

Académie de LYON

BTS SN IR

Brevet de technicien supérieur Systèmes Numériques

Option Informatique et Réseaux

Session 2022

Lycée ORT Lyon



Epreuve E62 – Projet technique

Piséo Etuve

Candidats :

LALICHE Lucas

PERRET Léandre

KORCHIA Maxime

Entreprise :

Piséo



Figure 1 : Logo de Piséo

1 Sommaire

Table des matières

1	Sommaire	2
2	Remerciements	5
3	Présentation de l'entreprise	6
4	Besoins de l'entreprise	9
4.1	Reformulations des besoins de Piséo	9
4.2	Propositions de solutions	10
5	UML	11
5.1	Diagramme de déploiement	11
5.2	Diagramme de cas d'utilisation	12
5.3	Diagramme de séquence	13
6	Technologies utilisées	14
6.1	Logiciel de communication : Discord	14
6.2	Logiciel de programmation : Visual Studio Code	14
6.3	Logiciel de versionning de code : GitLab	15
6.4	Langage de programmation : Python	15
6.5	Langage de programmation web : PHP	16
6.6	Formatage des données : JSON	17
6.7	API REST	17
7	Présentation des tâches	18
8	Candidat 2 : Lucas LALICHE	19
	Introduction et présentation générale du rôle	19
8.1	Tableau des tâches	19
	Planning prévisionnel individuel	20
	Etape 1 : Réalisation de la maquette	20
8.1.1	Eléments de réalisation	20
8.1.2	Bilan	21
8.2	Etape 2 – Réalisation de l'IHM	22
8.2.1	Objectifs	22
8.2.2	Eléments de réalisation	22
8.2.3	Création des différents onglets	23
8.2.4	Selecteur de fichier	23
8.2.5	Ajout des boutons liés aux étapes	24

8.2.6	Création du tableau des étapes	25
8.2.7	Jauge Température	25
8.2.8	Lancer expérience	25
8.2.9	Partie configuration	26
8.2.10	Enregistrer et importer	26
8.2.11	Graphique.....	27
8.2.12	Bilan.....	27
8.3	Etape 3 – Utilisation du mock	28
8.3.1	Objectifs	28
8.3.2	Éléments de réalisation	29
8.3.3	Bilan	29
8.4	Etape 4 – Intégration.....	29
8.4.1	Objectifs	29
8.4.2	Éléments de réalisation	30
8.4.3	Bilan	30
8.5	Etape 5 – Reste à faire.....	31
8.5.1	Finir la fonction modifier	31
8.5.2	Afficher la valeur de la jauge en C°	31
8.5.3	Finir la fonction lancer expérience	31
8.5.4	Lier les données du tableau avec le graphique	31
8.6	Bilan.....	31
9	Candidat 2 : PERRET Léandre : Back-end Python, bibliothèque de classe et API.....	32
9.1	Introduction et présentation générale du rôle	32
9.2	Planning prévisionnel individuel	32
	32
9.3	Tableau des tâches.....	33
9.4	Code de la bibliothèque de classe.....	34
9.4.1	Introduction et objectif.....	34
9.4.2	Éléments de réalisations	35
9.4.3	Conclusion.....	44
9.5	API-REST sous Django.....	45
9.5.1	Introduction et objectif.....	45
9.5.2	Éléments de réalisations	45
9.5.3	Conclusion.....	55

10	Candidat 3 : KORCHIA Maxime, PILOTAGE DE L'ETUVE (Python).....	56
10.1.1	Planning prévisionnel individuel	56
10.1.2	Tableau des tâches	57
10.1.3	Diagramme de classe	57
10.2	Tâche 1 – Procédure d'installation de UltraViewer	58
10.2.1	Eléments de réalisation	58
10.2.2	Bilan	60
10.3	Tâche 2 - Utilisation de APT-COM	61
10.3.1	Introduction.....	61
10.3.2	Lecture et compréhension de la documentation de APT-COM	61
10.3.3	Création de machine sur APT-COM.....	62
10.3.4	Tests sur la machine créée	65
10.3.5	APT-COM chez PISEO	66
10.3.6	Bilan	68
10.4	Tâche 3 - Wireshark	68
10.4.1	Introduction.....	68
10.4.2	Eléments de réalisation	68
10.4.3	Analyse de trames	69
10.5	Tâche 4 – Programmation en python	72
10.5.1	Eléments de réalisation	72
10.5.2	Bilan	76
11	Tableau des figures	77

2 Remerciements

Nous tenions à remercier toutes les personnes ayant contribuées de près comme de loin à la conception de ce projet.

Nous remercions tout d'abord Piséo de nous avoir fait confiance avec ce projet ainsi que notre professeur et tuteur de projet, monsieur Luc Gilot, qui nous a aidés tout au long du projet et crus en nous.

Mais aussi à l'ensemble du corps enseignant nous ayant formés durant ces deux dernières années.

Pour finir, nous souhaitons remercier l'ensemble de nos camarades de classe ayant partagé avec nous, ces deux dernières années au quotidien.

3 Présentation de l'entreprise

Créé fin 2011, Piséo est un centre d'innovation spécialisé dans l'optique photonique, allant de la conception à de la réalisation et caractérisation de systèmes d'illumination passant par de la détection et d'imagerie pour tout secteur d'activités. Piséo dispose d'équipements de pointe afin de réaliser des essais de performance photométrique et radiométrique (LED, capteurs d'image, illuminateurs...). L'entreprise s'inscrit dans le secteur d'activité des analyses, essais et inspections techniques. Ainsi, leur mission est d'accompagner les clients dans leurs démarches d'innovation et d'optimisation grâce à l'intégration de fonctions optiques et photoniques avancées. Son effectif est compris entre 10 et 19 salariés.

Photonique : c'est la science, ainsi que la technologie de générer, contrôler et détecter des photons, qui sont des particules de lumière. Cette science permet de nombreuses innovations dans le domaine des nouvelles technologies. La photonique se trouve derrière les différentes et nombreuses technologies de la vie quotidienne, telles que les smartphones, les PC portables, internet, les équipements médicaux, la technologie de la lumière ... L'amélioration de ces objets et utilitaires du quotidien est principalement due à l'utilisation de la photonique.

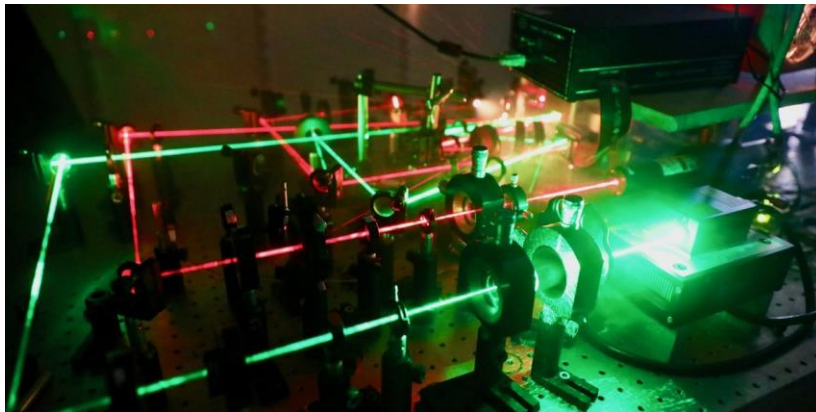


Figure 2 : La photonique

Les locaux de l'entreprise PISEO se situent 4 Rue de l'Arsenal, à Vénissieux précisément (69200)

(Voir figure 3 et 4).

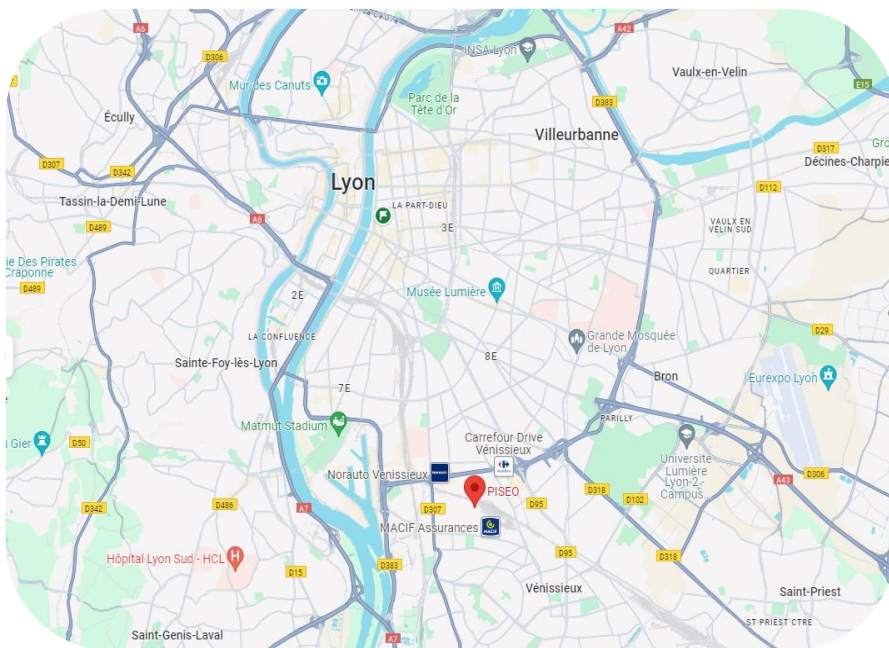


Figure 3 : Laboratoire Piséo



Figure 4 : Bureau Piséo

Secteurs d'activité de l'entreprise :

A. Secteur de l'imagerie

Le laboratoire Piséo œuvre sur le secteur de l'imagerie afin d'améliorer les performances des systèmes d'imagerie et augmenter leurs capacités de différenciation d'objets. Il est également crucial d'éclairer correctement la scène dans le champ de vision de la caméra, que ce soit dans la lumière visible ou non. Ou dans la sphère infrarouge.

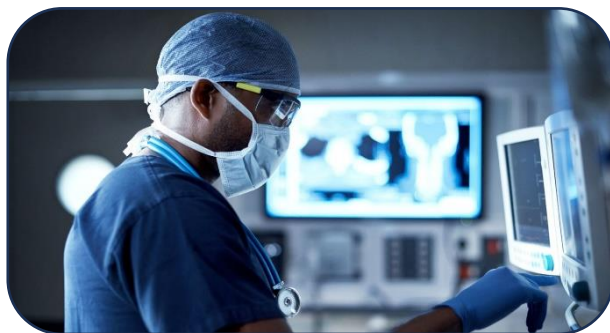


Figure 5 : Contexte Piséo

B. L'industrie de la santé et des soins corporels

Les appareils sont toujours de plus en plus performants et nombreux, c'est ainsi que Piséo travaille sur différents appareils lumineux dans le milieu pharmaceutique, tels que : épilateur à lumière pulsée, endoscope, polymérisation dentaire, etc. La désinfection par rayonnement UV connaît un regain d'intérêt, et les LED UV-C semblent prometteurs.

Analyse fonctionnelle



Figure 6 : Contexte Piséo

C. L'activité de défense militaire

Ils sont aussi exploités sur les domaines de la défense militaire. Ces domaines utilisent des systèmes de surveillance et de détection optique qui nécessitent une conception particulièrement soignée. PISEO a créé des systèmes qui utilisent les technologies photoniques les plus récentes dans les domaines de la lumière visible et infra-rouge.



Figure 7 : Contexte Piséo

D. L'industrie automobile

L'industrie automobile fait partie des nombreuses parts du marché des luminaires en effet, les guides de lumière combinés à des LED sont particulièrement prisés pour créer des effets de design attrayants dans l'habitacle et à l'extérieur des véhicules.



Figure 8 : Contexte Piséo

E. L'industrie générale de l'éclairage LED

La LED est largement utilisée dans l'éclairage général. De plus, la grande variété de formes, de puissance et de spectre lumineux des LEDS les rend adaptables à toutes les applications d'éclairage, y compris l'horticulture ou les caves à vin.

4 Besoins de l'entreprise

4.1 Reformulations des besoins de Piséo

Piséo faisant des expérimentations au sein d'une étuve qui consiste d'une chambre confinée permettant différentes études avec chauffage et capteur à haute précision. Pour Piséo, ça leurs sert principalement à faire leurs expérimentations sur différents types de luminaire à des températures spécifiques afin de notamment garantir l'usage

de ces luminaires dans un milieu prédéfini. Le problème que Piséo rencontre est que quand un technicien lance une chauffe de l'étuve pour faire une mesure sur un appareil, le technicien doit attendre que l'étuve atteigne sa température cible pour ensuite lancer son test. Nous avons été missionnés par Piséo pour créer une application qui permettra de contrôler cette étuve à distance afin de mener une expérimentation qui aura été définie au sein de l'application leur permettant d'effectuer leurs tests. L'expérimentation prend forme de différentes températures consignées à un temps donnée pouvant être allié à un programme s'exécutant lorsque la consigne est respectée. Permettant l'automatisation du chauffage de l'étuve, de mesure et du lancement de tests en série.

4.2 Propositions de solutions

C'est ainsi que trois solutions sont proposées afin de répondre au problème de Piséo. La première d'entre-elles consiste à une solution Web se basant sur du PHP permettant un site web de définir une expérimentation qui manipulera par la suite l'étuve. Cette solution est déployée sur un micro-ordinateur Raspberry qui hébergera le site via un serveur installé en son sein. La seconde solution qui a pour identique but à celle d'avant, elle est développée telle une application de bureau se basant sur du Python, permettant à son tour aussi de mener une expérimentation. La dernière solution proposée est une API permettant par le biais de simple requête URL, de manipuler la température de l'étuve et de lire la température de celle-ci.

5 UML

5.1 Diagramme de déploiement

Différents schémas de type UML furent rédigés afin de mieux appréhender de façon graphique la solution proposée par notre groupe. Tout d'abord le diagramme de déploiement (Voir figure 9).

Notre solution proposée fonctionnera par le biais de trois différentes machines. La première d'entre elle est l'étuve

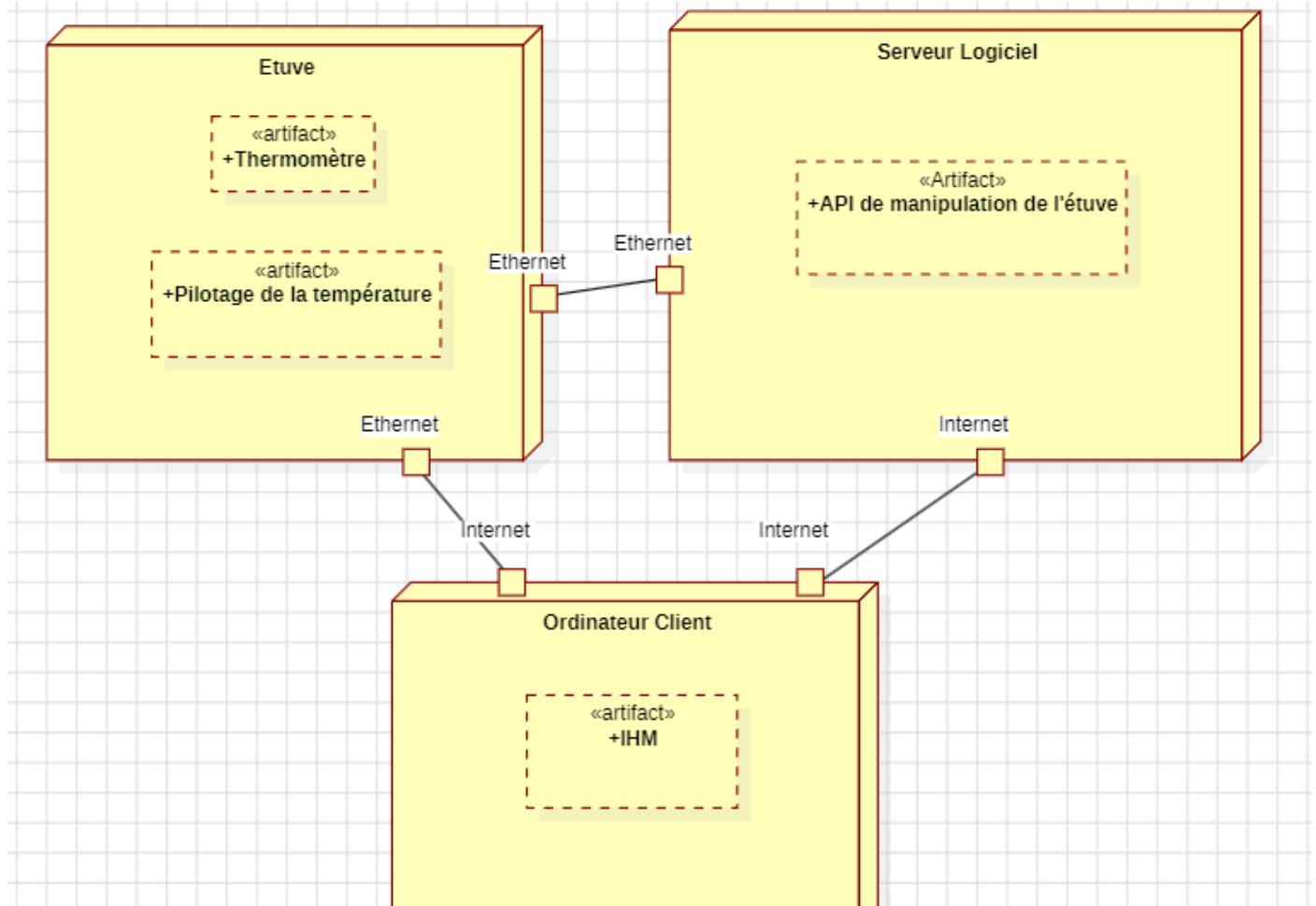


Figure 9 : Diagramme de déploiement

que possède Piséo. Dans cette étuve, se trouvent deux dispositifs, d'abord un thermomètre nous permettant de pouvoir visualiser la température au sein de l'étuve puis un pilote pour la température nous permettant de pouvoir contrôler celle-ci. La seconde machine et celle du serveur logiciel, servant à mettre l'API de manipulation de l'étuve, celle-ci est accessible depuis le net. Afin de fonctionner, le serveur est relié en Ethernet à l'étuve lui permettant de récupérer les informations. La dernière machine utilisée est celle de l'ordinateur client qui possède en son sein une IHM lui permettant de récupérer et d'envoyer des données à l'étuve. Mais il peut aussi accéder via le web à l'API mise en place sur le serveur logiciel.

