés. Le diagramme est explicité et décrit plus en détails dans le reste du rapport puisque chacune des parties principales du projet est détaillée sur cette base.

3 Thermostats auxiliaires

1 Hardware

D'un point de vue hardware, le montage de l'Arduino Nano est réalisé sur une BreadBord. Les éléments ayant été intégrés à ce câblage sont :

— Un Arduino Nano V2 [1]

— Une thermistance et une résistance de 10kΩ [2]

— Un capteur de présence infrarouge (PIR [1] sensor) [3]

— Un module Wi-Fi ESP-8266 [26]

— Une vanne thermostatique

Ces quatre composants doivent être alimentés et leur signaux de retour captés par le microcontrôleur. Leurs spécificités d'utilisation indiquées dans leur datasheet respective sont prises en compte afin d'éviter que le matériel ne soit endommagé. Ils sont donc branchés suivant la logique suivante (voir Figure 3.1) :

Le détecteur de présence est muni de 3 câbles, un servant d'alimentation permettant de recevoir la tension de 5V, un autre étant la terre du circuit et le dernier devant être relié à une pin numérique afin de lire les données fournies par le détecteur. La pin numérique choisie est la pin D11. Ce choix est purement arbitraire, la seule obligation étant que l'entrée soit digitale.

La thermistance nécessite également une tension d'entrée de 5V, celle-ci doit au préalable passer dans une résistance de 10kΩ branchée en série afin que la Loi d'Ohm puisse lui être appliquée. Elle est finalement reliée à la masse. Une lecture analogique du capteur est effectuée au moyen d'un câble reliant la pin A4 à la tension qui entre dans la thermistance.

Enfin, le branchement de module Wi-Fi est un petit peu plus complexe. D'abord, contrairement aux autres composants, il est connecté à une source d'alimentation de 3.3V délivrée par le microcontrôleur. Cette tension doit être dirigée vers les pins « VCC » et « CH_PD » du module Wi-Fi. Comme tous les autres composants, sa borne « Ground » est connectée à la masse. Notons enfin la présence de 2 câbles indispensables qui assurent la communication entre l'Arduino et le module Wi-Fi, ce sont les câbles qui relient respectivement les bornes « Rx » et « Tx » du premier aux bornes « Tx » et « Rx » du second.

1. Passive Infrared

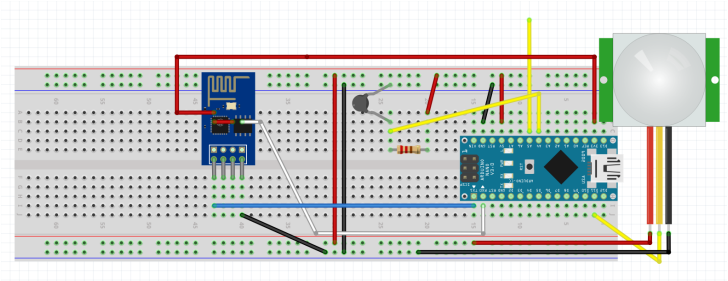


FIGURE 3.1 – Câblage du microcontrôleur

2 Software

D'un point de vue du software, il faut que le microcontrôleur soit capable de lire les informations des différents capteurs mais qu'il puisse aussi générer à partir de cette lecture un type d'information exploitable. Une structure de base de code a été fournie. Elle est axée sur une optimisation de la mémoire, assez réduite, de l'Arduino et elle est complétée par les fonctions nécessaires.

Le PIR n'envoie que deux signaux : présence ou pas de présence. Quant à la valve thermostatique, il s'agit d'un simple signal analogique. En fonction de celui-ci, on connaîtra aisément son degré d'ouverture compris entre 0 et 100 %.

Pour la mesure de la température, le calcul est un peu plus complexe. D'abord, ce qui sort du senseur est une valeur analogique comprise entre 0 et 1024 et qui représente la tension au borne de la thermistance. Elle dépend bien entendu de la température du milieu. La valeur de 4,64V est utilisée dans la relation de conversion car c'est la tension effective que délivre l'Arduino, et pas 5V. Ceci produit un biais sur la mesure de température si elle n'est pas corrigée.

Une fois la tension aux bornes de la thermistance connue, on peut déterminer sa résistance en effectuant un simple calcul de diviseur de tension. C'est en utilisant cette valeur qu'il est possible de déterminer la température de l'environnement grâce à la relation Steinhart-Hart[2] :

$$T(R) = (A_1 + B_1 ln\frac{R}{Ref} + C_1 ln^2\frac{R}{Ref} + D_1 ln^3\frac{R}{Ref})^{-1} \tag{3.1}$$

Les constantes de cette relation sont déterminées grâce à la datasheet de la thermistance (voir Table 3.1[2]). Tous les détails correspondants aux calculs se trouvent dans le code.

PARAMETER FOR DETERMINING NOMINAL RESISTANCE VALUES											
NUMBER	$B_{25/85}$ (K)	NAME	TOL. B (%)	A	B (K)	C (K^2)	D (K^3)	A_1	B_1 (K^{-1})	C_1 (K^{-2})	D_1 (K^{-3})
1	2880	Mat O. with Bn = 2880K	3	- 9.094	2251.74	229098	- 2.744820E+07	3.354016E-03	3.495020E-04	2.095959E-06	4.260615E-07
8	3740	Mat B. with Bn = 3740K	2	- 13.8973	4557.725	- 98275	- 7.522357E+06	3.354016E-03	2.744032E-04	3.666944E-06	1.375492E-07
9	3977	Mat A. with Bn =3977K	0.75	- 14.6337	4791.842	- 115334	- 3.730535E+06	3.354016E-03	2.569850E-04	2.620131E-06	6.383091E-08
10	4090	Mat C. with Bn = 4090K	1.5	- 15.5322	5229.973	- 160451	- 5.414091E+06	3.354016E-03	2.519107E-04	3.510939E-06	1.105179E-07
11	4190	Mat D. with Bn = 4190K	1.5	- 16.0349	5459.339	- 191141	- 3.328322E+06	3.354016E-03	2.460382E-04	3.405377E-06	1.034240E-07
12	4370	Mat E. with Bn = 4370K	2.5	- 16.8717	5759.15	- 194267	- 6.869149E+06	3.354016E-03	2.367720E-04	3.585140E-06	1.255349E-07
13	4570	Mat F. with Bn = 4570K	1.5	- 17.6439	6022.726	- 203157	- 7.183526E+06	3.354016E-03	2.264097E-04	3.278184E-06	1.097628E-07

TABLE 3.1 – Tableau des constantes utilisées de la datasheet du thermistor

Les Figures 3.2 et 3.3 illustrent l'implémentation du microcontrôleur. On y retrouve les deux étapes par lesquelles passent l'Arduino : le `setup` et le `loop`.