5.2 Diagramme de cas d'utilisation

Le second schéma rédigé est celui du diagramme de cas d'utilisation (Voir figure 10 et 11). Présentant les différentes interactions que peuvent effectuées les différents acteurs.

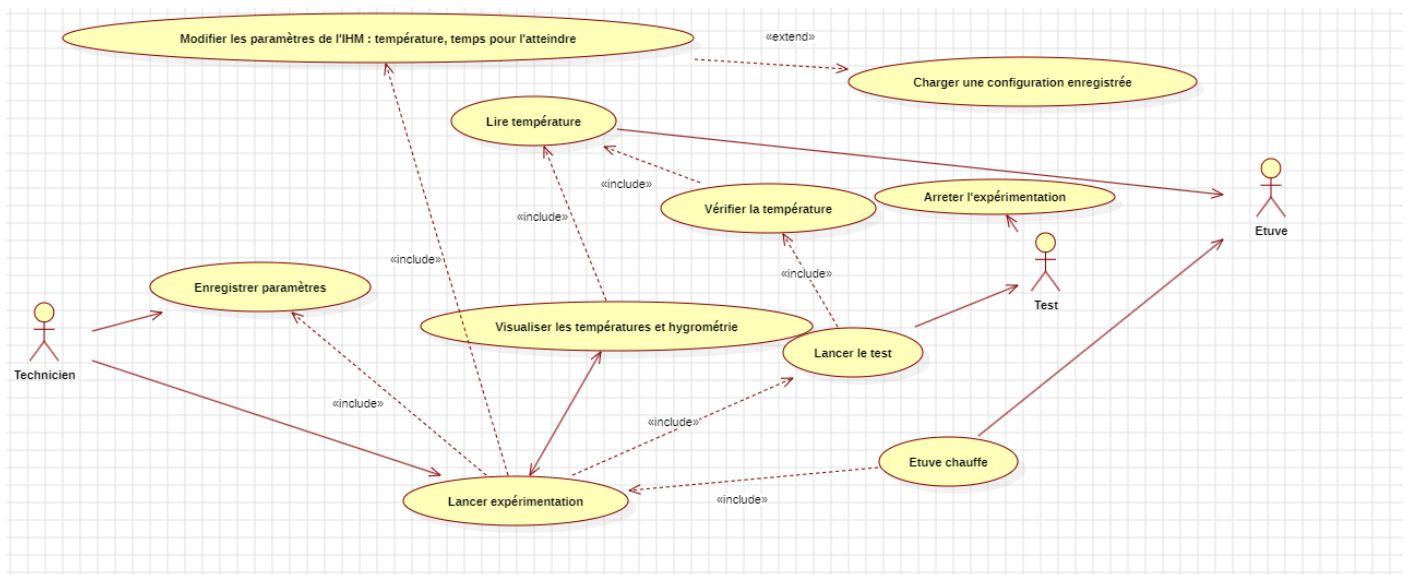


Figure 10: Diagramme de cas d'utilisation

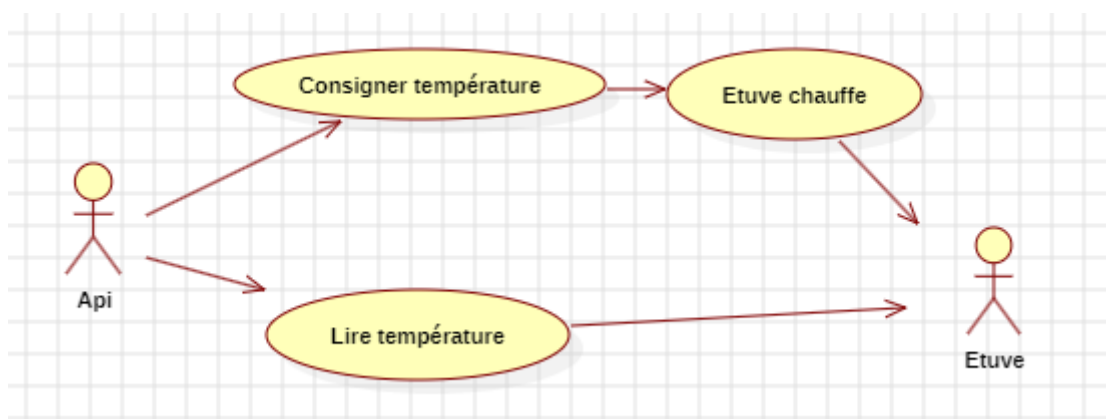


Figure 11: Diagramme de cas d'utilisation (API)

Le technicien peut enregistrer les paramètres. Il peut lancer l'expérimentation, en lançant l'expérimentation il peut tout d'abord choisir la température ainsi que le temps que cela va prendre pour l'atteindre (ou choisir de charger une configuration enregistrée) puis il peut visualiser les températures, mais il sera obligé de demander les informations à l'acteur « étuve ». Le technicien peut décider de lancer le test en demandant à l'acteur « test » après le lancement du test, il pourra vérifier les températures ainsi que demander à l'acteur « test » d'arrêter le test. Le lancement de l'expérimentation provoque une chauffe de l'étuve qui sera lancée par l'acteur « étuve ». Quant à elle l'API peut consigner une température qui pour la suite mettra l'étuve à chauffer à cette température durant la durée consignée puis elle peut aussi lire la température présente au sein de l'étuve.

5.3 Diagramme de séquence

Le troisième schéma rédigé est celui du diagramme de séquence (Voir figure 12). Présentant un exemple de possibilité d'utilisation de notre solution.

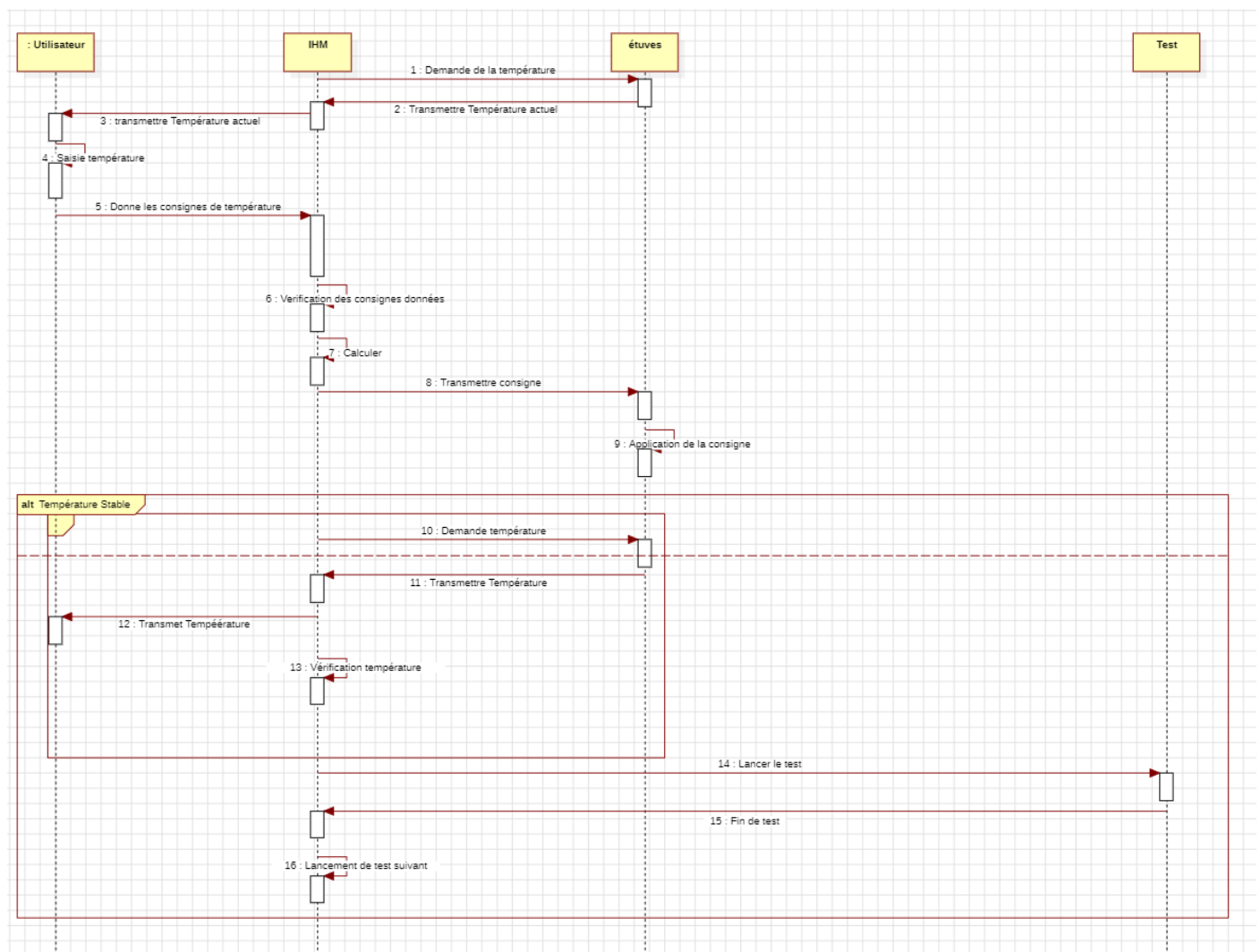


Figure 12 : Diagramme de Séquence

Ici, lors de son lancement, l'IHM demande à l'étuve sa température qui alors lui transmet ce qui permet à l'IHM de l'afficher à l'utilisateur qui pourra alors saisir la température qu'il souhaite. Une fois que l'utilisateur a consigné toutes les températures qu'il souhaitait, l'IHM vérifie ces données-là puis commence à calculer les consignes qu'elle va transmettre à l'étuve puis transmet ces dites consignes qui seront par la suite appliquées par l'étuve. Suite à cela, l'IHM va répéter en boucle, demande de la température, réception de la température, montre la température puis la vérifie. Si la température a atteint celle souhaitée alors l'on sort de la boucle en exécutant le test. Suite à ça, nous attendons que le test se finisse afin de lancer le test suivant. Ceci se répète jusqu'à tous les tests soient lancés.

6 Technologies utilisées

6.1 Logiciel de communication : Discord

Afin de communiquer les informations au sein du groupe, nous avons utilisé comme moyen de communication Discord, application web de messagerie instantanée et de communication vocale en réseau (VoIP), sortie le 13 mai 2015. Elle nous a permis notamment de créer un groupe afin d'y pouvoir dialoguer avec un maximum de 10 membres.



Figure 13 : Logo Discord

6.2 Logiciel de programmation : Visual Studio Code

Logiciel développé par Microsoft, sortie le 14 novembre 2015, Visual Studio Code est un éditeur Open Source, connu principalement, car il est présent sur plusieurs OS (Windows, Linux, MacOS) et est gratuit. Il permet notamment de coder une grande variété de langage de programmation tel que HTML, CSS, Python, PHP, JavaScript, etc. L'application permet aussi de créer ou de télécharger des extensions ajoutant des fonctionnalités supplémentaires tout en laissant l'utilisateur un choix de personnalisation au niveau des thèmes d'interfaces ou des raccourcis claviers personnalisables. Cet éditeur de code facilite grandement le travail des développeurs via une prise en charge du débogage, une mise en évidence de la syntaxe et bien d'autres facultés qui ne cessent de grandir via une communauté active développant de nouvelles fonctionnalités tous les jours.

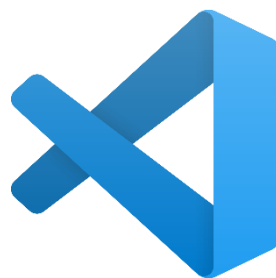


Figure 14 : Logo Visual Code Studio

6.3 Logiciel de versionning de code : GitLab

GitLab, logiciel libre d'accès qui se base sur Git permettant aux développeurs de pouvoir partager et stocker l'avancée de leurs fichiers au sein d'un même projet. Par le biais de branche, il permet à un ensemble de personnes de travailler sur un projet tout en fournissant une protection afin d'éviter que du code non-utilisable se retrouve en production. Il permet aussi de partager un projet à des personnes spécifiques ou alors au monde entier. Sortie le 13 octobre 2011, il est utilisé notamment par des grandes entreprises telles que la NASA, Sony ou SpaceX.



Figure 15 : Logo Gitlab

6.4 Langage de programmation : Python

Langage de programmation de haut niveau et polyvalent, Python se différencie via une accentuation sur la lisibilité du code par un système d'indentation significative permettant aux lignes de programmations d'être lues via des indentations. Il est d'ailleurs interprété ce qui signifie qu'il n'y a pas de compilation lors de l'exécution d'un programme ce qui facilite le débogage du code. Le majeur problème de Python tient de sa lenteur, qui est balancée par la facilité de programmation de compréhension du code.



Figure 16 : Logo Python

6.5 Langage de programmation web : PHP

PHP, qui signifie "PHP: Hypertext Preprocessor", est un langage de programmation open source largement utilisée pour le développement web côté serveur. Il a été créé par Rasmus Lerdorf en 1994 et est devenu l'un des langages les plus populaires pour la création de sites web dynamiques et interactifs. Ce langage est principalement utilisé côté serveur, ce qui signifie que le code PHP est exécuté sur le serveur web pour générer du contenu dynamique avant de l'envoyer au navigateur du client. Il est couramment utilisé pour traiter les données des formulaires HTML, récupérer les données soumises par l'utilisateur, et effectuer des opérations telles que la validation et le stockage dans une base de données. Ce langage offre des fonctionnalités pour se connecter à différentes bases de données, ce qui facilite le stockage et la récupération de données pour les applications web. Avec un large écosystème de bibliothèques et de frameworks tels que Laravel, Symfony et CodeIgniter, PHP simplifie le développement web. Distribué sous la licence PHP en tant que logiciel open source, son utilisation est gratuite et le code source est accessible. Au fil des années, PHP a évolué et est aujourd'hui utilisé pour développer différents types d'applications, y compris des applications en ligne, des API, et bien plus encore, ce qui en fait l'un des langages de programmation les plus utilisés sur le web.



Figure 17 : Logo de PHP

6.6 Formatage des données : JSON

Le JSON (JavaScript Object Notation) est un format léger d'échange de données largement utilisé dans le développement de logiciels et sur le web. Il est basé sur une syntaxe simple et lisible pour représenter des objets de données structurées. Voici une présentation des principales caractéristiques du JSON :

Format Léger et Lisible : Le JSON utilise une syntaxe simple composée de paire clé-valeur, ce qui le rend facilement lisible par les humains et les machines.

Structuré en Objets : Les données JSON sont généralement organisées en objets, qui sont des ensembles non ordonnés de paires clé-valeur encapsulées entre des accolades {}.

Types de Données Supportés : JSON prend en charge plusieurs types de données, notamment les chaînes de caractères, les nombres, les booléens, les tableaux, les objets et les valeurs nulles.

Analogie avec les Objets JavaScript : JSON est étroitement lié à JavaScript, et sa syntaxe ressemble à la syntaxe des objets JavaScript, ce qui facilite son utilisation avec ce langage de programmation.

Utilisation Polyvalente : Le JSON est largement utilisé pour échanger des données entre serveurs et clients dans les applications web, stocker des configurations et des paramètres, et même comme format de fichiers pour la sérialisation de données.

Facilité d'Analyse et de Génération : La simplicité de la syntaxe JSON le rend facile à analyser et à générer à l'aide de la plupart des langages de programmation, avec de nombreuses bibliothèques disponibles pour manipuler des données JSON. En résumé, le JSON est un format de données flexible, léger et largement utilisé, idéal pour l'échange de données entre applications et services sur le web, ainsi que pour la manipulation de données dans les programmes informatiques.

6.7 API REST

Les API REST sont un diminutif de l'anglais « Representational State Transfer » qui se définit comme pour toutes API par un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels. Ces API-là diffèrent par le fait qu'elles sont basées sur le protocole http utilisant des méthodes comme GET ou POST. Les réponses fournies sont généralement fournies au format JSON ou XML avec des communications dites « stateless » entre le client et le serveur permettant que les informations clientes ne soient stockées et que les requêtes GET sont indépendantes des autres.

7 Présentation des tâches

Faire les diagramme UML	Lucas L. / Léandre P. / Maxime K.
Faire maquette IHM	Lucas L.
Structure de l'IHM	Lucas L.
Rajout classes de l'E2	Lucas L.
Importer et exporter	Lucas L.
API sous Django	Léandre P.
Code de la bibliothèque de classes	Léandre P.
Procédure d'installation de UltraViewer	Maxime K.
Utilisation de APT-COM	Maxime K.
Utilisation de Wireshark	Maxime K.
Création de trames pour piloter l'étuve	Maxime K.
Mise en commun	Lucas L. / Léandre P. / Maxime K.

8 Candidat 2 : Lucas LALICHE

Introduction et présentation générale du rôle

Mon rôle au sein du projet est la création de l'IHM (Interface Homme Machine) développé en python. L'IHM est une interface utilisateur permettant de connecter une personne à une machine, à un système ou à un appareil. Dans notre cas on souhaite que l'utilisateur puisse communiquer et interagir avec une étuve. L'IHM doit contenir des spécifications détaillées listées ci-dessous :

- Donner des consignes pour atteindre une température cible (plusieurs dans le cas d'une série de mesure)
- Définir un intervalle de plus ou moins une certaine température pendant un temps donné pour définir que la température est stable (intervalle, durée)
- Définir l'action à mener quand la température est atteinte (mesure que souhaite effectuer Piseo quand la température consigne est atteinte) ;
- Démarrer ou arrêter la mesure et récupérer ses résultats (l'exemple sera la mesure de la température interne, qui est le travail du groupe « Ambiance ») ;
- Afficher l'état actuel de l'essai (heure de début, durée, température actuelle, seuils atteints, tests effectués et leurs résultats éventuellement ...) ;
- Mémoriser les configurations saisies, pour pouvoir les retrouver lors d'une session ultérieure.