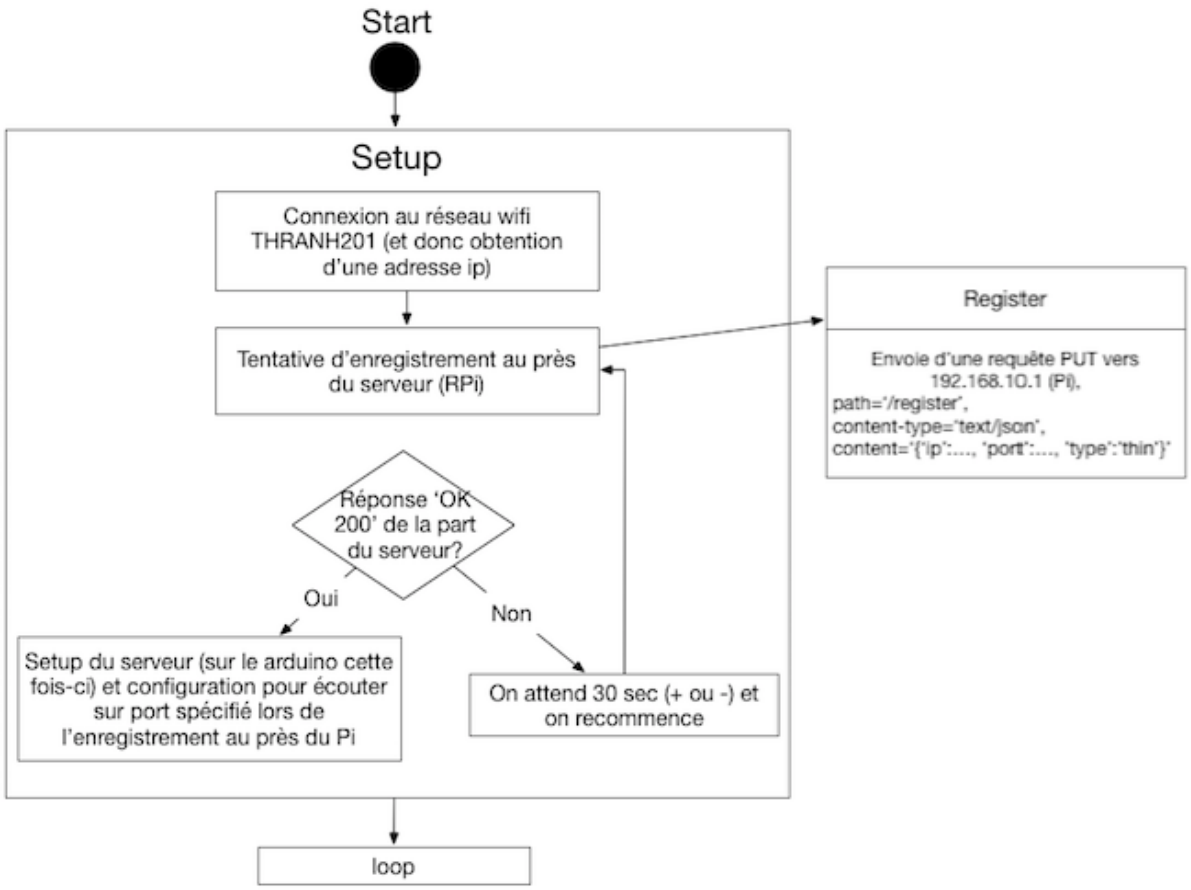


FIGURE 3.2 – Schéma des étapes d'initialisation d'un arduino

Le `setup` se lance à la mise sous tension de l'Arduino et une seule fois. Il est donc logique d'y retrouver la fonction d'initialisation de connexion au Wi-Fi du serveur et celle d'enregistrement.

Le `loop`, quant à lui, est la boucle qui tourne tant que l'Arduino est sous-tension. Il vérifie à intervalle régulier qu'aucune requête ne lui est adressée. Dans le cas échéant, il découpe la requête et compose sa réponse en fonction de celle-ci. Si elle est bien adressée, alors il renvoie l'information demandée en respectant le protocole HTTP. Sinon, il envoie la réponse « 400 » au serveur.

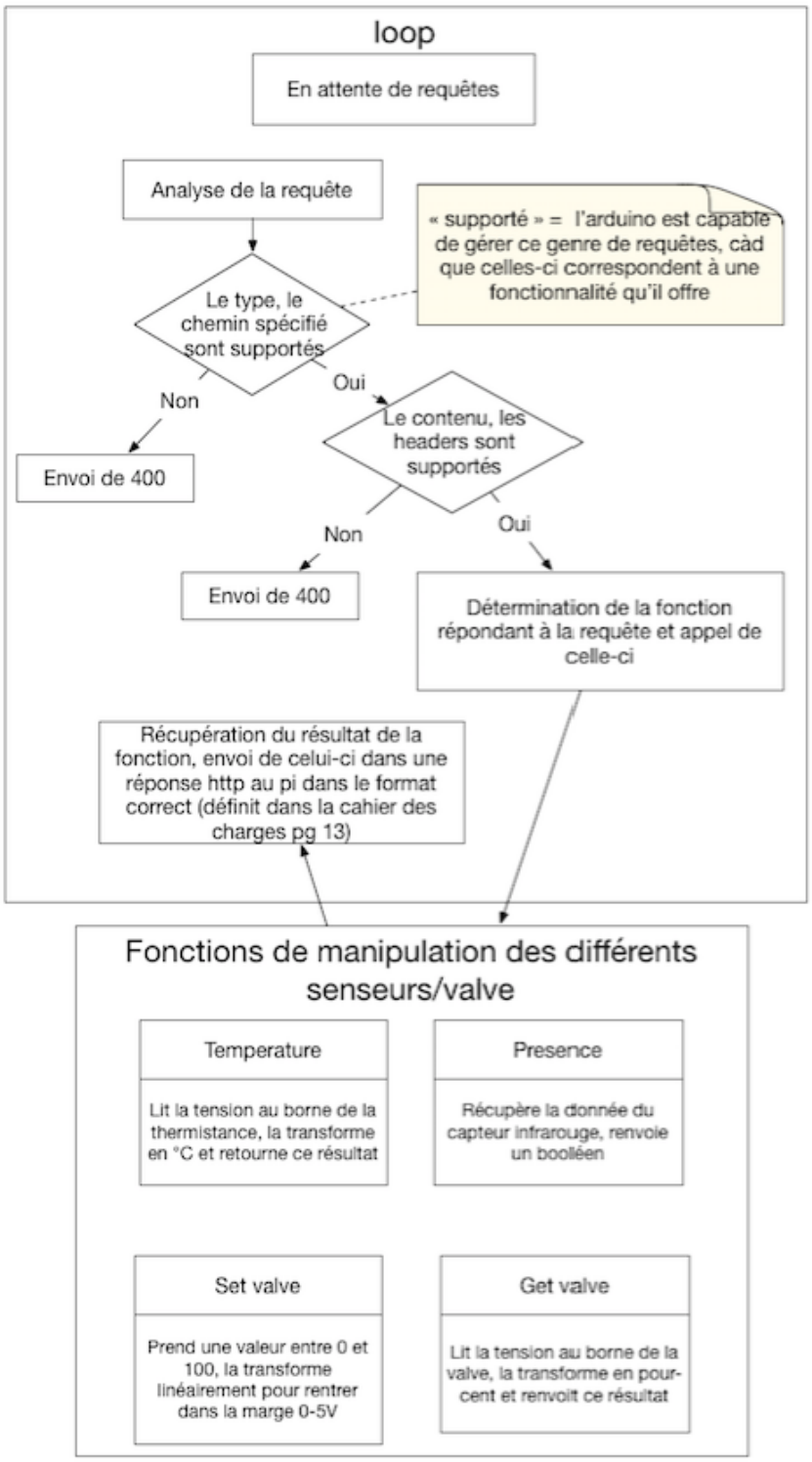


FIGURE 3.3 – Schéma du comportement d'un arduino

4 Thermostat central

1 Serveur

Cette partie est dédiée à la description du module `ThermoServer` dont le but principal est de gérer les Arduino connectés (dès la connexion et pendant la récolte des mesures).

Il a été implémenté de façon à répondre strictement à la description du protocole de communication du cahier des charges. Son fonctionnement global est synthétisé sur la Figure 4.1.

L'implémentation est telle qu'il est très facile de créer et de gérer de nouveaux types de senseur (autres que ceux de type « thin » et « outside » demandés dans le cahier des charges).

1.1 Implémentation

Le protocole de communication s'appuie sur un ensemble de classes qui interagissent entre elles, effectuant chacune une tâche bien précise (décrites brièvement dans le diagramme de classes présenté à la Figure 4.2 [1]. Chacune d'elles sont détaillées dans les sections suivantes.

1.1.1 `ThermoServer`

`ThermoServer` est la classe principale de ce serveur. Son but est de s'occuper simultanément de l'enregistrement des Arduino, de la récupération et de l'enregistrement de leurs mesures dans une base de données et, à l'avenir, de la programmation de l'horaire de contrôle des vannes thermostatiques connectées. Ces fonctions sont également réparties sur les autres classes de ce module ainsi que sur les modules `Data`, `Utils` et ceux qu'il faudra encore développer et qui géreront l'aspect réactif et l'intelligence du thermostat.

Plus techniquement, cette classe est initialisée au démarrage du Raspberry et lance un premier thread consacré au serveur enregistrant les Arduino qui tentent de se connecter (à l'aide de la classe `RequestRouter` décrite dans la partie 3), et un second récoltant périodiquement les mesures renvoyées par les thermostats auxiliaires connectés et les stockant dans une base de données dédiée. A l'avenir, cette classe enverra les requêtes de contrôle des vannes thermostatiques.

1.1.2 `RemoteDevice`

Cette classe sert d'interface de communication pour les Arduino. Une instance de celle-ci est créée à chaque connexion d'un nouveau Remote Device et permettra d'envoyer chaque requête pour récupérer chacune de ses mesures et pour contrôler ce qui est modulable (les vannes par exemple).

De façon plus technique, cette classe est décrite par une adresse IP et un port permettant la communication avec l'Arduino, par un nom identifiant ces mesures et une liste de senseurs qui sont branchés sur ce thermostat auxiliaires. Cette liste permet de créer tout type de Remote Device, composé de plusieurs senseurs, tels qu'une thermistance, un capteur de présence ou même

1. Les diagrammes de classes présentés dans ce rapport s'inspirent des règles proposés par l'UML mais ne sont pas strictement corrects du point de vue de ce langage. Leur but étant principalement documentaire, une trop grande rigueur n'a pas été jugée utile.