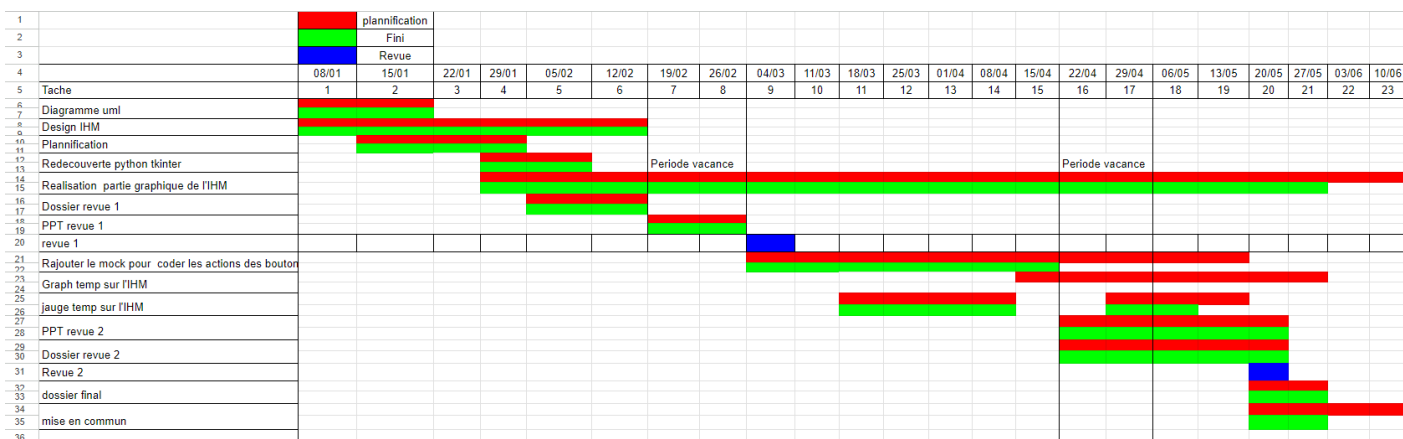
Ensuite je vais utiliser la bibliothèque de classes faite par Léandre (étudiant E2), dans un premier temps je n'aurai que le mock (le squelette), cela permet aux développeurs de tester le fonctionnement d'une section de leur programme sans avoir besoin d'utiliser les véritables composants ou objets.

8.1 Tableau des tâches

Tache	Description
Faire les diagramme UML	Diagramme UML de séquence et cas d'utilisation
Maquette	Réalisation de la maquette
Structure de l'IHM	Réaliser la maquette en python avec tkinter
Rajout classes de l'E2	Utilisation de la bibliothèque de classes de E2 (mock dans un premier temps)
Importer et exporter	Système de sauvegarde et réutilisation de la version antérieur
Mise en commun	Rassemble toutes les solutions du groupe(partie python) au sein d'un même programme (pilotage de l'étuve, API, IHM)

Après avoir planifié les différentes tâches, nous abordons désormais la phase suivante du processus de développement.

Planning prévisionnel individuel



Lors du début du projet j'ai fait un planning prévisionnel qui est représenté en rouge, ensuite en vert il y a le planning réel et en bleu ce sont les revues.

Légende du planning :

Bleu = Revue de projet.

Rouge = La planification des taches dans le temps.

Vert = Ce qui est fait.

Etape 1 : Réalisation de la maquette

Dans cette étape plutôt importante pour développer une interface graphique, l'accent est mis sur la création d'une maquette préliminaire de l'IHM, marquant ainsi le passage du concept à la réalisation avec l'aide de Edouardo charger de faire l'IHM.

Cette phase vise à résoudre le problème posé par la nécessité de traduire les besoins de l'entreprise en une IHM fonctionnelle. Les résultats attendus comprennent la visualisation interactive du flux de l'application, la validation des choix de conception avec l'entreprise et les autres membres du projet, et la création d'un patron solide mais qui peut être modifié au besoin pour guider le développement de cette IHM.

8.1.1 Eléments de réalisation

Description de Figma :

Figma est un outil de conception d'interface utilisateur en ligne qui offre une approche collaborative. Il permet une compréhension approfondie des besoins de l'entreprise en facilitant la création de la maquette, la structuration

visuelle de l'IHM, et le prototypage interactif. Figma sert également de plateforme pour la collecte des retours des parties prenantes, favorisant ainsi des itérations rapides et précises.



En utilisant Figma, j'ai pu traduire les exigences identifiées en éléments visuels concrets. La création de wireframes a permis de mettre en évidence la disposition générale des éléments de l'IHM, tandis que le prototypage interactif a simulé le flux de l'application. La collaboration en temps réel a été facilitée, et les retours recueillis ont alimenté des itérations successives pour parvenir à une maquette optimale.

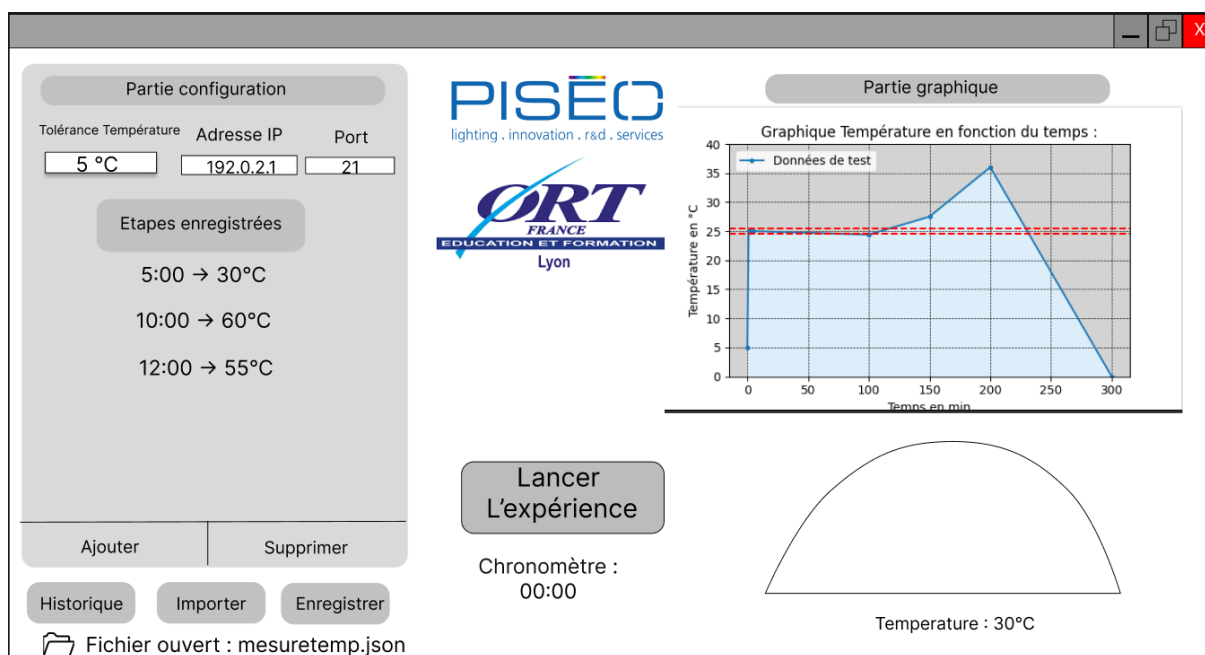


Figure 18 : Prototype IHM

Figma a joué un rôle clé dans la création d'un patron solide pour le développement de l'IHM. Cette approche interactive et collaborative garantit la cohérence entre la conception initiale et la réalisation finale de l'application.

8.1.2 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Réalisation de la maquette	Réaliser la maquette pour voir si elle répond a la demande du client	Le prototype répond à la demande du client	Conforme	OK

Je n'ai pas eu de problème lors de cette mission qui consistait à faire une maquette. J'ai aimé travailler sur le design de l'IHM car j'ai dû faire appel à ma créativité pour concevoir quelque chose de fonctionnel et beau à la fois.

À la suite de cette mission j'ai pu commencer à développer mon IHM en respectant la maquette.

8.2 Etape 2 – Réalisation de l'IHM

L'objectif de cette tâche est de concrétiser l'Interface Homme-Machine (IHM) faite auparavant pour la développer en python tout offrant une expérience utilisateur intuitive et fonctionnelle.

8.2.1 Objectifs

La tâche vise à créer une IHM complète, intégrant des fonctionnalités d'historique ou encore de visualisation de graphiques de température et d'hygrométrie, ainsi que des fonctions de chronométrage pour les expériences.

8.2.2 Éléments de réalisation

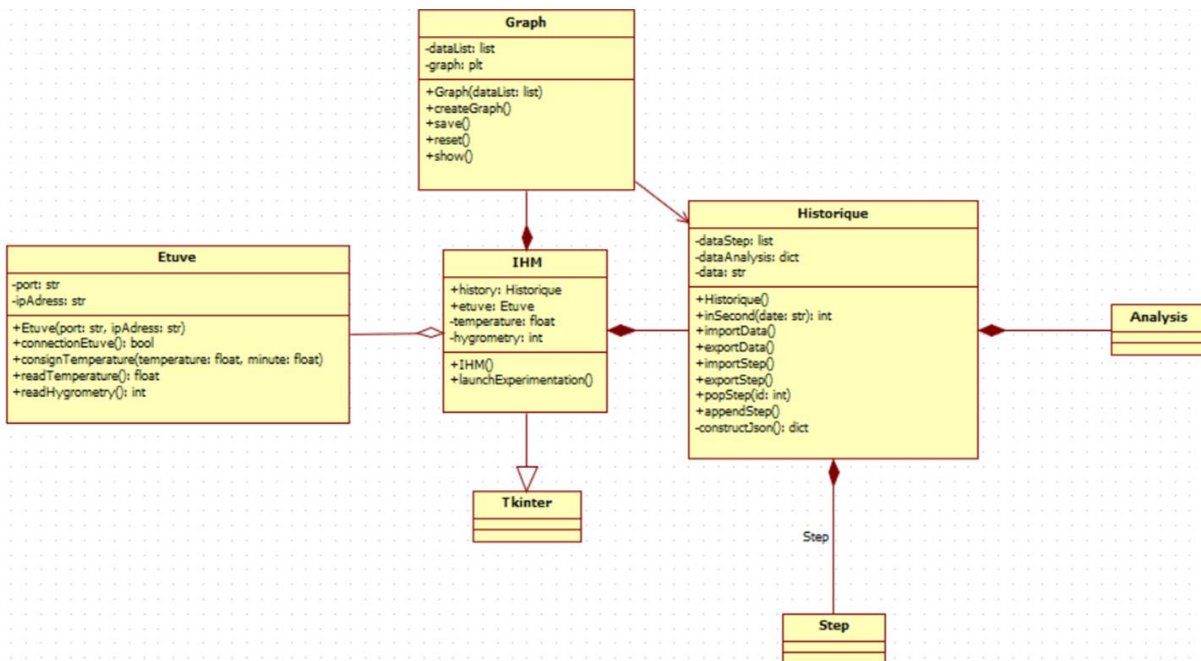
La réalisation d'une interface homme-machine (IHM) en Python sur Visual Studio Code implique plusieurs éléments essentiels pour garantir une conception efficace et conviviale. Tout d'abord, l'utilisation d'une bibliothèques telle que Tkinter offre des outils puissants pour créer des interfaces graphiques interactives. Cette bibliothèque fournit des composants d'interface pré-conçus tels que des boutons, des champs de texte et des fenêtres, facilitant ainsi le processus de développement.

Ensuite, Visual Studio Code offre des fonctionnalités avancées d'édition de code qui simplifient la création et la gestion de l'IHM. Des fonctionnalités telles que l'autocomplétion, le débogage intégré et les extensions tierces

permettent aux développeurs de travailler de manière plus productive et de déboguer efficacement leur interface utilisateur.

Enfin, l'intégration de Visual Studio Code avec des outils de gestion de version tels que Git facilite la collaboration sur des projets d'IHM. Les développeurs peuvent facilement partager leur code, suivre les modifications et résoudre les conflits, ce qui contribue à un développement plus fluide et à une meilleure qualité de l'IHM finale.

8.2.3 Création des différents onglets



Pour commencer la reproduction de la maquette j'ai créé 2 onglets "Accueil" et "Historique" en Tkinter qui est une bibliothèque Python pour le développement d'interfaces graphiques, plusieurs étapes sont nécessaires. Tout d'abord on crée un widget Notebook, qui est utilisé pour afficher plusieurs onglets. Chaque onglet est créé en tant que cadre distinct à l'intérieur du Notebook. Pour l'onglet "Accueil", on peut y placer les widgets correspondant au contenu de cette page. De même, pour l'onglet "Historique", on ajoute les widgets qui affichent les données historiques. Enfin, on ajoute chaque cadre (onglet) au Notebook et on spécifie leurs titres respectifs. Cela permet à l'utilisateur de naviguer entre les onglets en toute simplicité.

Ensuite j'ai créé 2 canvas un qui correspond à la page accueil et l'autre à la page historique. Chaque canvas fait 1500*800 se qui représente la taille de la page principale.

```
first_canvas = Canvas(first_tab, width=1500, height=800, background="#fefefe")
first_canvas.pack()

second_canvas = Canvas(second_tab, width=1500, height=800, background="#fefefe")
second_canvas.pack()
```

8.2.4 Selecteur de fichier

Pour créer un bouton qui ouvre un explorateur de fichiers en Tkinter, une bibliothèque Python pour le développement d'interfaces graphiques, plusieurs étapes sont nécessaires. Tout d'abord, il faut importer le module Tkinter et créer une fenêtre principale. Ensuite, on définit une fonction qui sera exécutée lorsque le bouton est cliqué. Dans cette fonction, on utilise la boîte de dialogue de fichier de Tkinter pour ouvrir

l'explorateur de fichiers. Cette boîte de dialogue permet à l'utilisateur de naviguer dans son système de fichiers et de sélectionner un fichier ou un dossier. Une fois le fichier sélectionné, on peut récupérer le chemin d'accès à ce fichier pour effectuer les opérations nécessaires, comme l'ouverture du fichier pour le traitement ultérieur. En associant cette fonction au clic du bouton, on crée ainsi un mécanisme simple et intuitif pour permettre à l'utilisateur de naviguer dans son système de fichiers et de sélectionner les fichiers désirés. En suivant ces étapes, les développeurs peuvent intégrer facilement cette fonctionnalité dans leurs applications Tkinter pour une expérience utilisateur améliorée.

```
def explo():
    file_path = filedialog.askopenfilename()
    print(file_path)
    NameFile = ttk.Label(
        root,
        text='Fichier ouvert : ' + file_path)

    imageDossier = Image.open("Partie_lucas/IHM/Image/dossier2.jpg")
    imageDossierRedimensionner = imageDossier.resize((50,50))
    image_tkC = ImageTk.PhotoImage(imageDossierRedimensionner)
    ExplorateurBt = ttk.Button(root, image=image_tkC, command=explo)
    ExplorateurBt.pack(pady=20)
```

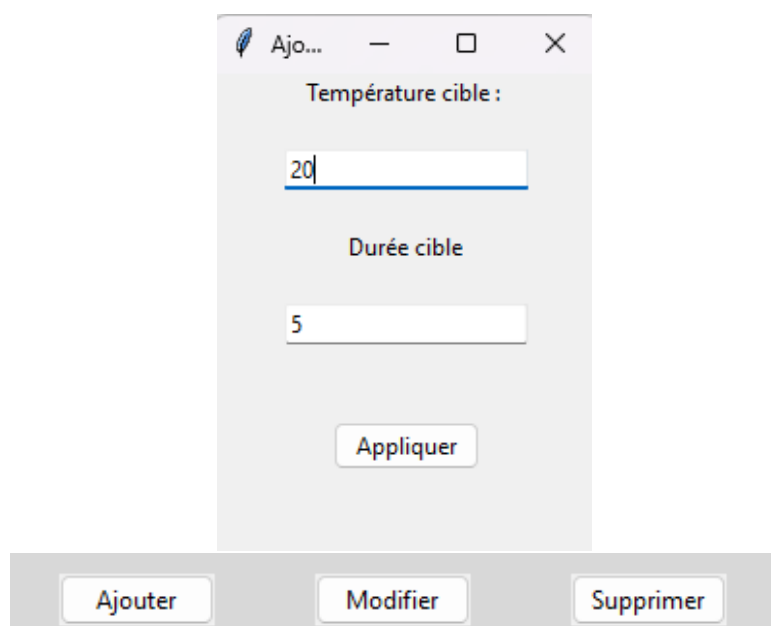


Pour réaliser ce bouton j'ai pris une image que j'ai redimensionner pour qu'elle rentre parfaitement dans le bouton

8.2.5 Ajout des boutons liés aux étapes

Pour créer des boutons qui permettent d'ajouter, de supprimer et de modifier les étapes dans un tableau en Tkinter, plusieurs étapes sont nécessaires. Ensuite, pour chaque fonctionnalité (ajouter, supprimer, modifier), on définit des fonctions correspondantes qui seront exécutées lorsque les boutons associés seront cliqués. Par exemple, pour ajouter une étape, la fonction récupérera les informations saisies par l'utilisateur dans des champs de saisie dans une pop-up, puis les ajoutera au tableau. Pour supprimer une étape, la fonction supprimera la dernière ligne du tableau. En associant ces fonctions aux clics des boutons correspondants, on permet à l'utilisateur de gérer facilement les étapes de l'expérience dans le tableau.

Le bouton ajouter ouvre une pop-up lors de l'interaction. Cette pop-up contient deux champs texte le premier pour rentrer la température cible, celle-ci a une valeur par défaut de 20 C°, le deuxième champs correspond à la durée cible en minute ça valeur par défaut est 5 minutes. La durée cible signifie le temps que doit mettre l'étuve pour atteindre la température cible.