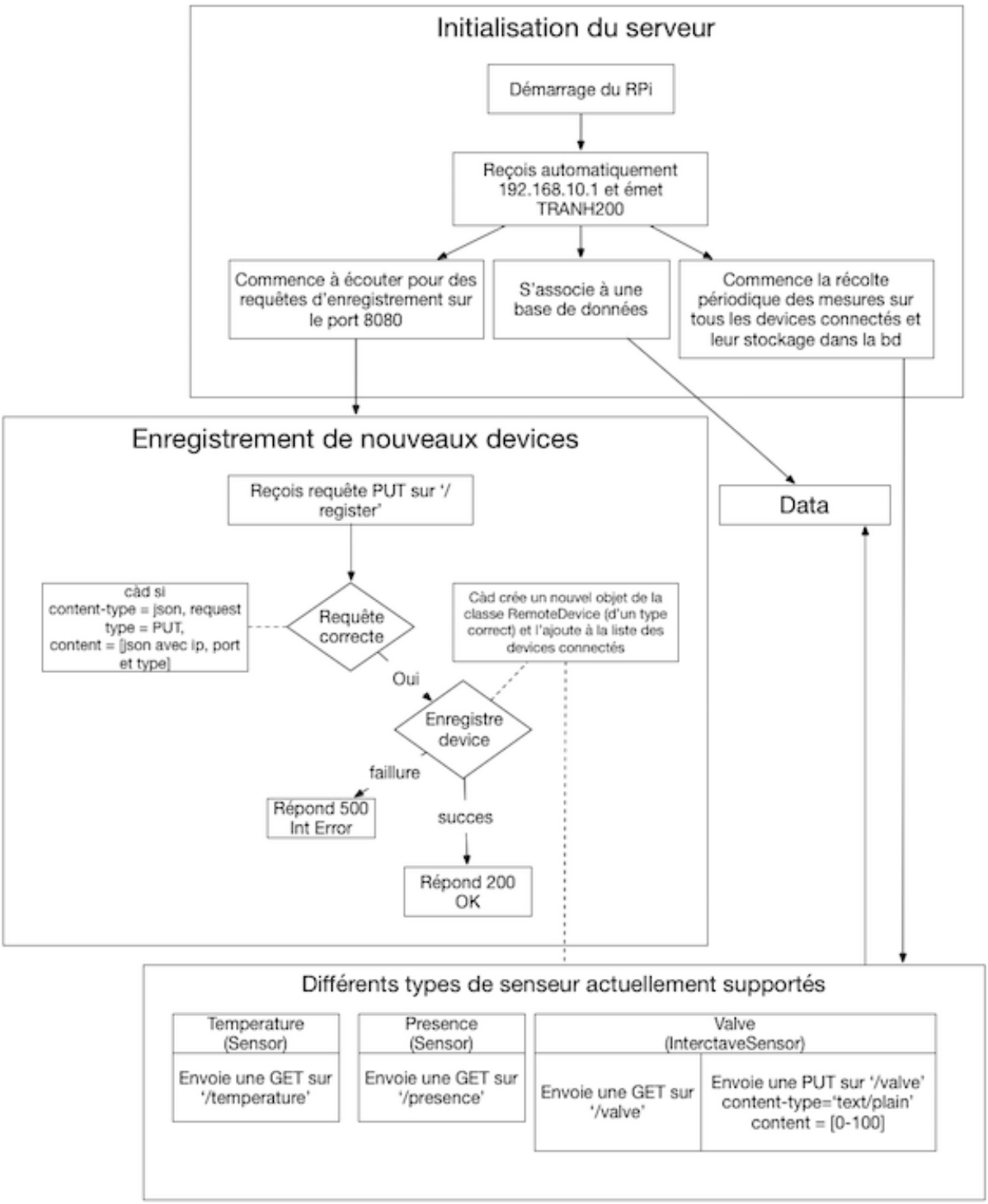


FIGURE 4.1 – Schéma du comportement du serveur

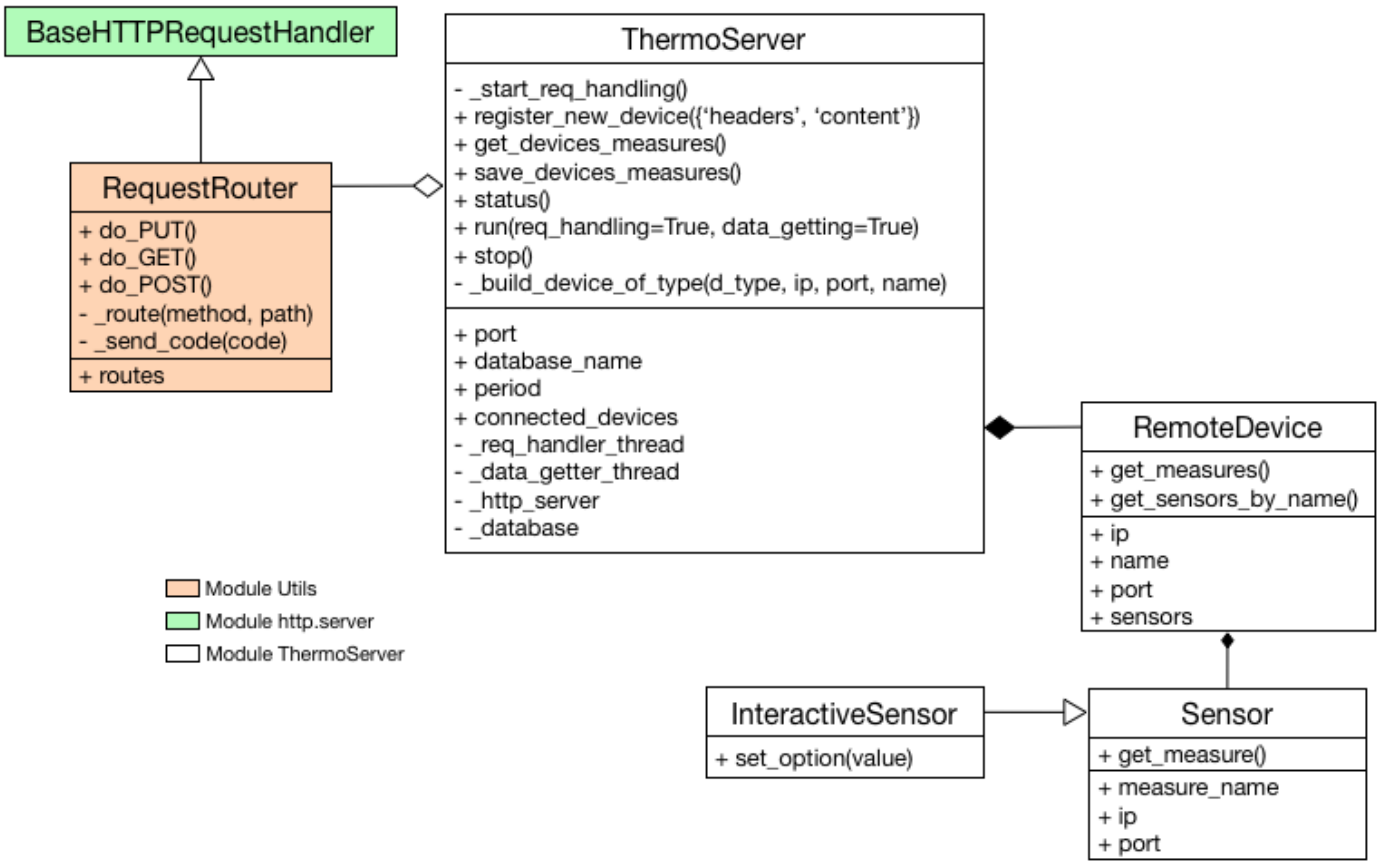


FIGURE 4.2 – Diagramme de classes du module ThermoServer

d'humidité, ou tout autre type de mesures imaginables et compatibles avec le circuit et le code de l'Arduino.

Grâce à cette implémentation modulaire des Remote Devices, il est possible de créer des objets connectés de tout type et pas seulement de type "thin" ou "outside" comme décrit dans le cahier des charges. L'évolution et l'expansion du projet est rendue beaucoup plus facile sans pour autant compliquer l'implémentation de ce qui est demandé.

1.1.3 `Sensor`

Une instance de cette classe permet simplement de récupérer la mesure du senseur correspondant. Par exemple, si l'on imagine un Arduino sur lequel est connecté une thermistance et configuré pour renvoyer la température ambiante en réponse à une requête de type `GET` sur le chemin
`temperature`, cet objet de type `Sensor` permettra d'envoyer cette requête et de récupérer la température.

Dans le cadre du protocole de communication implémenté ici, un objet de type `Sensor` est créé pour chaque senseur physique branché et configuré à chaque Remote Device connecté au serveur.

1.1.4 `InteractiveSensor`

Cette classe hérite de la classe `Sensor` décrite ci-dessus en ajoutant la possibilité d'envoyé une requête de type `PUT` à un senseur pour le contrôler à distance. Typiquement, cette classe permet de récupérer ET de modifier l'état d'ouverture d'une vanne thermostatique.

2 Gestion des données

2.1 Introduction

Afin d'implémenter une analyse intelligente des habitudes de l'utilisateur, un système de stockage de données est nécessaire. Deux options sont envisageables. Soit une structure sous forme de fichier, soit une autre utilisant les bases de données. Des recherches ont été menées pour déterminer quel système serait le plus adéquat. Il en est ressorti que les bases de données correspondent mieux à la problématique. En effet, celles-ci sont plus rapides, plus simples à manipuler et s'adaptent facilement à différentes tailles de jeux de données. De plus, en Python, de nombreuses bibliothèques les concernant existent, et certains types de bases de données, comme par exemple SQLite, y sont intégrés par défaut. Les raisons citées précédemment ont dès lors mené à une décision en faveur des bases de données.

2.2 Choix de la base de données

Il existe différentes bases de données, chacune ayant ses avantages et inconvénients. Une analyse comparative des plus populaires a été réalisée et SQLite a été choisi. La Table 4.1 justifie cette décision.