8.2.6 Création du tableau des étapes

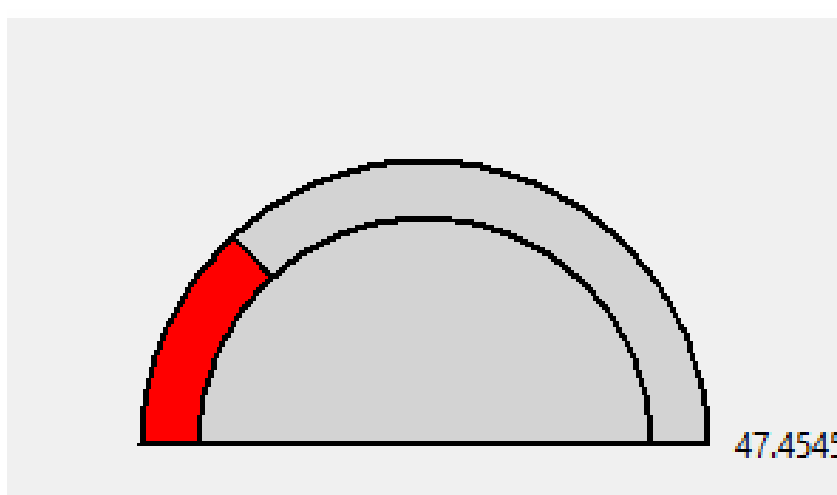
Pour créer un tableau qui récupère les données d'une pop-up de saisie de température et de temps cible pour les afficher, j'ai utilisé ces données pour mettre à jour un tableau dans la fenêtre principale. Pour chaque nouvelle entrée de données de température et de temps cible, nous ajoutons une nouvelle ligne au tableau avec les valeurs correspondantes. De cette manière, le tableau affiche toutes les données saisies par l'utilisateur au fur et à mesure qu'elles sont ajoutées.

En associant la fonction d'affichage des données à la confirmation de la saisie dans la pop-up, nous pouvons garantir que les données saisies sont immédiatement affichées dans le tableau. Cela crée une expérience utilisateur fluide et intuitive, où les utilisateurs peuvent voir rapidement les données qu'ils ont saisies et les analyser dans le contexte du tableau.

8.2.7 Jauge Température

Je récupère les données de température pour mettre à jour la jauge dans notre interface graphique. Nous ajustons la position de la jauge en fonction de la valeur de température actuelle. Cela permet à l'utilisateur de visualiser facilement la température en un coup d'œil grâce au remplissage de la jauge.

En associant la mise à jour de la jauge à intervalles réguliers avec les données provenant du capteur de température, nous assurons que la jauge reflète toujours avec précision la température actuelle. Cela crée une interface utilisateur dynamique et informative, permettant à l'utilisateur de surveiller facilement les changements de température en temps réel.


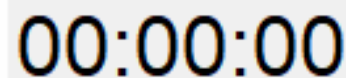


8.2.8 Lancer expérience

Pour créer un bouton "Lancer expérience" qui active un chronomètre dans une interface utilisateur (UI) développée avec Tkinter en Python, plusieurs étapes sont nécessaires. J'ai ajouté un bouton "Lancer expérience" à cette fenêtre. Lorsque l'utilisateur clique sur ce bouton, une fonction associée à cet événement est déclenchée et se transforme en annuler expérience.

Pour afficher le chronomètre à l'utilisateur, nous pouvons utiliser un label Tkinter qui sera mis à jour à intervalles réguliers pour afficher le temps écoulé depuis le début de l'expérience. Cela peut être réalisé en utilisant la méthode `after()` de Tkinter pour planifier la mise à jour du label à intervalles réguliers, par exemple toutes les 100 millisecondes.

En associant cette fonction de mise à jour à l'horloge système ou à un compte à rebours basé sur le temps écoulé depuis le début de l'expérience, nous pouvons fournir à l'utilisateur une indication en temps réel du temps écoulé.



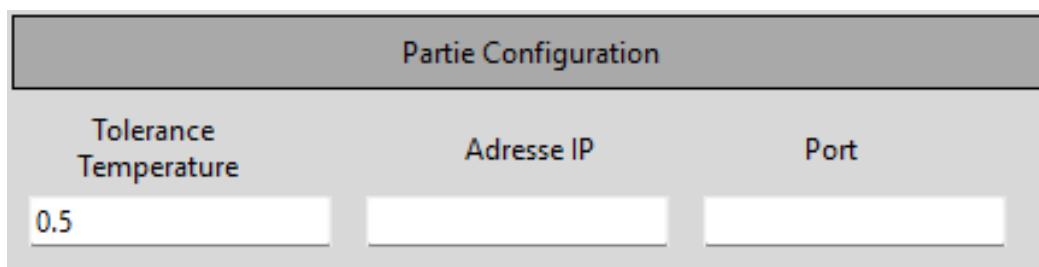
Lancer
l'expérience

8.2.9 Partie configuration

Pour créer la partie configuration d'une interface utilisateur, qui comprend le port, l'adresse IP et la tolérance de température.

Pour le port, la tolérance de température et l'adresse IP, nous pouvons utiliser des champs de saisie textuelle où l'utilisateur peut entrer les valeurs nécessaires. Nous pouvons également ajouter des libellés pour indiquer à quoi servent ces champs.

En suivant ces étapes, les développeurs peuvent créer efficacement une partie configuration dans leur interface utilisateur Tkinter, permettant à l'utilisateur de spécifier les paramètres nécessaires pour l'application, tels que le port, l'adresse IP et la tolérance de température.



Partie Configuration		
Tolerance Temperature	Adresse IP	Port
0.5		

8.2.10 Enregistrer et importer

Lorsque l'utilisateur clique sur le bouton "Importer", une fonction associée est déclenchée. Cette fonction ouvre une boîte de dialogue de fichier à l'aide de laquelle l'utilisateur peut sélectionner le fichier qu'il souhaite importer. Une fois le fichier sélectionné, la fonction peut traiter les données de ce fichier selon les besoins de l'application.

De même, lorsque l'utilisateur clique sur le bouton "Enregistrer", une fonction associée est déclenchée. Cette fonction peut enregistrer les données actuelles de l'application dans un fichier spécifié par l'utilisateur. Encore une fois, une boîte de dialogue de fichier est généralement utilisée pour permettre à l'utilisateur de choisir l'emplacement et le nom du fichier dans lequel enregistrer les données.

Enregistrer

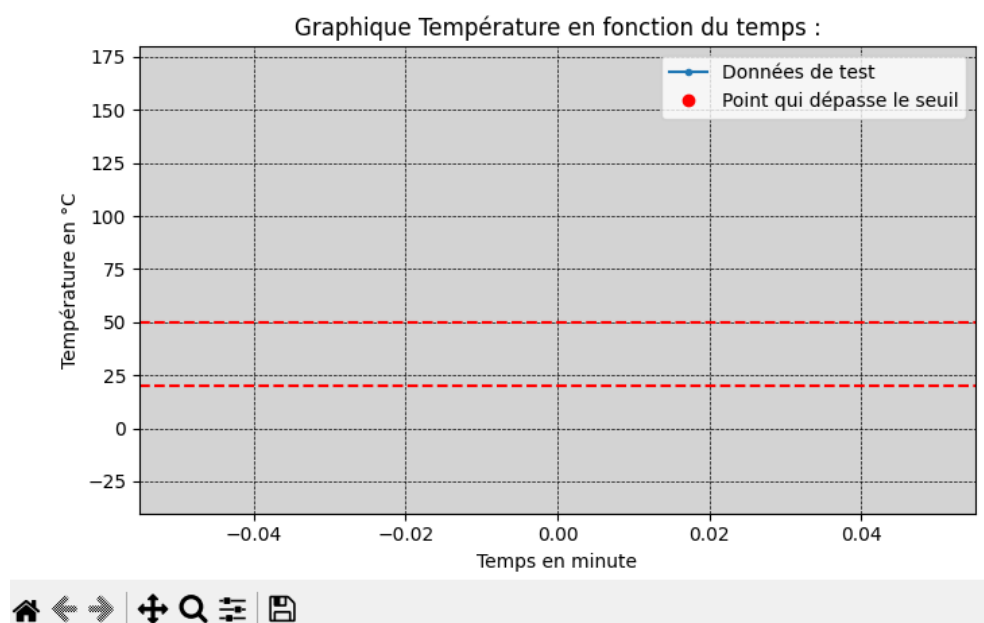
Importer

8.2.11 Graphique

Pour obtenir les données de température par rapport au temps, un capteur de température est utilisé pour mesurer la température à intervalles réguliers. Ces données sont ensuite collectées et stockées.

Ensuite, les données de température et de temps sont utilisées pour générer le graphique à l'aide de Matplotlib. Le temps est représenté sur l'axe des x, tandis que les températures sont représentées sur l'axe des y. Les données sont tracées sous forme de ligne ou de points, selon la préférence de l'utilisateur.

Une fois que le graphique est généré, il est ajouté au widget de graphique dans la fenêtre Tkinter. Les utilisateurs peuvent ainsi visualiser les variations de température au fil du temps d'une manière claire et intuitive. Il est possible de se déplacer sur le graphique il est aussi possible de zoomer.



8.2.12 Bilan

On fait le bilan calmement posé repensant à chaque instant.

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
IHM	Réalisation de l'IHM.	L'IHM est fonctionnelle et répond à la demande du client	Conforme	En cours

Onglet	Création d'onglet.	Les onglets permettent de changer de page	Conforme	Fini
Ajout d'une étape	Essaie des fonctionnalités.	Ouverture de la pop-up et ajout d'une étape	Conforme	Fini
Modifier	Modifier une étape.	Ouverture de la pop-up et modification des données	La modification n'est pas effectuée	En cours
Supprimer	Supprimer une étape.	Lors du clic le dernier élément du tableau est supprimé	Conforme	Fini
Tableau d'étapes	Est-ce que le tableau affiche les étapes ?	Les étapes s'affiche dans le tableau	Conforme	Fini
Jauge température	Est-ce que la jauge est réactive ?	Le remplissage de la jauge est modifié	Conforme	Fini
Lancer expérience	Est-ce que le bouton devient « annuler expérience », le chrono se lance et la partie étape se désactive ?	Le bouton devient « annuler expérience », le chrono se lance et la partie étape se désactive.	Le bouton devient « annuler expérience », le chrono se lance.	En cours
Configuration	Est-ce que la partie configuration est complétée correctement	La partie configuration est complétée correctement	Conforme	Fini
Enregistrer et importer	Est-ce que l'enregistrement et l'importation fonctionne ?	L'importation et l'enregistrement fonctionne.	Je peux enregistrer un mon tableau en fichier .json et importer	Fini
Graphique	Est-ce que mon graphique est fonctionnel ?	Graphique fonctionnel	Le graphique est fait mais fonctionne pas correctement	En cours

8.3 Etape 3 – Utilisation du mock

8.3.1 Objectifs

L'objectif de cette tâche est d'utiliser le « squelette » de la bibliothèque de classe de Léandre pour par la suite faciliter l'intégration. Le "mock" est un outil précieux en développement logiciel pour simuler le comportement de composants

ou de dépendances externes lors des tests unitaires. Plutôt que d'utiliser des composants réels, qui peuvent introduire des variables indésirables ou des instabilités dans les tests, les développeurs utilisent du "mock" pour imiter le comportement de ces composants de manière contrôlée.

En utilisant un "mock", nous pouvons tester notre application dans des conditions contrôlées, en nous assurant que chaque composant fonctionne comme prévu, sans avoir à nous soucier des dépendances externes. Cela rend nos tests plus rapides, plus fiables et plus faciles à maintenir. En fin de compte, l'utilisation de "mock" contribue à améliorer la qualité du code et la stabilité de l'application.

8.3.2 Éléments de réalisation

La mise en œuvre d'une utilisation de "mock" dans le développement logiciel repose sur plusieurs éléments clés pour assurer son efficacité. Tout d'abord, il est essentiel de déterminer les composants ou les dépendances externes qui doivent être simulés à l'aide de "mock" dans le cadre des tests unitaires.

Ensuite, il est crucial de créer un "mock" approprié qui simule fidèlement le comportement des composants réels tout en offrant un contrôle total sur les données et les scénarios de test.

Une fois le "mock" créé, il est important de les intégrer de manière transparente dans les tests unitaires existants en remplaçant les dépendances réelles par le "mock" approprié. Cela permet d'isoler le code testé et de garantir que les tests sont fiables et reproductibles, indépendamment des fluctuations dans les composants externes.

Enfin, il est crucial de maintenir les "mock" au fil du temps pour refléter les évolutions du code et des dépendances. Cela garantit que les tests restent pertinents et fiables à mesure que le logiciel évolue, contribuant ainsi à la qualité globale du produit final.

En combinant ces éléments, les développeurs peuvent tirer pleinement parti de l'utilisation de "mock" pour améliorer la couverture des tests, accélérer les cycles de développement et garantir la qualité et la stabilité du logiciel.

8.3.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
mock	Utilisation du mock	L'IHM est fonctionnelle et répond à la demande du client	Conforme	Ok

8.4 Etape 4 – Intégration

8.4.1 Objectifs

L'objectif de cette tâche est d'utiliser la bibliothèque de classe de Léandre pour que l'IHM fonctionne correctement. L'intégration est un processus complexe mais essentiel dans de nombreux domaines. Fondamentalement, il s'agit de combiner différents éléments de manière harmonieuse et fonctionnelle. L'intégration peut impliquer l'assemblage de

différentes composantes matérielles ou logicielles pour créer un système fonctionnel. Cela nécessite souvent une compréhension approfondie des différents éléments impliqués, ainsi qu'une vision claire de l'objectif final. En fin de compte, l'intégration bien exécutée peut conduire à de bons résultats, où les parties individuelles contribuent à créer quelque chose de plus grand que la somme de ses parties.

8.4.2 Éléments de réalisation

L'intégration de classes dans un programme en Python implique plusieurs éléments essentiels pour assurer une conception cohérente et efficace. Tout d'abord, il est important de définir clairement les responsabilités de chaque classe et de déterminer comment elles interagissent entre elles pour atteindre les objectifs du programme. Cela nécessite une analyse approfondie des besoins fonctionnels et des relations entre les différentes parties du système.

Ensuite, il est crucial de concevoir des interfaces claires et bien définies pour chaque classe, décrivant les méthodes disponibles et les paramètres requis pour interagir avec elle. Les interfaces bien conçues facilitent l'intégration des classes en définissant des points d'interaction clairs et en minimisant les dépendances directes entre les composants.

Une fois les interfaces définies, l'intégration des classes implique généralement l'instanciation et l'utilisation de ces classes dans le programme principal. Cela peut nécessiter la création d'objets à partir des classes et l'appel de leurs méthodes pour exécuter les fonctionnalités spécifiques du programme.

Enfin, il est crucial de tester soigneusement l'intégration des classes pour garantir leur bon fonctionnement dans le contexte global du programme. Cela peut impliquer des tests unitaires pour chaque classe individuelle, ainsi que des tests d'intégration pour vérifier que les différentes parties du système interagissent correctement les unes avec les autres.

En combinant ces éléments, les développeurs peuvent réaliser une intégration efficace des classes dans un programme Python, garantissant ainsi un code bien organisé, modulaire et facile à maintenir.

8.4.3 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
integration	Utilisation de la bibliothèque de classe	L'IHM est fonctionnelle et répond à la demande du client	En cours	En cours

8.5 Etape 5 – Reste à faire

8.5.1 Finir la fonction modifier

A l'heure actuelle le bouton modifier n'est pas terminer, il faut donc que j'arrive à récupérer les données de la ligne du tableau à modifier et les mettre dans la pop-up.

8.5.2 Afficher la valeur de la jauge en C°

La valeur qui est affiché avec la jauge est une valeur entre 0 et 180°, je dois afficher la valeur de la température à la place.

8.5.3 Finir la fonction lancer expérience

Actuellement quand je clic sur le bouton lancer expérience le résultat obtenu est :

- Le bouton devient « Annuler expérience »
- Le chrono se lance

Il me reste à désactiver toute la partie ajout, modification et supprimer du tableau des étapes pour ne pas pouvoir interagir lors de l'expérience. Les boutons ne pourront pas être cliqué.

8.5.4 Lier les données du tableau avec le graphique

Mon graphique est fait mais il manque l'affichage des courbes de prédiction et courbe réel.

8.6 Bilan

Le projet m'a apporté beaucoup de connaissance surtout au niveau du développement d'une interface graphique en python. C'est une très bonne expérience pour moi que ce soit du travail d'équipe ou sur plan personnel.

9 Candidat 2 : PERRET Léandre : Back-end Python, bibliothèque de classe et API.

9.1 Introduction et présentation générale du rôle

Développeur Python au sein du projet Piséo etuve s'occupant de la partie Back-End. Mon rôle au sein de cette équipe est de développer une API-REST permettant l'accès aux différentes fonctionnalités de l'étuve telles que la lecture de la température en son sein ou la transmission de consigne à l'étuve. Cette API sera faite par le biais du framework Django qui permet de coder facilement des applications web via du python. Il est principalement connu pour sa rapidité de navigation à travers des navigateurs internet et aussi de sa sécurité évitant beaucoup de problèmes aux utilisateurs et aux développeurs. Je me dois aussi de fournir une bibliothèque de classe permettant manipulation de l'étuve et le bon fonctionnement de l'IHM produite par mon camarade et qui soit réutilisable. Pour finir, je dois intégrer les travaux de mon camarade ayant étudié les trames réseaux au sein de mon API et de mes classes afin de permettre le tout de dialoguer avec la dîtes etuve.

Le tout sera codé via le langage Python qui en plus de ces bibliothèques natives utilisera quatre autres bibliothèques telles que, python-dotenv qui permet de lire et de manipuler des fichiers .env qui sont des fichiers permettant de définir des variables de configurations d'applications, matplotlib permettant d'avoir une vision de données mathématiques via des graphes, Django qui est le framework me permettant de faire tourner mon API et aussi la bibliothèque pyModbusTCP permettant de dialoguer via le protocole ModbusTCP sur des liaisons Ethernet et Internet. La mise en forme des données s'opérera via le format Json.

9.2 Planning prévisionnel individuel

Avant de commencer à la production des différentes tâches qui me sont données, j'ai d'abord rédigé un planning prévisionnel quant aux tâches que je me dois effectuer (Voir figure 22).

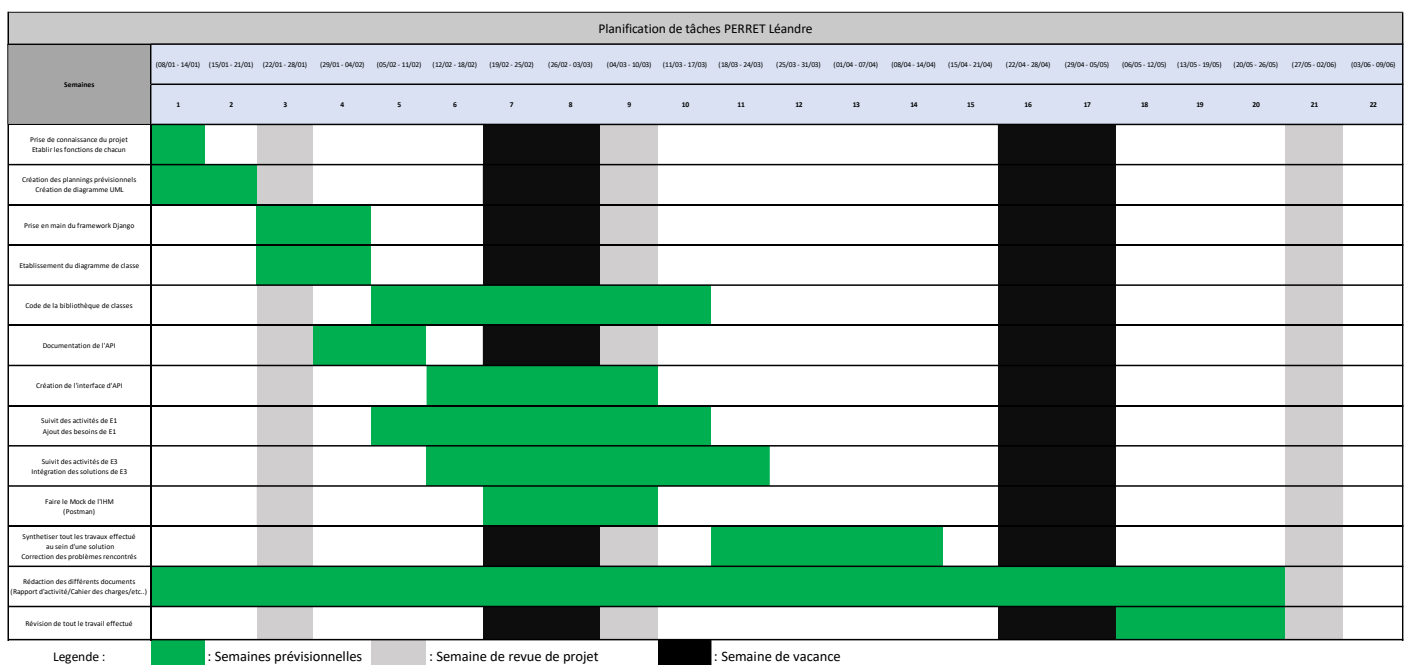


Figure 19 : Planning Léandre P.

Cependant ce planning étant prévisionnel, je n'avais pas encore conscience de ce qu'impliquaient les différentes tâches que j'allais mener et du temps que ça allait me prendre, au final rien de ce planning s'est révélé être vrais et la réalité se transcrit par un peu toutes les tâches furent effectuées en même temps car cela me permettait d'avoir une

meilleure connaissance de l'ensemble du projet, la tâche sur laquelle j'ai passé plus de temps est celle de la mise en place de l'API-REST.

9.3 Tableau des tâches

L'ensemble de ce que j'ai pu produire au sein du projet peut se résumer au sein de quatre tâches différentes qui sont les suivantes :

Tâches	Description	Validée ?
Etablissement de diagramme UML	Création de diagramme UML permettant de transcrire les classes et le déploiement de l'application	OK
Code de la bibliothèque de classe	Bibliothèque de classe permettant la manipulation des données fournies par l'étuve et le fonctionnement de l'IHM.	OK
API sous Django	API permettant de consigner ou de récupérer la température de l'étuve	OK
Mise en commun	Rassembler toutes les solutions produites par le groupe au sein d'un même programme (pilotage de l'étuve, API, IHM)	En cours

La première d'entre-elles est celle de l'établissement de diagramme UML, car en plus de ceux présenter en figure 9 à 12, j'ai dû en plus en rédiger un qui retranscrit la bibliothèque de classe que j'ai codée. La seconde est le code de la bibliothèque permettant l'application de fonctionnée, c'est elle qui s'occupera de manipuler les différentes données et de mener à terme l'expérimentation définit par l'utilisateur. Enfin, l'API sous Django, cette tâche consiste à prise en main du framework Python : Django, afin de créer une API accessible par des requêtes permettant d'accéder aux deux fonctionnalités primordiales de l'étuve : lecture de température et consigne de température. Pour finir sur la mise en commun, qui consiste à inclure les solutions produites par mon camarade Maxime K. au sein de mes classes et de l'API, mais aussi d'inclure mes classes au sein de l'IHM de mon camarade Lucas L.

9.4 Code de la bibliothèque de classe

9.4.1 Introduction et objectif

Production d'une bibliothèque de classe avec son diagramme UML permettant le bon fonctionnement.

Tâches	Description	Validée ?
Diagramme UML de classe et données sous forme d'objet	Création d'un diagramme UML sur les classes présentes au sein de l'application et création des classes permettant de mettre en forme les données	OK
Manipulation des données	Pouvoir manipuler les données utilisées au sein de l'application	OK
Fonctionnement de l'expérimentation	Codage des attributs et méthodes permettant le bon fonctionnement de l'expérimentation au sein de l'IHM	OK
(BONUS) Affichage graphique des données	Utilisation de la bibliothèque Matplotlib afin de pouvoir visualiser les données manipulées durant l'expérimentation.	En cours

9.4.2 Éléments de réalisations

9.4.2.1 Diagramme UML

Afin de mieux présenter mes classes, j'ai rédigé un diagramme UML de classe (Voir figure 25).

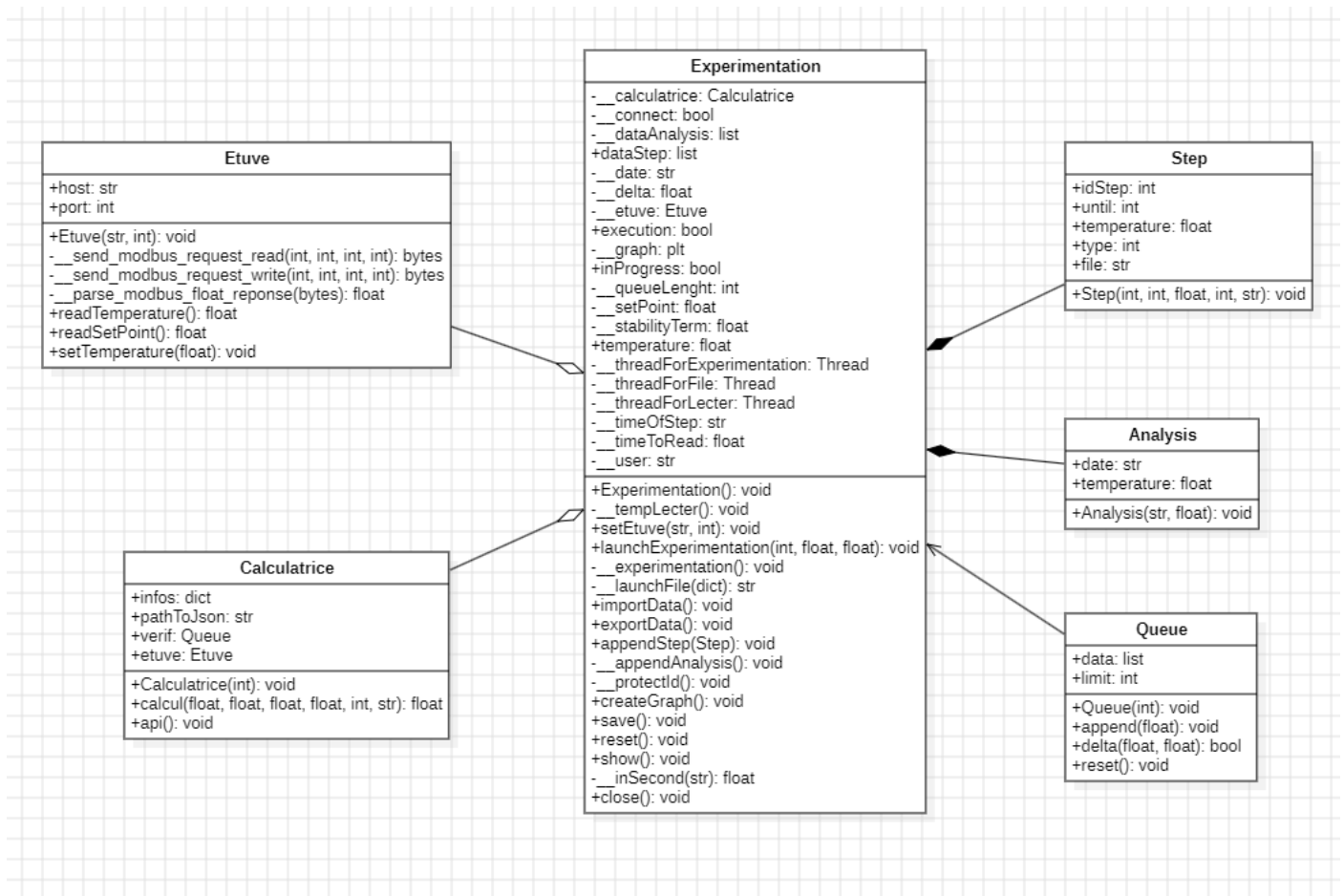


Figure 20 : Diagramme de classes UML (Léandre P.)

L'ensemble de l'application fonctionnera sur 6 classes, le fonctionnement global de ces classes est le suivant : une classe générale nommée « Experimentation » permettant la grande majorité des actions faites par l'application. Une classe « Etuve » permettant le dialogue avec l'étuve. Une classe « Calculatrice » permettant principalement les calculs et le fonctionnement de l'API. Puis deux classes « Step » et « Analysis » permettant la mise en forme des données. Enfin la classe « Queue » qui me permet au sein de la classe « Experimentation » de vérifier mes données.

9.4.2.2 Classe Etuve

Parmi elles, se trouve la classe Etuve, celle-ci fut codée par mon collègue Maxime qui s'occupe des trames réseaux. Elle permet d'encoder et de traduire les trames réseaux nécessaires au dialogue entre l'application et l'étuve. Cette classe comprend deux attributs qui sont :

- host : String qui représente l'adresse IP de l'étuve.
- port : Nombre entier qui représente le port de connexion de l'étuve.

Et différentes méthodes permettant l'encryptage et le cryptage de trames réseaux.

9.4.2.3 Classe de stockage de données.

Parmi les classes s'en trouvent deux qui permettent de définir le type des données avec lesquelles nous travaillons et de les stockées. La première des deux est la classe Step qui représente les étapes définies par l'utilisateur souhaitant utiliser notre application, elle est représentée par 5 attributs :

- idStep : Nombre entier représentant son ID et son numéro de passage au niveau de l'expérimentation.
- until : Nombre entier de secondes durant lesquelles l'étape va perdurer.
- temperature : Nombre flottant représentant la température cible souhaitée indiquée par l'utilisateur.
- type : Nombre entier représentant le type d'étape qu'il s'agit (0 = pas d'attente, 1 = on attend que l'utilisateur donne son feu vert pour passer à la prochaine étape, 2 = on attend qu'un programme python se finisse pour passer à la prochaine étape).
- file : String qui permet de définir le chemin vers un fichier qui doit être lancé si le scientifique souhaite lancer un fichier lorsque l'étape est atteinte.

Seule une méthode accompagne la classe Step, c'est celle du constructeur permettant d'initialiser un objet. L'autre classe de type de données est Analysis, celle-ci représente toutes les mesures faites lors de l'expérimentation sont alors conservées au sein d'objet Analysis. Cette classe est définie par 2 attributs :

- date : String permettant de définir la date et heure à laquelle la température fut mesurée.
- temperature : Nombre flottant représentant la température mesurée à cet instant.

Une seule méthode est présente au sein de la classe, c'est celle du constructeur permettant encore une fois d'initialiser un objet.

9.4.2.4 Classe Queue

Cette classe me servira au sein d'un algorithme afin de vérifier les valeurs en son sein. Elle s'inspire de la structure de données nommée « File » connue par l'acronyme FIFO (first in, first out) qui définit une sorte de liste où la première valeur rentrée et aussi la première à sortir de la file. Le plus intéressant pour moi, c'est son mode opérationnel avec un nombre limité de valeur me permettant de ne vérifier qu'un certain nombre de valeur.

9.4.2.4.1 Attributs

Elle ne possède que deux attributs :

- data : C'est la liste contenant toutes les valeurs. Valeur par défaut : []
- limit : c'est le nombre de valeur maximale contenue au sein de l'attribut data.

9.4.2.4.2 Méthodes

Quelques méthodes ont été codées au sein de cette classe :

9.4.2.4.2.1 Constructeur

Prends en argument un nombre entier afin de définir la taille limite de la queue.

9.4.2.4.2.2 append(float)

Récupères en argument une valeur à rajouter au sein de l'attribut « data ». S'il y a trop de valeur en son sein, supprime le premier arrivé et rajoute à la fin de la liste la valeur.

9.4.2.4.2.3 delta(float,float)

Permet de vérifier si toutes les valeurs au sein de l'attribut « delta » soient à une distance définie par le second argument d'une valeur définie par le premier argument. Exemple : si j'ai comme valeur dans l'attribut « data » : [9,5,8,6] et que je lance delta(10,5) alors toutes les valeurs présentes au sein de l'attribut vont être vérifiées si elles sont proches de 10 avec une distance d'erreur de 5. Dans ce cas, la méthode retournera True par contre si une des valeurs est plus grande que 15 ou plus petite que 5 et que je lance delta(10,5) alors ça me retournera False.

9.4.2.4.2.4 `reset()`

Remet à zéro l'attribut « data ».

9.4.2.5 Classe Experimentation

Maintenant, parlons de la plus grosse des classes qui se nomme Experimentation, elle permet de nombreuses choses, allant à la manipulation de données au procédé d'une expérimentation.

9.4.2.5.1 Manipulation des données

La première chose que me permet cette classe c'est, déjà de stocker les données et de les manipuler pour cela, j'ai différents attributs et méthodes

9.4.2.5.1.1 Attributs

Les attributs permettant le stockage des données sont les suivants :

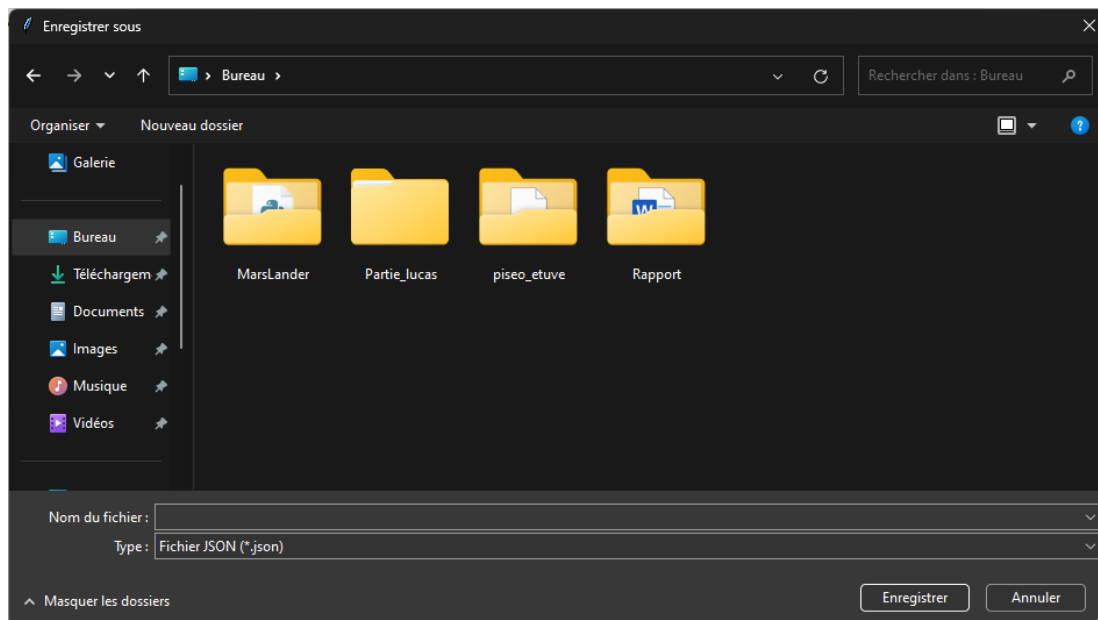
- dataStep : Liste d'étapes contenant toutes les étapes données par l'utilisateur. Valeur par défaut : []
- dataAnalysis : Liste de températures mesurées lors de l'expérimentation. Valeur par défaut : []
- date : String représentant la date et heure à laquelle fut lancée l'expérimentation. Valeur par défaut : date du jour sous forme JJ/MM/AAAATHH:mm:ss (J = jour, M = mois, A = année, T = simple séparateur afin de différencier jour et heure, H = heure, m = minute et S = seconde)
- user : String qui permet d'identifier l'utilisateur ayant mené l'expérimentation. Valeur par défaut : Nom de la session de l'utilisateur.

9.4.2.5.1.2 Méthodes

Il y a un total de six méthodes permettant la manipulation de ces attributs :

9.4.2.5.1.2.1 exportData()

La méthode exportData est une méthode permettant à un utilisateur, s'il le souhaite, d'exporter les données de l'expérimentation. Pour ce faire, elle ouvre l'explorateur Windows via une méthode de la bibliothèque pyplot de tkinter en demandant l'emplacement et le nom du fichier qui va être créé contenant toute la donnée (Voir figure). Suite à ça, tous les attributs de données vont être formatés sous le format Json et vont être ensuite sauvegardés au sein du fichier qui a été paramétré dans l'explorateur.



9.4.2.5.1.2.2 importData()

La méthode importData sert quant à elle à récupérer un des fichiers enregistrés par la méthode exportData afin d'importer les données enregistrées. Avec un intérêt particulier aux données d'étapes enregistrées, car c'est elles qui permettent de refaire une expérience déjà menée. Pour ce faire, elle récupère le fichier via l'explorateur Windows par

une méthode de la bibliothèque pyplot de tkinter permettant de demander à l'utilisateur un fichier dans leur ordinateur (Voir figure).

9.4.2.5.1.2.3 *appendStep(Step)*

Cette méthode-là permet de rajouter des étapes au sein du tableau d'étapes, c'est elle qui me permettra par la suite de manipuler les températures de l'étuve comme souhaitées via un algorithme.