Caractéristiques	SQLite	MySQL	PostgreSQL
Installation	SQLite est déjà implémenté dans Python.[22]	Installation d'un serveur MySQL nécessaire.[13]	Installation de PostgreSQL nécessaire.[21]
Types de données supportés	NULL, INTEGER, REAL, TEXT, BLOB.[24]	Complet, accepte également des entrées de type « Date ».[12]	Complet, accepte également des entrées de type « Date ».[17]
Stockage de la base de donnée	La base de donnée est condensée en un seul « fichier » stocké directement dans la mémoire (Du Raspberry dans ce cas).[25]	La base de donnée est stockée sur un serveur au quel doivent se connecter les utilisateurs.[11]	La base de donnée est stockée sur un serveur au quel doivent se connecter les utilisateurs.[18]

Table 4.1 – Tableau comparatif des différentes bases de données étudiées

2.3 Implémentation

L'implémentation actuelle du module `Data` consiste en deux classes, `Database` et `Table`, s'occupant des interactions entre une base de données et le serveur pour la première, et entre une table et le serveur pour la seconde. La structure adoptée pour l'instant crée une base de données

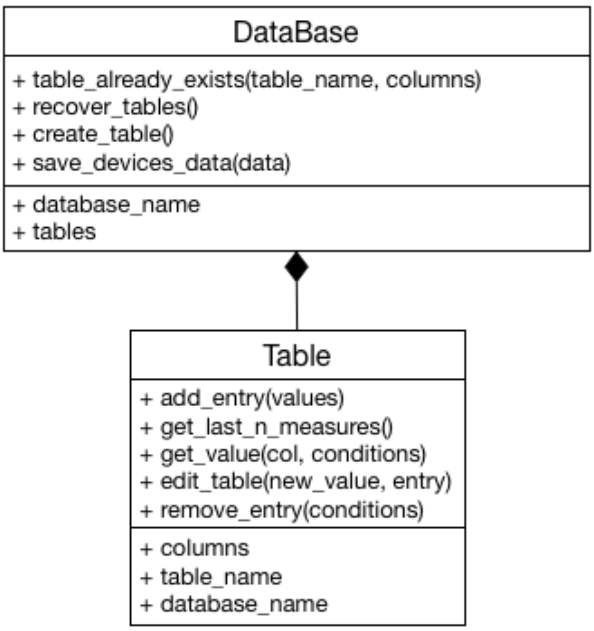


FIGURE 4.3 – Diagramme de classe du module `Data`

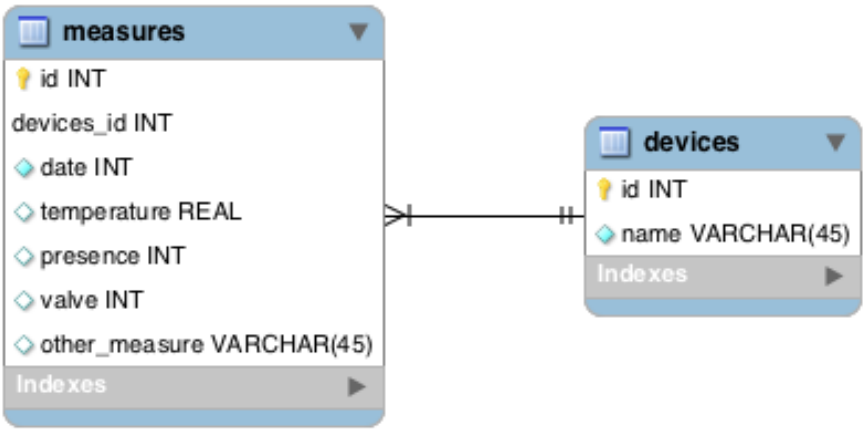


FIGURE 4.4 – Schéma de la structure plus adaptée envisagée

et, à chaque fois qu'un Remote Device se connecte au serveur, une table portant son nom est générée comme le montre la Figure 4.5. Le détail des classes est schématisé à la Figure 4.3 et développé dans les paragraphes suivants.

2.3.1 Classe `Database`

La classe `Database` est l'intermédiaire entre le serveur et la base de données. Elle permet, entre autres, de créer une base de données, de s'y connecter et d'y créer des tables. Elle a été conçue de telle sorte à ne pas créer d'instabilité. En effet si le serveur vient à redémarrer, une instance identique à celle en cours précédemment est créée afin que le code continue à s'exécuter de la manière souhaitée.

2.3.2 Classe `Table`

Quant à elle, la classe `Table` permet d'effectuer des manipulations sur une table, comme par exemple ajouter des entrées ou en supprimer. Elle a principalement été conçue en généralisant la

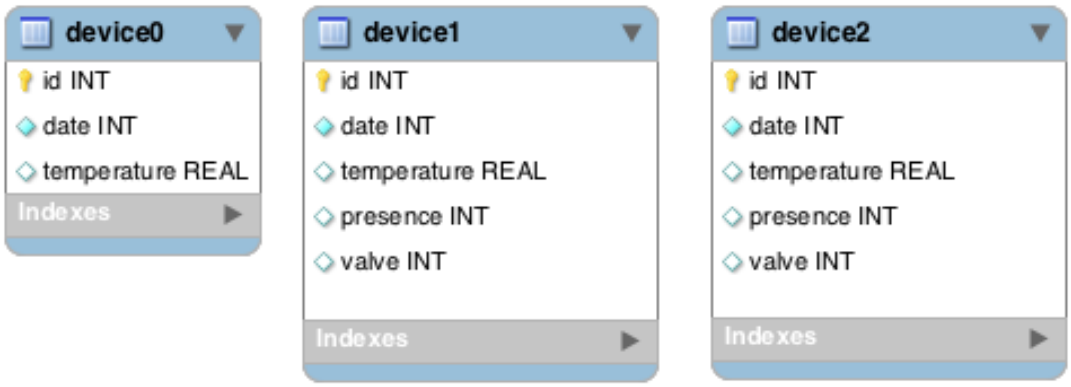


FIGURE 4.5 – Schéma de la structure présentement implémentée de la base de donnée

syntaxe du module `sqlite3` de façon à créer des méthodes d'un plus haut niveau d'abstraction et élégantes d'utilisation.

2.4 Regard critique sur l'implémentation de la base de données

L'implémentation actuelle du module `Data` n'est pas optimale. Cette manière de faire a été décidée tôt dans la conception du serveur. Elle a mené à de longues discussions et semblait être une bonne solution dans la mesure où elle correspondait à la structure globale du code qui consiste une indépendance des différents Remote Devices afin de rendre ce dernier plus flexible. Par la suite, il s'est avéré que ce n'est pas une bonne structure à adopter. En effet, ce n'est pas une façon propre de faire car cela va à l'encontre des règles de bonnes pratiques. Lorsqu'une base de données est utilisée, afin d'optimiser les performances, une seule et unique table est utilisée. De plus, si il y a des problèmes de connexion et que le serveur oublie l'existence des Remote Devices, à chaque fois que ceux-ci se reconnecteront des nouvelles tables seront créées, potentiellement en très grands nombres.

Ces raisons motivent le fait qu'une nouvelle structure soit implémentée dès que possible. Celle-ci s'articule comme l'indique la figure 4.4. Une seule grande table contiendra toutes les données de tous les Remote Device, peu importe leur type, et aura en plus un identifiant qui fera le lien avec le Remote Device d'où proviennent les données correspondantes grâce à une table de jointure. De cette manière, le système gagne en flexibilité car il est possible d'ajouter n'importe quel type de capteur [2] sur les Remote Devices, ce qui est le résultat recherché.

3 Module `Utils`

Ce module ne contient (pour le moment) que deux classes : `RepeatingTimer` et `RequestRouter`.