9.4.2.5.1.2.4 *appendAnalysis()*

Tout comme `appendStep`, celle-ci me permet de rajouter une température atteinte par l'étuve au sein de mes données en allant piocher directement la date d'aujourd'hui et la température de l'étuve afin de créer un objet `Analysis`.

9.4.2.5.2 Algorithmie de l'expérimentation

Maintenant, que ma classe possède des données, je me suis penché sur la mise en œuvre d'une expérimentation, c'est la partie qui me fut la plus compliquée à mettre en place et qui possède beaucoup d'attributs et de méthodes afin qu'elle soit opérationnelle.

9.4.2.5.2.1 Attributs

De nombreux attributs me servent afin de suivre le bon déroulé de mon expérimentation en plus de ceux de données :

9.4.2.5.2.1.1 *Attributs d'états*

Quelques-uns de ces attributs me servent afin de savoir où en est mon algorithmie ou pour indiquer à mon algorithme que certaines actions doivent être terminées.

- `inProgress` : C'est un booléen permettant d'indiquer si une expérimentation est en train d'être menée ou non. Lorsqu'elle passe à `False`, l'expérimentation actuelle s'arrête. Valeur par défaut : `False`
- `execution` : Booléen permettant lui de savoir si l'application est toujours en cours, elle me permet d'éteindre tous mes threads lancés lors de l'expérimentation afin d'éviter qu'à la fermeture de l'IHM ceux-là continuent de fonctionner. Valeur par défaut : `True`

9.4.2.5.2.1.2 *Attributs de threading*

Certains de ces attributs sont des threads me permettant d'exécuter plusieurs tâches à la fois.

- `threadForExperimentation` : Ce thread-là me permet de lancer l'algorithme s'occupant le dérouler de l'expérimentation fournie sans pour autant bloquer le thread principal.
- `threadForFile` : Celui-là n'est pas encore utilisé, mais dans une prochaine évolution du programme, il servira à lancer des programmes ayant été fournie durant le paramétrage d'une étape (ces programmes sont principalement des tests conçus par les scientifiques).
- `threadForLecter` : Celui-ci ce lancer dès la création d'un objet de la classe `Experimentation`, il permet de lire en boucle la température de l'étuve et d'y consigner les températures voulues.

9.4.2.5.2.1.3 *Attributs de classes externes*

Mon algorithme utilise un total de deux classes externes qui sont initialisées par le biais d'attributs.

- `etuve` : Attribut initialisé via la classe `Etuve` codée par mon camarade Maxime K., il me permet la communication avec une étuve distante. Valeur par défaut : étuve possédant une adresse IP 0.0.0.0 et un port 0.
- `calculatrice` : Celui-ci est initialisé à la classe `Calculatrice` qui me permet de faire les calculs dans mon algorithme.

9.4.2.5.2.1.4 *Attributs d'informations*

Une grande partie des attributs me servent principalement afin de permettre à mon algorithme de stocker des informations lui permettant son bon déroulement.

- connect : Cet attribut me permet de savoir si une étuve est bien connectée. Valeur par défaut : False
- delta : Celui-ci me permet de vérifier la stabilité de la température lorsqu'elle est censée atteindre la valeur finale donnée par l'utilisateur, je vérifie sa stabilité à plus ou moins une valeur près de la valeur finale. Cette valeur est définie par l'attribut delta. Valeur par défaut : 1
- queueLenght : Lui aussi utilisé pour vérifier la stabilité, c'est le nombre de valeurs de température qui soient assez proche de la valeur finale. Pour exemple, si défini par 4 alors mon algorithme attendra que 4 valeurs soient proches de la valeur finale afin de conclure à la stabilité de la température. Valeur par défaut : 4
- setPoint : Valeur de température donnée à l'étuve afin qu'elle chauffe jusqu'à cette valeur. Valeur par défaut : 0
- stabilityTerm : Troisième et dernier attribut permettant de vérifier la stabilité, il définit lui un temps minimum avant de confirmer la stabilité. Par exemple si définit par 5 minutes et que l'étuve a atteint la température demandée alors la stabilité de cette température ne sera pas acceptée avant 5 minutes. Valeur par défaut : 0
- temperature : Dernière valeur de température lue de l'étuve. Valeur par défaut : 0
- timeOfStep : Heure du lancement d'une étape, à chaque fois que l'algorithme commence à suivre une étape, il écrit au sein de cet attribut l'heure de commencement afin de lui permettre par la suite de faire ses calculs.
- timeToRead : Durée d'attente entre deux lectures de température. Valeur par défaut : 0

9.4.2.5.2.2 Méthodes

Pour ce qu'y est des méthodes, il y en a un total de 6, dont trois qui servent pour le dérouler d'une expérimentation.

9.4.2.5.2.2.1 Constructeur

Il initialise tous les attributs à leurs valeurs initiales puis lance aussi le thread s'occupant de lire la température de l'étuve et de consigne de température.

9.4.2.5.2.2.2 tempLecter()

Cette méthode est lancée dans un thread à part lors de l'initialisation d'un objet Experimentation. C'est une boucle qui lit la température de l'étuve, si elle y arrive alors elle met l'attribut « connect » à True sinon False. Si une expérimentation est en cours, elle utilise la méthode « appendAnalysis » afin de rajouter la dernière valeur de température lue au sein des données puis consigne la prochaine température que doit atteindre l'étuve qui est définie par l'attribut « setPoint ». C'est ainsi que via un thread, je m'occupe de lire et de consigner la température.

9.4.2.5.2.2.3 setEtuve(str,int)

Cette méthode récupère une adresse IP et un port en argument puis remplace la dernière étuve par une nouvelle dans l'attribut « etuve » qui est définie par les arguments de la fonction.

9.4.2.5.2.2.4 launchExperimentation(int,float,float)

Cette méthode permet de lancer une expérimentation une fois que tout est paramétré, elle récupère en entrée trois arguments « queueLenght », « delta » et « stabilityTerm ». Chacun sera mis au sein des attributs possédant le même nom qu'eux. Puis lance au sein de l'attribut « threadForExperimentation » un thread qui permet à la fonction « experimentation » de se dérouler dans un thread à part.

9.4.2.5.2.2.5 experimentation()

Fonction principale de la classe, c'est l'algorithme s'occupant du bon dérouler de l'application. Celui-ci se continue tant que les attributs « connect », « inProgress » et « execution » sont égal à True permettant d'empêcher quelconque erreur en cas d'étuve non branchée et en cas d'arrêt de l'expérimentation ou du programme, ça permet aussi de l'arrêter. Le dérouler de l'algorithme suit la logique suivante : Vérification qu'une étuve est bien connectée -> si True alors commencement de l'expérimentation. Mise en place d'une queue de vérification de stabilité (voir classe Queue page 36), de taille « queueLenght ». Récupère toutes les étapes de « dataStep » au sein d'une variable nommée « stepList » et fait passer l'attribut « inProgress » à True. C'est alors qu'une boucle while vérifiant si l'expérimentation est toujours en cours ou si le programme est fermé se lance. Cette boucle commence par vérifier s'il reste des étapes dans stepList à traiter sinon fait passer « inProgress » à False. Remets la queue à zéro, définit la température de départ à celle de l'étuve et met l'heure de départ à l'heure actuelle. Tant que l'on n'est pas arrivé à la minute définie par

l'utilisateur, je calcule à un intervalle défini par l'attribut « `timeToRead` » la prochaine température que je dois atteindre afin que quand j'arrive à la minute définie, je sois arrivé à la valeur finale de température (voir classe Calculatrice méthode calcul page 42, le calcul de température suit une fonction affine) et j'ajoute dans la queue de vérification la température actuelle de l'étuve. Une fois que je suis censé atteindre la température finale au moment voulue, je donne comme dernière valeur à atteindre la valeur finale et je vérifie la stabilité de la température durant une durée définie par l'attribut « `stabilityTerm` » (la vérification de stabilité s'effectue via la méthode « `delta()` » de la classe Queue). Lorsque la stabilité est vérifiée, je lance le fichier fourni par l'utilisateur et j'enlève l'étape de l'expérimentation. Je répète cette boucle jusqu'à qu'il n'y ait plus d'étape.

9.4.2.5.2.2.6 *launchFile(dict)*

Cette méthode récupère en argument un dictionnaire contenant un total de 4 clés. La première d'entre-elles : « `path` ». C'est l'emplacement du fichier que l'on souhaite lancer. La seconde valeur « `command` », permet de lancer une commande dans un cmd au lieu d'un fichier. « `argument` » permet de rajouter des arguments à l'ouverture du fichier. La dernière clé « `type` » permet de définir s'il faut attendre que le programme se termine ou si l'utilisateur donne son accord que c'est terminer ou s'il faut ne rien attendre et continue à la suite. Cette méthode bien qu'étant codée et testée n'est pas fonctionnelle à 100 % et demande d'être vérifiée dans des cas d'utilisations réelles.

9.4.2.5.2.2.7 *close()* :

Fait passer l'attribut « `execution` » à False afin de faire fermer tous les threads et que l'application s'arrête définitivement.

9.4.2.5.3 (BONUS) Affichage graphique des données

Durant le développement de cette classe, je me suis amusé à coder la possibilité d'afficher par une graphique les données lues lors de l'expérimentation. Cependant, à ce moment-là, il y avait une donnée de plus de lut qui était l'hygrométrie, mais due à un changement du cahier des charges, l'hygrométrie a disparue des valeurs à lire, mais ayant quand même été codé, je souhaitais montrer ce que j'ai effectué sur l'affichage graphique des données.

9.4.2.5.3.1 Attribut

Seul un attribut a été utilisé pour le graphique :

- `graph` : Initialisé comme objet pyplot de la bibliothèque tkinter, cette classe propose de nombreuses solutions afin de générer des graphiques mathématiques.

9.4.2.5.3.2 Méthodes

Afin de manipuler le graphe, quelques méthodes ont été codées telles que :

9.4.2.5.3.2.1 *createGraph()*

Méthode me permettant d'initialiser le graphe et d'y ajouter toutes les valeurs présentes au sein de mes données. Il nomme aussi les axes et donne un titre.

9.4.2.5.3.2.2 *save()*

Permet de sauvegarder en format .png le graphe afin de visualiser via une image toutes les données que l'expérimentation a récupérées.

9.4.2.5.3.2.3 *reset()*

Remets à zéro le graphe.

9.4.2.5.3.2.4 *show()*

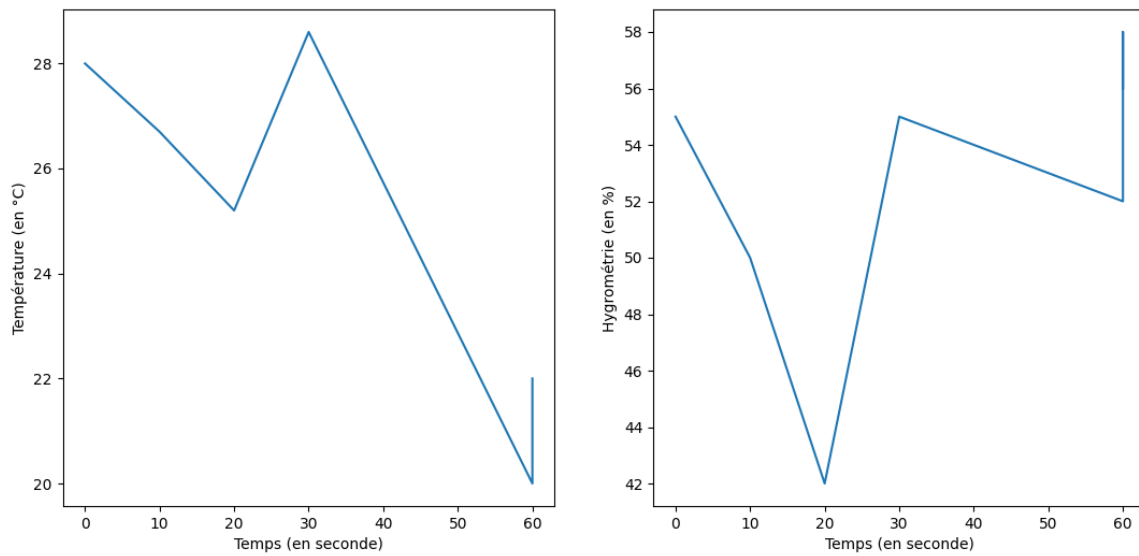
Utilise une méthode de la classe pyplot permettant d'afficher le graphe au sein d'une fenêtre.

9.4.2.5.3.2.5 *inSecond()*

Transforme une date en seconde (calcul le nombre de seconde entre la date et le 1/1/1970) afin qu'elle puisse être utilisée dans l'affichage du graphe.

C'est via ces méthodes-là que j'arrivais il y a quelques mois à fournir un résultat graphique à mes données qui ressemblait à cela :

Température et hygrométrie en fonction du temps



9.4.2.6 Classe Calculatrice

Cette classe me sert principalement pour l'API, cependant une des méthodes sert à calculer une température à transmettre à l'étuve permettant de suivre des courbes mathématiques. C'est pour cette raison que cette classe existe afin de permettre l'utilisation de cette méthode au sein de l'IHM et de l'API. Je vais donc présenter ici très rapidement les attributs et méthodes, mais je développerais plus en profondeur lorsque je parlerai de l'API.

9.4.2.6.1 Attributs

Pour ce qui est des attributs, il y en a un total de 4 qui me servent uniquement pour l'API.

- infos : Dictionnaire contenant les variables d'environnement afin de changer le changement de l'algorithme de l'API.
- pathToJson : C'est le chemin vers un fichier Json permettant à mon API de dérouler son algorithme.
- verif : C'est une « Queue » similaire à celle qu'il y a dans la méthode « experimentation » me permettant elle aussi de vérifier la stabilité, mais ici dans le cadre de l'API.
- etuve : C'est un attribut défini par la classe « Etuve » de mon camarade Maxime K. permettant les interactions entre l'API et l'étuve.

9.4.2.6.2 Méthodes

Pas énormément de méthode pour cette classe, il s'en trouve uniquement deux :

9.4.2.6.2.1 calcul(float,float,float,float,int,str)

Récupères en argument toutes les informations lui permettant de mener à bien son calcul tel que : la température à atteindre, en combien de temps l'on souhaite l'atteindre, depuis combien de temps l'on souhaite l'atteindre et pour finir la température initiale. Il y a pour l'instant que deux calculs possibles qui sont : suivre une fonction affine et suivre une fonction logarithme. Le but principal de cette méthode est de permettre de simuler une courbe de montée de température à l'étuve, car l'étuve monte de température comme elle le souhaite, on ne peut pas définir une montée en température spéciale. Donc cette méthode permet de corriger ce problème en simulant une montée via des courbes mathématiques ce qui aura pour effet qu'en donnant des valeurs pas à pas la température au sein de l'étuve est censée suivre cette courbe de plus ou moins proche en fonction du paramétrage (plus le temps entre deux consignes de température est petit, plus la température de l'étuve est censée suivre la courbe). Je souhaite intégrer comme possibilité en plus dans cette fonction celle de transmettre une fonction qui sera alors suivie par la méthode d'où le str en argument d'entrer, mais ce n'est pas encore mis en place.

9.4.2.6.2.2 api()

Cette méthode, je ne vais pas trop en parler ici, car je reviendrais dessus lorsque j'évoquerais l'API. Tout ce qu'il faut savoir, c'est que c'est un algorithme permettant à mon API de faire monter en température l'étuve comme l'utilisateur le souhaite ressemblant à celui présent au sein de la méthode « experimentation() » de la classe « Experimentation ». (Pour plus d'information quant à cette méthode voir page 53)

9.4.3 Conclusion

9.4.3.1 Cahier des charges

Fonctions	Description	Validée ?
Diagramme UML de classe	Rédaction d'un diagramme de classe type UML afin de représenter les classes utilisées	OK
Classes de données	Créer des classes représentant toutes les données manipulées par le logiciel	OK
Manipulation des données	Pouvoir manipuler les différentes données utilisées par le logiciel	OK
Manipulation de l'expérimentation	Permettre le bon dérouler de l'expérimentation programmée par l'utilisateur	OK
(BONUS) Affichage graphique des données	Afficher les données récupérées via une solution graphique.	En cours

9.4.3.2 Pour aller plus loin

Sur cette partie, le principal est terminé. L'algorithme fonctionne très bien, cependant, il reste plusieurs choses à inclure qui ne sont pas primordiales tel que permettre de faire des calculs via des fonctions mathématiques définies par l'utilisateur ou de remettre le graphe à jour avec les nouvelles données.

9.4.3.3 Bilan

Cette partie-là est bien moins intéressante que l'autre partie sur laquelle j'ai pu travailler. Bien qu'étant plus compliquée. Surtout qu'elle est très en accord avec mon camarade s'occupant de l'IHM ce qui a été l'origine de beaucoup de code retravaillé, modifié ce qui était nécessaire, mais qui au final m'a fait perdre pas mal de temps.

9.5 API-REST sous Django

9.5.1 Introduction et objectif

Utilisation du framework Django Python avec comme objectif de fournir une API fonctionnelle permettant lecture et manipulation de l'étuve.

Tâches	Description	Validée ?
Installation et mise en place de Django	Installation du framework puis mise en place d'un projet Django contenant une application	OK
Mise en forme des données en Json	Préparation des données qui vont être manipulées par notre API	OK
Mise en place des URL, des fonctions et des pages attribuées.	Paramétrage des différents fichiers python de Django afin de programmer des URL	OK
Mise en place d'une calculatrice	Programmer une calculatrice qui s'occupe de calculer les valeurs de température à donner à l'étuve.	OK
Variable d'environnement, debug et lancement de l'API	Mise en place de variable d'environnement, d'un mode debug et faciliter le lancement de l'API via un .bat	OK

9.5.2 Éléments de réalisations

9.5.2.1 Installation de Django

Afin de pouvoir commencer à travailler avec le framework Django, n'étant pas natif de base au langage python, il me fallut l'installer. Pour ça rien de bien compliquer il me fallut juste de rentrer la ligne suivant au sein de mon CMD :

```
pip install Django
```

Si Python est installé alors l'installation du framework se fera automatiquement.

9.5.2.2 Initialisation d'un projet Django

Maintenant l'initialisation d'un projet Django. Pour son bon fonctionnement, Django fonctionne sous le système de projets. Ces projets-là prennent forme sous un ensemble de paramètres et options spécifiques à Django, de base de données configurées et d'application. Ici nous n'allons pas utiliser les bases de données mais les paramètres spécifiques à Django et les applications nous seront utiles plus tard dans la mise en place de l'API. Pour que le projet soit fonctionnel, un serveur est fourni au sein du projet permettant d'y accéder à partir du net. Afin d'initialiser le projet pour l'API, je plaçai un CMD au sein d'un répertoire dans lequel je souhaitais que mon api soit et je lançai la commande suivante :

```
django-admin startproject apiEtuve
```

Ce qui pour action de créer différents répertoires et fichiers :

```
apiEtuve\apiEtuve
apiEtuve\manage.py
apiEtuve\apiEtuve\asgi.py
apiEtuve\apiEtuve\settings.py
apiEtuve\apiEtuve\urls.py
apiEtuve\apiEtuve\wsgi.py
apiEtuve\apiEtuve\__init__.py
```

Figure 21 : Création Projet apiEtuve

Ici se trouve différents fichiers python dont seuls trois d'entre eux vont nous intéresser. `manage.py` lancer avec comme arguments `runserver ip:port` permettra de lancer le serveur Django, `settings.py` est le fichier python contenant tous les paramétrages du serveur Django et `urls.py` qui permet de définir toutes les urls utiles.

9.5.2.3 Initialisation d'une application Django

Maintenant que nous avons notre projet il faut que nous créions une application, les applications permettent de définir le procédé qu'exécuterons les pages du serveur Django. Pour créer l'application contenant les pages de notre serveur il suffit de rentrer la commande suivante au sein d'un CMD qui se trouve dedans le répertoire `apiEtuve\` :

```
python manage.py startapp pages
```

Alors de nouveaux répertoires et fichiers python sont créer :

```
apiEtuve\pages\admin.py
apiEtuve\pages\apps.py
apiEtuve\pages\migrations
apiEtuve\pages\models.py
apiEtuve\pages\tests.py
apiEtuve\pages\views.py
apiEtuve\pages\__init__.py
apiEtuve\pages\migrations\__init__.py
```

Figure 22 : Création application Pages

Et c'est ainsi que notre serveur Django est initialisé et que nous pouvons commencer à travailler sur la mise en place des URL de celui-ci.

9.5.2.4 Mise en forme des données en Json

Juste avant de paramétrer les URL et le fonctionnement de celle-ci, il me fallait avoir un .json qui contient les données qui vont être manipulées par mon API lors d'une consigne de température vers l'étuve. Ce Json contient un objet possédant un total de 7 attributs qui sont les suivants :

- température : float

C'est l'attribut dans lequel sera stockées la variable de température consignée à l'étuve.

- second : int

Il sera stocké ici, en seconde, le temps voulu pour que la température atteigne cette valeur.

- since : float

Ici l'on récupère, en seconde, la date à la seconde près à laquelle fut donnée la consigne par rapport à la date du 1/1/1970.

- base : float

Température de base à laquelle l'étuve était lors de la consigne.

- type : int

Permet de définir la fonction mathématique que va suivre la montée en température. (Voir méthode « calcul » page 43)

- done : bool

Utiliser afin de savoir si la consigne fut bien appliquée ou non.

- host : str

Adresse IP de l'étuve sur laquelle la consigne est menée.

- port : int

Port sur laquelle se trouve l'étuve.

Exemple de données Json :

```
{"temperature":127.0,"second":52,"time":1716107107.354506,"base":0.0,"type":0,"done":true, "host":"192.168.1.73","port":502}
```

Ce fichier Json est nommé data.json et ce trouve au même niveau de répertoire que la classe Calculatrice.

Maintenant que nous avons un formatage des données nous pouvons nous attaquer à la mise en place des URL, des fonctions et pages qui leurs sont attribuées.

9.5.2.5 Mise en place des URL, des fonctions et des pages

Afin de définir toutes les URL utiles au sein de Django, il faut aller modifier le fichier `urls.py`. Au sein de ce fichier se trouve une variable de type list nommée `urlpatterns`. Il faut ajouter en son sein toutes les URL utiles aux sites qui prendront la forme d'un objet path. L'objet path se définit selon le procédé suivant : `path(arg1,arg2)` où `arg1` correspond à l'URL sous forme string qui sera entrée par l'utilisateur et `arg2` correspond à une fonction qui sera lancée lors de l'arrivée sur l'URL. Ici ces fonctions jointes à un URL seront programmées au sein d'un fichier nommé `views.py` qui se trouve dedans le répertoire de l'application pages initialisée plus tôt.

Voici mon `urlpatterns` avec les 7 URL qui lui sont attribuées :

URL	Description
<code>hostAdress/</code> ou <code>hostAdress/index/</code>	Page index
<code>hostAdress/consign/<str:host>/<int:port>/<str:temperature>/<int:second>/</code>	Consigne une température
<code>hostAdress/consign/<str:host>/<int:port>/<str:temperature>/<int:second>/<int:type>/</code>	Consigne une température et définit un type de courbe
<code>hostAdress/read/<str:host>/<int:port>/</code>	Lis les informations sur la consigne et la température de l'étuve.
<code>hostAdress/readTemperature/<str:host>/<int:port>/</code>	Lis la température de l'étuve.
<code>hostAdress/readConsign/</code>	Lis la consigne actuelle

Afin de gérer les mauvaises URL, j'ai rajouté la ligne `handler404 = 'pages.views.custom_404'` qui permet d'exécuter une fonction spéciale lors d'une URL non reconnues parmi celles présentes dans l'`urlpatterns`.

9.5.2.6 Fonction de redirection d'URL

Maintenant que les URL sont paramétrées, il faut que je code la fonction associée. Pour ce faire, il faut les coder au sein du fichier `views.py` du répertoire `pages`. Dans ce fichier se trouve 7 différentes fonctions. Pratiquement chacune de ces fonctions renvoie vers une page web nommée « `result.html` » qui affiche tout simplement quel type de requête fut demandée et les données associées.

9.5.2.6.1 `connecto`

Cette fonction me permet de me connecter à l'étuve et d'éviter des erreurs en cas d'échec de connexion. Pour ce faire elle passe par une variable globale nommée « `client` » qui devient un objet de classe étuve initialisé au port et adresse IP donnés. Alors je tente d'y lire la température, en cas d'échec je retourne `False`, si je réussis je retourne `True`.

9.5.2.6.2 `index`

Renvoie à une page index qui n'est pas encore finie.

9.5.2.6.3 `consign`

Formate via un dictionnaire les différentes données qui seront par la suite affichée sur la page internet. Les données sont les suivantes :

- « `host` » : l'adresse IP de l'étuve
- « `port` » : port de l'étuve
- « `consignTemperature` » : température consignée
- « `consignSecond` » : en combien de seconde est-elle consignée
- « `type` » : comment la température monte elle en température (Voir méthode calcul page 43)
- « `statut` » : Si réussite : 200, si erreur avec serveur Django : 500, si erreur avec étuve : 400
- « `message` » : Permet de savoir ce qui s'est passé avec plus d'informations.

Une fois les données préparées, la fonction tente de se connecter à l'étuve, en cas d'échec renvoie une erreur 400. Puis tente d'ouvrir le fichier « `data.json` » afin d'y enregistrer les valeurs de consignes dedans puis le ferme. En cas de réussite une valeur 200 est retournée indiquant la réussite sinon une valeur 500 est retournée indiquant un problème avec le serveur Django. Les données sont alors redirigées vers la page « `result.html` ».

9.5.2.6.4 `read`

Formate via un dictionnaire les différentes données qui seront par la suite affichée sur la page internet. Les données sont les suivantes :

- « `host` » : l'adresse IP de l'étuve
- « `port` » : port de l'étuve
- « `temperature` » : température actuelle de l'étuve
- « `consignTemperature` » : température consignée
- « `consignSecond` » : en combien de seconde est-elle consignée
- « `statut` » : Si réussite : 200, si erreur avec serveur Django : 500, si erreur avec étuve : 400
- « `message` » : Permet de savoir ce qui s'est passé avec plus d'informations.

Une fois les données préparées, la fonction tente de se connecter à l'étuve, en cas d'échec renvoie une erreur 400. Puis tente de récupérer les informations concernant la consigne dans le fichier « `data.json` » et aussi la valeur de température de l'étuve puis place ces informations-là dans le dictionnaire. Si c'est réussi une valeur 400 est retournée sinon une valeur 500 est retournée. Les données sont alors redirigées vers la page « `result.html` ».

9.5.2.6.5 readTemperature

Formate via un dictionnaire les différentes données qui seront par la suite affichée sur la page internet. Les données sont les suivantes :

- « host » : l'adresse IP de l'étuve
- « port » : port de l'étuve
- « temperature » : température actuelle de l'étuve
- « statut » : Si réussite : 200, si erreur avec serveur Django : 500, si erreur avec étuve : 400
- « message » : Permet de savoir ce qui s'est passé avec plus d'informations.

Une fois les données préparées, la fonction tente de se connecter à l'étuve, en cas d'échec renvoie une erreur 400. Puis tente de récupérer la valeur de température de l'étuve puis place cette information-là dans le dictionnaire. Si c'est réussi une valeur 400 est retournée sinon une valeur 500 est retournée. Les données sont alors redirigées vers la page « result.html ».

9.5.2.6.6 readConsign

Formate via un dictionnaire les différentes données qui seront par la suite affichée sur la page internet. Les données sont les suivantes :

- « host » : l'adresse IP de l'étuve
- « port » : port de l'étuve
- « consignTemperature » : température consignée
- « consignSecond » : en combien de seconde est-elle consignée
- « statut » : Si réussite : 200, si erreur avec serveur Django : 500, si erreur avec étuve : 400
- « message » : Permet de savoir ce qui s'est passé avec plus d'informations.

Une fois les données préparées, la fonction tente de se connecter à l'étuve, en cas d'échec renvoie une erreur 400. Puis tente de récupérer les informations concernant la consigne dans le fichier « data.json » puis place ces informations-là dans le dictionnaire. Si c'est réussi une valeur 400 est retournée sinon une valeur 500 est retournée. Les données sont alors redirigées vers la page « result.html ».

9.5.2.6.7 custom_404

Cette fonction est en cas d'URL non conforme à l'API, elle renvoie vers « result.html » qui affiche alors le dictionnaire suivant :

```
{"result":{"statut":404,"message":"Page not found, check URL"}}
```

9.5.2.6.8 Explication redirection d'URL

Chacune de ces fonctions finissent par une ligne ressemblant à la chose suivante :

```
return render(request_iter,"page.html",context={"property":value})
```

Cette ligne permet au navigateur web de recevoir une page HTML contenant les valeurs qui ont été calculées par mes fonctions. Ces pages HTML se trouvent au sein d'un répertoire que j'ai créé à la racine du projet nommé « templates ». Seul deux pages s'y trouve, « index.html » et « result.html », index.html n'étant pas finie, seul result.html est opérationnel. Result.html est un fichier ne contenant pas beaucoup de code seul deux lignes s'y trouvent :

```
<title>{{ title }}</title>
{{ result }}
```

Cette première ligne me permet de changer le titre de ma page en celui de la requête API demandé et la seconde ligne permet d'afficher les résultats directement dans la page. Ces deux lignes là me permettent d'avoir une page très versatile permettant de répondre à toutes les requêtes directement en récupérant juste leurs données.

Maintenant que mon API peut récupérer, stocker, afficher de la donnée, elle ne peut pas pour autant utiliser ces données, c'est alors que j'ai codé un programme me servant de calculatrice pour ces données.

9.5.2.7 Variables d'environnement et données

Afin de rendre mon API plus paramétrable j'ai rajouté un fichier .env à la racine du projet de groupe qui possède en son sein plusieurs variable :

```
[SETTINGS]

HOST = STR
PORT = INT
SLEEP = INT
QUEUELENGHT = INT


[DEBUG]

API = BOOL
DJANGO = BOOL

#Default host = your local ip address
#Default port = 5555
#This is an example, those variables are used by the API and Django Server.
```

9.5.2.7.1 Paramètres de l'API

Les variables HOST et PORT permettent de définir l'adresse IP et le port sur lesquelles va se lancer le serveur Django. La valeur « default » permet que le serveur lance sur la même adresse IP que votre machine, sur le port 5555. Il se trouve aussi deux autres variables, SLEEP et QUEUELENGHT. Ces deux variables permettent de choisir le temps d'attente entre deux envois de température à l'étuve et le nombre de valeur permettant de vérifier la stabilité de la température (pour plus d'informations concernant les deux dernières variables voir méthode « experimentation » page 41 ou « Algorithme méthode api() » page 53)

9.5.2.7.2 Debug de l'API

Pour ce qui est du debug, deux variables sont paramétrables. Si API est passée à « true » alors un serveur modbus servant de simulateur d'étuve se lancera possédant la même adresse IP que votre machine sur le port 502. Si DJANGO est passé à « true » alors le serveur passera dans le mode Debug intégré au serveur permettant d'obtenir plus d'informations en cas d'erreur de programme sur le serveur Django.

9.5.2.7.3 configLecter

Afin de pouvoir lire ces données-là, j'ai codé au sein d'un fichier nommé configLecter une fonction nommée « set » permettant de récupérer au sein d'un dictionnaire toutes les valeurs du fichier .env.

9.5.2.8 Mise en place d'une calculatrice

Evoquée plus tôt durant la bibliothèque de classe, j'ai codé un algorithme permettant à mon API de faire évoluer la température comme souhaité par l'utilisateur.

9.5.2.8.1 Information concernant la calculatrice

L'algorithme se trouve au sein d'une classe nommée « Calculatrice » évoquée à la page 43. Je vais ici me focaliser sur l'explication de l'algorithme qui se situe derrière la méthode « api() » de la classe. Le reste de la classe a déjà été expliqué au sein de la page 43, si une information manque à cette partie, il est possible que cette information se situe au sein de cette page. Quand bien même je tâcherais d'expliquer l'entièreté de l'algorithme sans besoin de retourner à cette page.

9.5.2.8.2 Méthode api()

9.5.2.8.2.1 Initialisation de la calculatrice

Lors du lancement de l'algorithme, quelques attributs de la classe sont initialisés. « infos » récupère les données du fichier .env via la fonction « set » présent dans le fichier « configLecter » (Voir page 52 pour des informations concernant configLecter). Suite à ça, l'attribut pathToJson est lui initialisé au chemin vers le fichier data.json. Ce même fichier est initialisé avec les valeurs de base (tout à 0 sauf « done » = True afin d'éviter que l'algorithme croie qu'une consigne vient d'être donnée). Alors j'initialise l'attribut verif en tant que « Queue » de taille « infos[queueLenght] » qui est la valeur donnée au sein du fichier .env sur la ligne :

```
QUEUELENGHT = INT
```

9.5.2.8.2.2 Boucle infinie : algorithme

Une fois les données initialisés, une boucle infinie se lance qui consiste à l'ensemble de l'algorithme. Tout d'abord l'on lit les données présentes au sein du fichier « data.json » et vérifions si « done == False » car lorsqu'une nouvelle consigne est donnée « done » passe de True à False durant l'entièreté du respect de la nouvelle consigne. On se connecte à l'étuve en initialisant l'attribut « etuve » en tant qu'objet de la classe Etuve. Pour la connexion l'on récupère les informations d'adresse IP et de port au sein du fichier « data.json ». Puis l'on ajoute à notre Queue de vérification la température actuelle de l'étuve et consignons à l'étuve la température qu'elle doit atteindre via la méthode « calcul() » de la classe. Enfin nous vérifions la stabilité de la température via la méthode « delta() » de la Queue, c'est cette méthode qui permet de vérifier si les dernières températures de l'étuve sont stables et proches de la valeur finale. Si la température est stable et que nous sommes arrivés au temps indiqué par l'utilisateur alors je donne comme dernière consigne d'atteindre la valeur finale (fonction logarithme proposée par la méthode calcul() n'atteint jamais exactement la valeur finale donc je met à la valeur finale directement comme ça je suis sûre qu'elle l'atteigne) et j'inscris dans « data.json » que la consigne est terminée. A chaque itération de la boucle je supprime la connexion à l'étuve afin d'éviter la surcharge de connexion et j'attends le nombre de seconde défini par la ligne : SLEEPT = INT du fichier .env.

9.5.2.8.2.3 Lancement automatique de l'algorithme

Afin de faciliter le lancement de l'algorithme, si le fichier contenant la classe « Calculatrice » est lancé en tant que programme principal alors l'algorithme est lancé tout seul. Ceci est possible par le biais de la ligne `if __name__ == "__main__":` qui vérifie si c'est le programme principal ou non.

9.5.2.9 *Threading et lancement de l'API*

Un problème provient de la façon dont j'ai codé mon API car l'entièreté de son fonctionnement ne peut tenir sur un thread. Il me faut un thread permettant au serveur Django de fonctionner, un pour permettre à mon algorithme de tourner et dans le cas du test sur une fausse étuve alors il faut lancer le serveur modbus. Pour ce faire, un fichier « launchApi.py » s'occupe de lancer ces threads automatiquement par le biais de la bibliothèque native à Python « subprocess » qui me permet de lancer les trois programmes. Il permet notamment d'installer toutes les bibliothèques nécessaires au fonctionnement de l'API, via un fichier requirements.txt contenant toutes les bibliothèques nécessaires et la commande :

```
pip install -r requirements.txt
```

Qui alors va installer toutes les bibliothèques présentes au sein du fichier requirements.txt et si elles le sont déjà alors il lancera le programme tout de suite.

9.5.3 Conclusion

9.5.3.1 Cahier des charges

Fonctions	Description	Validée ?
Serveur Django	Mise en place d'un serveur Django	OK
URL d'accès	Pouvoir accéder aux différentes URL nécessaire pour l'utilisation de l'API	OK
Lecture de la température d'une étuve	Pouvoir lire la température actuelle de l'étuve	OK
Lecture de la consigne actuelle	Pouvoir lire la dernière consigne donnée à l'API	OK
Consigner une température	Pouvoir consigner une température avec une durée	OK
Evolution de la température en fonction du temps	Température consigner à l'étuve qui évolue en fonction du temps afin de suivre une courbe.	OK
Installation automatique	Bibliothèques python nécessaire installée automatiquement dès le premier lancement.	OK

9.5.3.2 Pour aller plus loin

Afin d'améliorer l'API je peux, faire une page index qui un accès facile aux différentes fonctionnalités de l'API.