Il a pour but de contenir des classes ou plus généralement des outils qui sont complètement indépendants du projet (mais utiles à celui-ci) et qui pourraient donc être facilement utilisables pour un quelconque autre projet. Outre le fait que la séparation des classes génériques leur permet d'être réutilisées dans d'autre projet, se pliant ainsi aux règles de bonnes pratiques, la construction d'un module séparé contenant ces classes permet d'alléger les autres modules et de les rendre plus compréhensible.

Ces deux classes sont donc construites de façon générique et accomplissent des taches élémentaires.

2. Dans une perspective de développement ultérieur du projet, par exemple ajouter des capteurs d'humidité,...

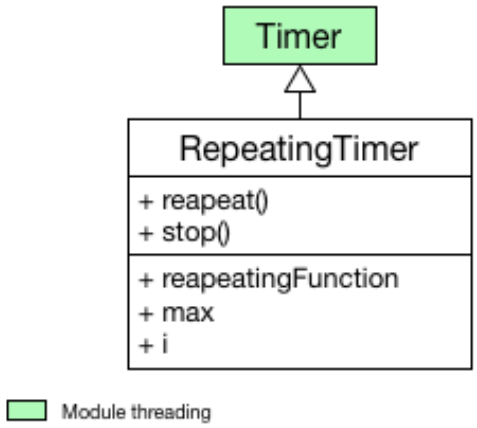


FIGURE 4.6 – Diagramme de classe pour RepeatingTimer

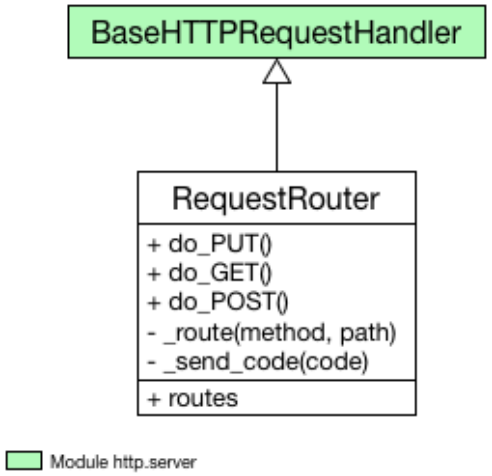


FIGURE 4.7 – Diagramme de classe pour RequestRouter

3.1 RepeatingTimer

La seule fonction de cette classe (héritant de la classe `Timer` du module `threading` de python) est d'appeler une fonction spécifiée à intervalle régulier.

Elle est utile, dans ce projet, à la collecte des mesures régulière du serveur sur les remote devices connectés.

3.2 RequestRouter

Cette classe, héritant de `BaseHTTPRequestHandler` fournie dans le module `http.server` de python 3, a pour but d'écouter sur un port spécifié pour d'éventuelles requêtes et de les "router" vers un callback spécifié en fonction de la méthode http et du chemin de la requête.

Elle est totalement générique puisque n'importe quelle requête peut déclencher n'importe quelle fonction (pour peu qu'elle soit configurée correctement) et peut-être utilisée dans n'importe quel cas nécessitant un serveur.

Dans le cadre spécifique de ce projet, cette classe est utile à la réception et au traitement des requêtes d'enregistrement des remote devices.

4 Debugage

Un système de logging est implémenté dans tous les modules pour garder une trace de chaque éventuelle erreur, disfonctionnement, anomalie lors du fonctionnement du serveur. Le dépannage sera alors simplifié lors d'une utilisation réelle et le debugage est plus rapide lors du développement dès lors que l'origine des erreurs est plus facilement identifiable. Il est ainsi possible dès le lancement du serveur d'indiquer quels types (niveau de gravité) de messages doivent être affichés et/ou stockés dans un fichier journal.

5 Fonctionnement de groupe

Afin d'atteindre les objectifs du projet le plus rapidement et le plus efficacement possible, il était important d'établir une bonne organisation du travail en groupe afin de maximiser la productivité du groupe et de répartir la quantité de tâches à effectuer par chacun de manière équitable. Quelques outils ont donc été mis en place dès la première semaine dans cette optique.

Pour commencer, le plus important était de définir comment les informations transiteraient. Un groupe privé sur Facebook a été choisi pour la communication informelle en dehors des réunions, l'avantage étant de pouvoir conserver un historique de ces conversations. Le transfert des documents, codes, parties de rapport,... s'effectue quant à lui via une Dropbox qui permet d'archiver et de partager tous ces fichiers de façon simple et efficace. Il a également été décidé que le rapport serait rédigé en LaTeXsur « Overleaf [1] », cette plate-forme permettant la modification en temps réel du document par tous les membres de l'équipe.

La fréquence des réunions fut fixée au rythme de deux par semaines. Une avec le tuteur du projet afin de le tenir au courant des avancées réalisées lors de l'autre réunion qui concerne uniquement les membres du groupe. Lors de ces rencontres hebdomadaires, chacun se doit de comprendre les avancée exposées par les autres membres du groupe afin d'obtenir des connaissances homogènes dans tous les domaines. Cette fréquence permet de ne pas oublier de travailler chaque semaine sur le projet. Il arrive que des réunions supplémentaires soient organisées en équipe lorsque le besoin s'en fait sentir ou qu'une deadline approche.

Ces réunions se tiennent, sauf cas de force majeure, à la Bibliothèque des Sciences et Techniques de l'ULB, principalement pour des raisons de proximité. Durant ces réunions, un membre est désigné et joue le rôle de secrétaire et d'animateur. La fusion de ces deux rôles a été décidée en raison de la taille du groupe. La tâche de de cette personne est d'envoyer l'ordre du jour aux autres membres au moins 24 heures avant la réunion, de l'animer ensuite et d'en rédiger le procès-verbal par après. Ce poste est attribué à une autre personne toutes les deux semaines sur base volontaire afin de ne surcharger personne de travail.

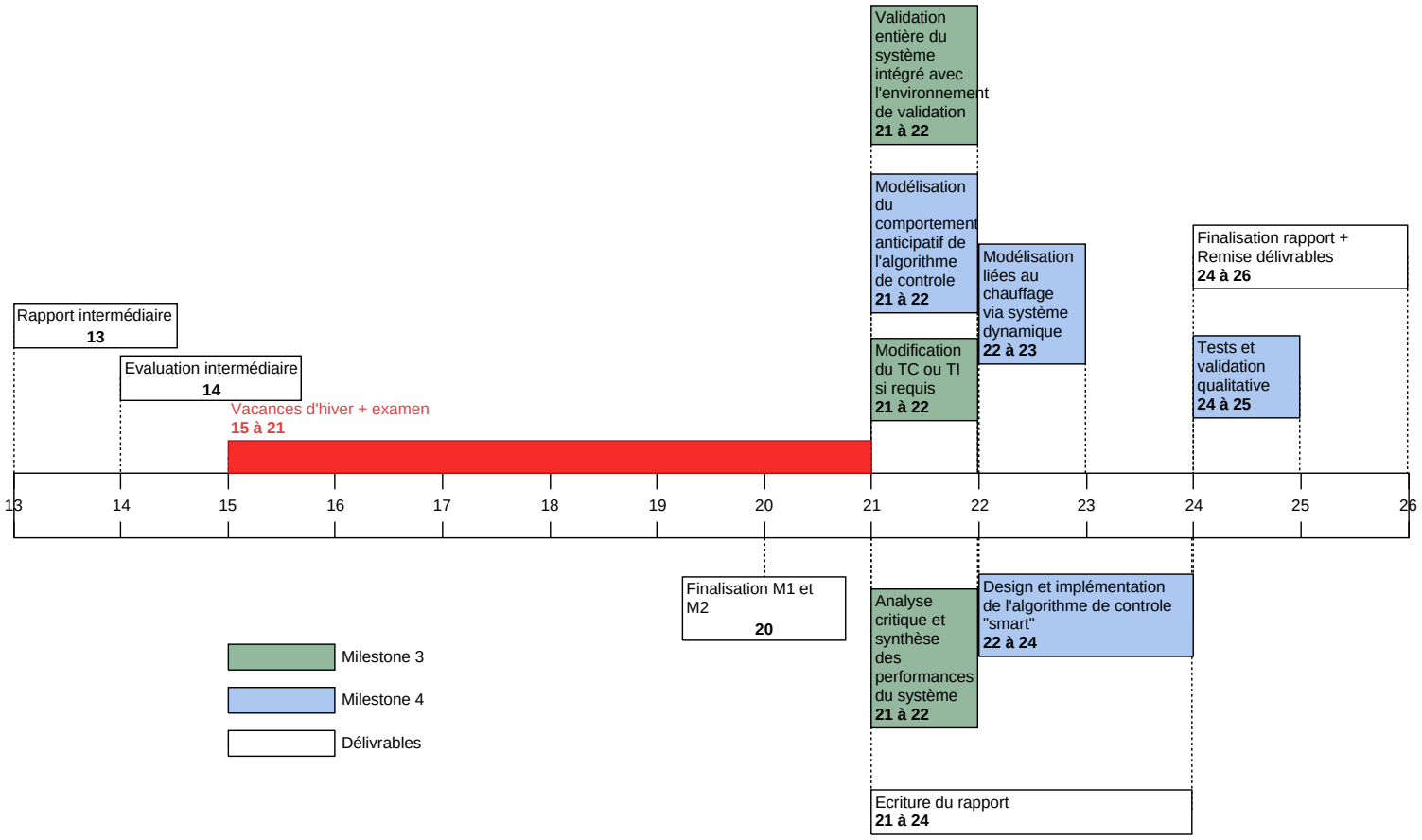
1. `https://www.overleaf.com/`

6 Perspectives d'évolution

En ce qui concerne de la suite du projet, il a été décidé de maintenir la même méthode de travail,à savoir en planifiant les tâches en se basant sur le cahier des charges. Le premier quadrimestre a été essentiellement consacré à la familiarisation avec les différents outils. Raspberry Pi, programmation orientée objet, Arduino, base de données étant autant de concept qu'il a fallu apprendre à utiliser et maîtriser. C'est pourquoi les milestones une et deux ont été abordé jusqu'à présent.

Suivant cette logique, c'est donc bien le développement des milestones trois et quatre qui sera d'actualité au second quadrimestre (voir Figure 6.1), bien que celui-ci soit beaucoup plus court. Il y aura une réflexion qui précédera la mise en place d'un algorithme intelligent suivant les exigences du cahier de charges. Ce planning est potentiellement sujet à des modifications ultérieures mais il reste une bonne base de travail. Il est volontairement "sur-estimé", notamment en ce qui concerne la finalisation du rapport final. Ceci a pour but de pallier aux imprévus comme ceux rencontrés au premier quadrimestre.

FIGURE 6.1 – Planning du second quadrimestre



7 Conclusion

La conception et la visualisation des demandes du projet était la première étape clé du processus de fabrication du thermostat intelligent. L'utilisation du diagramme présenté en Annexe A était le moyen le plus efficace de se représenter l'objectif final. La réalisation du câblage des différents composants a été assez rapidement effectué, tout comme l'implémentation de l'Arduino, bien que parfois celle-ci ait été retardée suite à un manque de matériel.

L'autre partie abordée en parallèle concernait le Raspberry. La création d'un serveur était nécessaire, l'implémentation de celui-ci est telle qu'il permet une plus grande flexibilité que celle exigée par le cahier des charges. En effet, on peut y connecter n'importe quel type de Remote Device. Comme exigé dans le cahier des charges, les communications respectent le protocole HTTP.

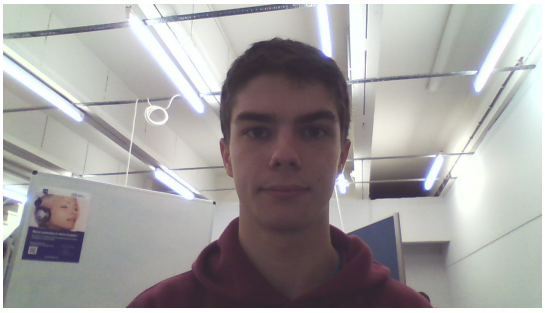
Un stockage des données était nécessaire afin d'implémenter l'analyse intelligente. Une étude a été menée afin de choisir entre les différentes façon de faire et SQLite a finalement été retenu pour sa légèreté, sa simplicité et sa présence par défaut dans Python. L'implémentation actuelle ne respecte pas les conventions et sera prochainement revue. Elle avait été structurée de cette manière car elle semblait plus souple.

Le fonctionnement du groupe tel qu'il est à présent semble ne souffrir d'aucun problème en particulier. Il sera donc conservé sauf imprévu lors du deuxième quadrimestre.

Le travail restant à fournir est assez conséquent, les milestones une et deux sont presque achevées et les milestones trois et quatre devront être entamées dès le début du second quadrimestre afin de respecter les deadlines. Celui-ci sera entamé par la finalisation des algorithmes de contrôle réactifs. Une fois cette étape achevée, le code intelligent sera implémenté.

8 Membres du groupe

Amzur Soufiane
000328725
soufiane.amz@gmail.com



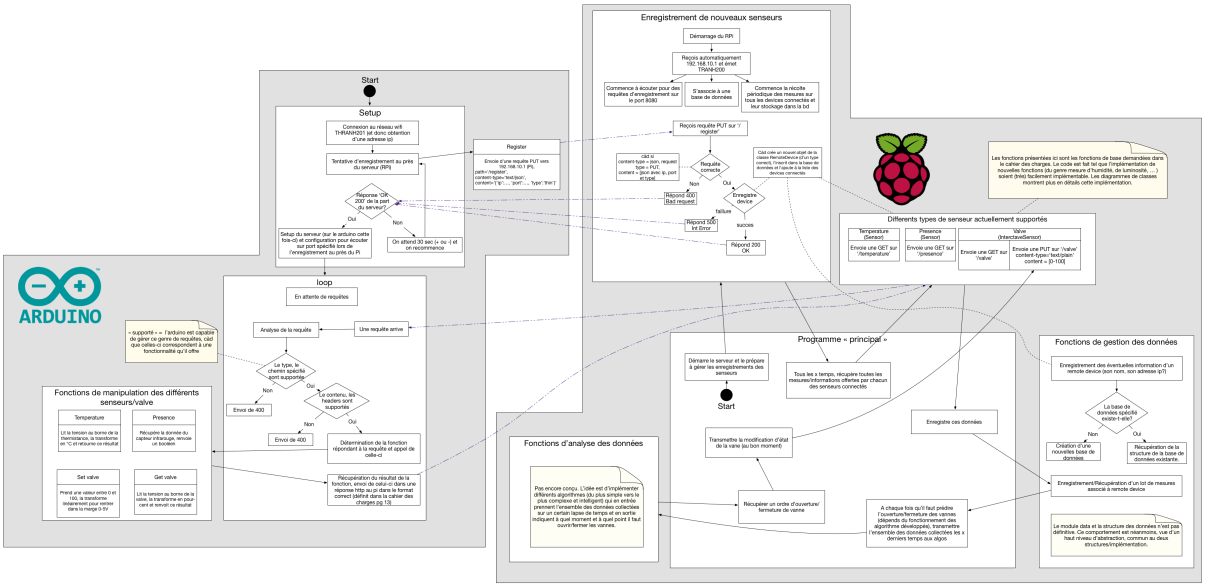
Daubry Wilson
000408780
daubry.wilson@gmail.com

Sercu Stephane
000408643
ssercu@ulb.ac.be

Verstraeten Denis
000401967
denverst@ulb.ac.be

A Diagramme de comportement global

Comme expliqué dans la partie Conception, ce diagramme a été créé principalement pour faciliter le travail de groupe. Il offre une vue très synthétique du comportement de l'ensemble du système ainsi que de la façon dont les différentes parties communiquent et interagissent.



B Conventions de programmation

Dans le but de garder un code propre et maintenable par tous les participants du projet sur le long terme, des conventions et règles de bonne pratique ont été choisies pour être respectées.

1 Code Python

- les conventions proposées dans le cours de Thierry Massart [10]. Notamment pour les docstring, le nommage des classes, des fonctions et des variables,
- la langue choisie pour le code est l'anglais, pour la simple et bonne raison que c'est la langue universelle des développeurs. Le code n'en est que plus cohérent par rapport aux fonctions prédéfinies.
- L'orienté objet a été un choix facile : le code n'en est que plus compréhensible, structuré, modulaire et évolutif.
- Une attention particulière a été portée à la documentation de chaque méthode/fonction/classe pour éviter une perte de temps à la compréhension et l'analyse du code par les autres membres du groupe et/où par l'auteur lui même dans l'avenir. De cette façon, l'utilisation de ces fonctions est simplifié et ne nécessite que la lecture de la documentation associée pour savoir de quelle façon il faut les appeler et quel type de paramètre il faut leur passer.
- Les deux modules principaux (`ThermoServer` et `Data`) sont accompagnés de diagrammes de classe permettant leur documentation. Ils offrent une vue globale, simplifiée et résumée de leur structure. Cette pratique sera maintenue pour les prochains modules principalement pour son pouvoir documentaire.

2 Code Arduino

Cette partie a été programmée directement dans le code de base fournis, le style de programmation a donc été calqué sur celui-ci.

Bibliographie

[1] Arduino. Arduino - Website. https://www.arduino.cc/.

[2] Vishay BCcomponents. Ntc thermistors, radial leaded, standard precision - datasheet. https://www.vishay.com.

[3] Elec-Freaks. Specification of dyp-me003. http://elecfreaks.com/store/download/datasheet/sensor/DYP-ME003/Specification.pdf.

[4] The Python Software Foundation. HTTP servers — Python 3.5.0 documentation. https://docs.python.org/3/library/http.server.html.

[5] The Python Software Foundation. Thread-based parallelism — Python 3.5.0 documentation. https://docs.python.org/3.5/library/threading.html.

[6] Communauté francophone d'utilisateurs d'Ubuntu. Http : hypertext transfer protocol. http://doc.ubuntu-fr.org/tutoriel/console_ligne_de_commande.

[7] Daniel Jackson. Http : hypertext transfer protocol. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-170-software-studio-spring-2013/lecture-notes/MIT6_170S13_07-http-prtcol.pdf.

[8] Jeff. Raspberry Pi Thermostat Part 1 : System Overview. http://www.nooganeer.com/his/projects/homeautomation/raspberry-pi-thermostat-part-1-overview/.

[9] Araki M. Control systems, robotics, and automation. http://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf.

[10] Thierry Massart. Info-h-100 informatique - release 3.4.0 (2015) chapitre 15. http://www.ulb.ac.be/di/verif/tmassart/Informatique/latex/SyllabusPython32.pdf.

[11] MySQL. Chapter 1 general information. http://dev.mysql.com/doc/refman/5.7/en/introduction.html.

[12] MySQL. Chapter 11 data types. http://dev.mysql.com/doc/refman/5.7/en/data-types.html.

[13] MySQL. Chapter 4 connector/python installation. https://dev.mysql.com/doc/connector-python/en/connector-python-installation.html.

[14] Raspberry Pi. Raspberry pi - website. https://www.raspberrypi.org.

[15] Raspberry Pi. Raspberry pi 2, model b. http://www.adafruit.com/pdfs/raspberrypi2modelb.pdf.

[16] Pavan Podila. Http : The protocol every web developer must know. http://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177.

[17] PostgreSQL. Chapter 8. data types. http://www.postgresql.org/docs/9.3/static/datatype.html.

[18] PostgreSQL. Postgresql 9.4.5 documentation. http://www.postgresql.org/docs/9.4/static/tutorial-arch.html.

[19] R.M. Price. RMP Lecture Notes. http://facstaff.cbu.edu/rprice/lectures/contalgo.html.

[20] Maxim Integrated Products. Arduino nano (v3.0) - user manual. http://www.jameco.com/jameco/products/prodds/2127970.pdf.

[21] PyGreSQL. Pygresql – postgresql module for python. http://www.pygresql.org/.

[22] Python. Db-api 2.0 interface for sqlite databases. https://docs.python.org/2/library/sqlite3.html.

[23] A. Kenneth Reitz. Requests : HTTP for Humans — Requests 2.8.1 documentation. http://docs.python-requests.org/en/latest/.

[24] SQLite. Datatypes in sqlite version 3. https://www.sqlite.org/datatype3.html.

[25] SQLite. Distinctive features of sqlite. https://www.sqlite.org/different.html.

[26] Espressif Systems IOT Team. Esp8266ex datasheet. https://www.adafruit.com/images/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf.

[27] Unknown. Http commands and response codes. http://papa.det.uvigo.es/~theiere/cursos/Curso_WWW/codes.html.

[28] Unknown. Linux bash shell cheat sheet. http://cli.learncodethehardway.org/bash_cheat_sheet.pdf.

[29] Unknown. Types of Controllers Proportional Integral and Derivative Controllers Electrical4u. http://www.electrical4u.com/types-of-controllers-proportional-integral-derivative-controllers/.