9.5.3.3 Bilan

Ce fut fort intéressant de travailler sur un framework sur lequel je n'avais jamais travaillé car c'était ma première fois. J'ai pu apprendre tout un tas de choses sur ce qu'est une API et ce fut fort enrichissant.

10 Candidat 3 : KORCHIA Maxime, PILOTAGE DE L'ETUVE (Python)

Mon objectif durant ce projet était de piloter l'étuve, cela veut dire pouvoir lire la température actuelle ainsi que la température cible. Il faudra également être en mesure de redéfinir la température cible.

Durant ce projet j'ai travaillé en binôme avec Rayan ZIANI, qui est dans le groupe PHP, nous avons tous les deux à comprendre le fonctionnement de l'étuve. Nous nous sommes donc entraides.

J'ai donc tout d'abord lu la documentation d'APT-COM, l'application que Piséo utilise pour piloter l'étuve afin de comprendre son fonctionnement.

Puis j'ai utilisé APT-COM afin de mieux comprendre comment cela fonctionnait en pratiquant.

Enfin, pendant que je piloterai l'étuve en changeant sa température cible par exemple, je devrai analyser les trames que s'échangent APT-COM et l'étuve dans l'optique de les recréer dans notre propre application.

Cette application que nous allons créer permettra donc de réguler la température de l'étuve puis de lancer automatiquement un test.

Pour ce faire j'utiliserai donc APT-COM, l'étuve Binder MK-115, WireShark et enfin Visual Studio Code pour recréer les trames en python.

10.1.1 Planning prévisionnel individuel

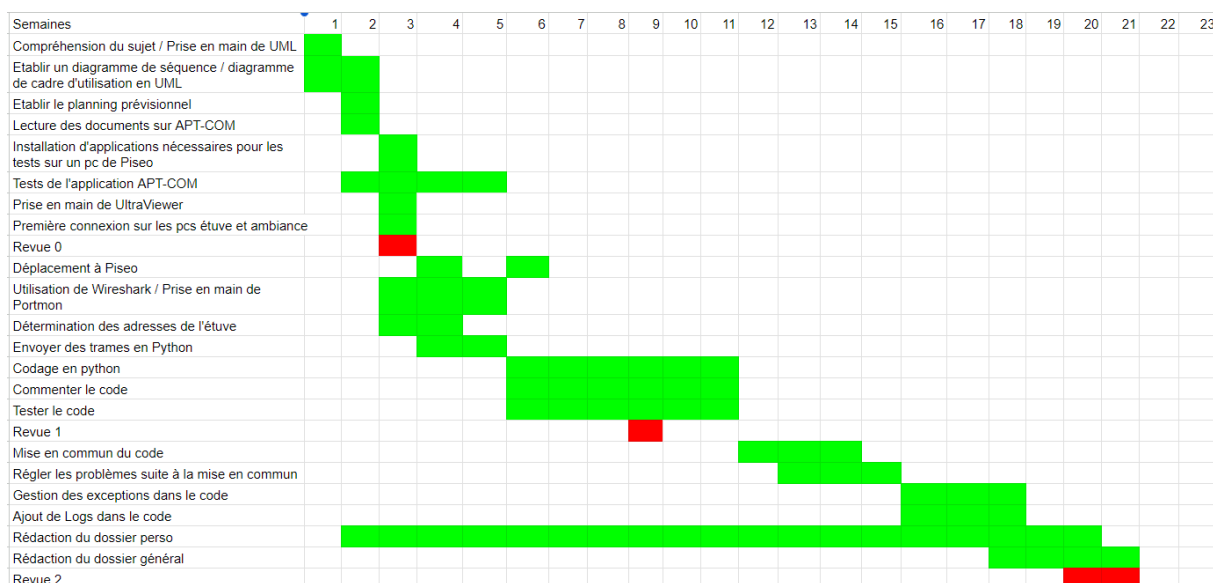


Figure 23 : Planning Maxime K.

10.1.2 Tableau des tâches

Voici mes quatre tâches principales

Tache	Description	Validée ?
Procédure d'installation de UltraViewer	Procédure d'installation pour que M. Gilot, notre professeur référent puisse l'installer sur les postes de travail que l'on a chez Piséo	OK
Utilisation de APT-COM	Compréhension et utilisation du logiciel afin de comprendre comment fonctionne l'étuve.	OK
Utilisation de Wireshark	Nous nous sommes déplacés chez Piséo et avons récupéré les trames que s'échangent APT-COM et l'étuve, dans l'optique de les analyser et les comprendre.	OK
Création de trames pour piloter l'étuve	Après avoir compris comment fonctionnaient les trames, je devrais créer un script qui permettra d'envoyer les mêmes trames que APT-COM envoie à l'étuve mais cette fois depuis notre application, par la suite je changerai le code et le mettrai sous forme de classe.	OK

10.1.3 Diagramme de classe

Voici le diagramme de ma classe Etuve

EtuvePython
-host: String -port: String
+__init__(host: String, port: String) -__send_modbus_request_read(unit_id: int, function_code: int, start_address: int, count: int) : : bytes -__send_modbus_request_write(unit_id: int, function_code: int, start_address: int, value: float) : : bytes -__parse_modbus_float_response(response: bytes) : : float +readTemperature() : : float +readSetPoint() : : float +setTemperature(value: float) : : bytes

10.2 Tâche 1 – Procédure d'installation de UltraViewer

L'étuve étant à l'entreprise Piséo, nous devons trouver un moyen de pouvoir y accéder tout le temps afin d'effectuer des tests.

Nous avons donc envisagé d'installer un logiciel permettant de se connecter à distance sur un poste, étant donné que nous ne pouvions pas aller sur place, M. Gilot, notre professeur référent, m'a demandé de faire une procédure d'installation du logiciel Ultraviewer pour qu'il puisse l'installer sur les PCs à Piséo.

10.2.1 Eléments de réalisation

J'ai eu besoin de Word ainsi que de mon PC

Les captures d'écran qui suivent sont uniquement les moments phares de la procédure.

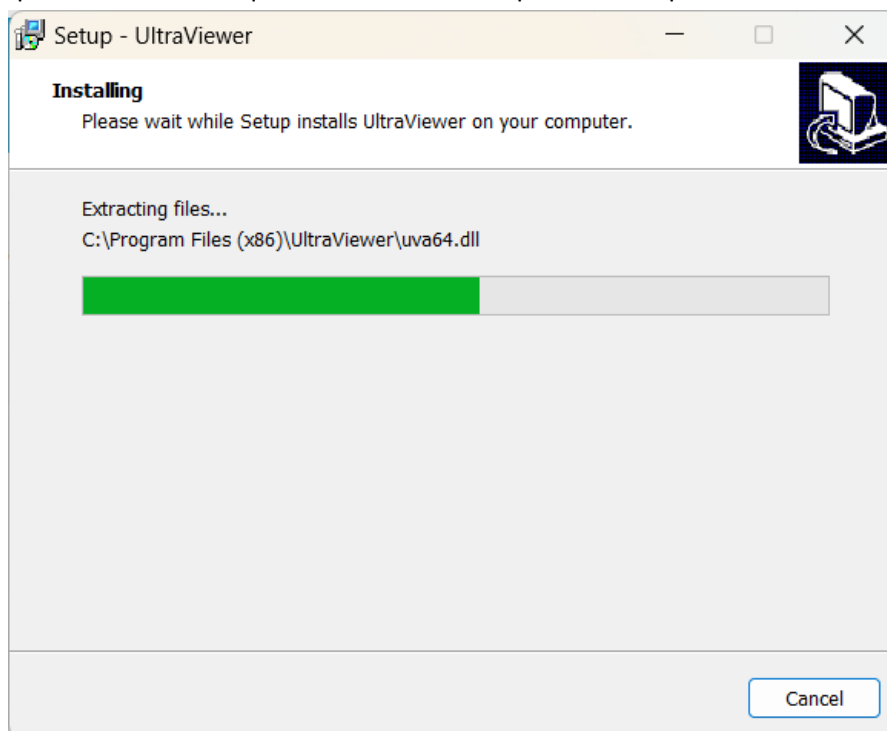


Figure 24 : Installation du setup

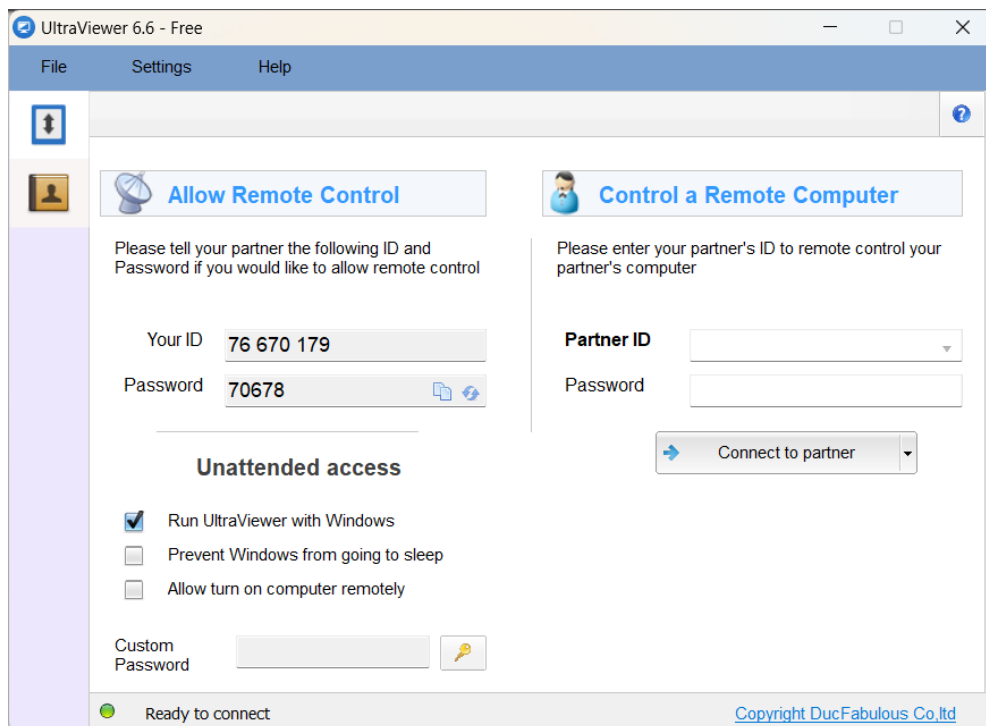


Figure 26 : Affichage d'UltraViewer

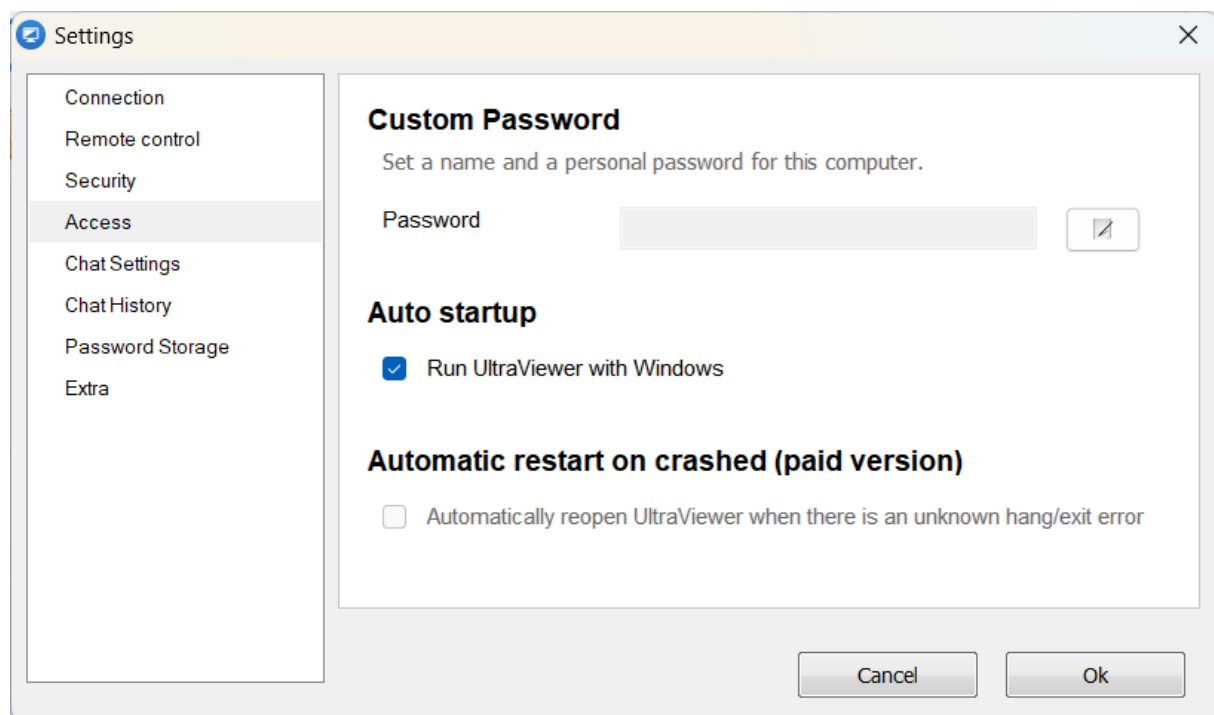


Figure 25 : Settings Ultraviewer

Cocher « Run UltraViewer with Windows » pour qu'il se lance dès que le poste s'allume.

Un mot de passe personnalisé qui ne change pas pour pouvoir se connecter.

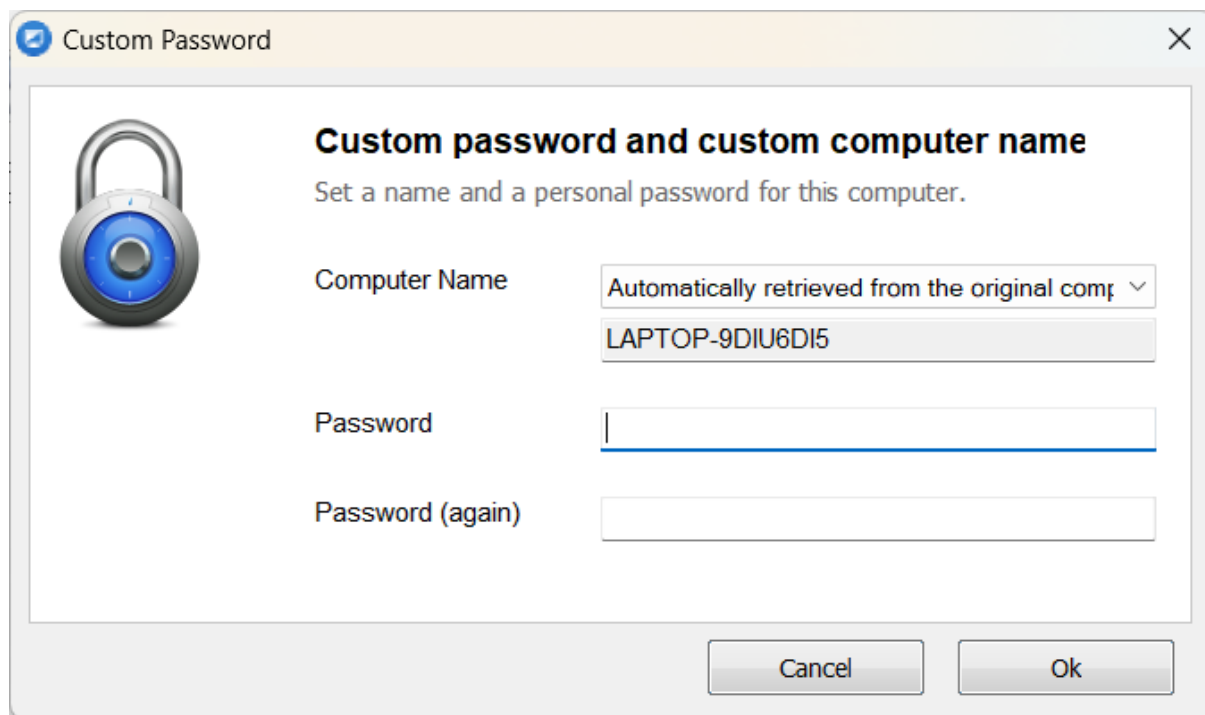


Figure 27 : Configuration mot de passe Ultraviewer

10.2.2 Bilan

Voici un bilan de ma première tâche personnelle

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Procédure UltraViewer	Rédaction d'une procédure d'installation du logiciel UltraViewer	M. Gilot a pu installer directement le nécessaire sur les postes qui ont été laissés chez Piséo	Conforme	OK

10.3 Tâche 2 - Utilisation de APT-COM

10.3.1 Introduction

Afin de préparer notre intervention chez Piséo, nous avons eu besoin de nous former à APT-COM.

Nous avons tout d'abord lu la documentation de APT-COM qui nous a été fournie.

Puis nous avons testé l'application, premièrement nous avons créé une machine qui ferait office d'étuve, la création nécessite une adresse IP et MAC, un nom, un numéro de série.

Enfin nous sommes allés chez Piséo et nous étions donc prêt pour récupérer un maximum d'informations sur place pour ensuite pouvoir avancer quand on sera sur nos temps de cours.

10.3.2 Lecture et compréhension de la documentation de APT-COM

Dans notre projet nous allons principalement utiliser le protocole Modbus, ce protocole permet la communication entre plusieurs dispositifs électroniques, en l'occurrence la communication que notre PC et l'étuve font, est le Modbus TCP.

Les messages Modbus contiennent une adresse de registre, un code de fonction, les données, et des contrôles.

Grâce à la documentation de l'étuve j'ai pu créer un Excel avec les numéros de registres en décimal / hexadécimal et leur fonction :

Value Name	Base address (hex)	Base address (decimal)	Data Type	Access	Note
Process value 1	1004	4100	float	R/O	Temperature
Process value 4	100A	4106	float	R/O	Humidity
Set point 1	10B2	4274	float	R/O	Temperature
Set point 2	10B4	4276	float	R/O	Humidity
Set point 3	10B6	4278	float	R/O	Fan speed
Set point 1 manual	114C	4428	float	R/W	Temperature
Set point 2 manual	114E	4430	float	R/W	Humidity
Set point 3 manual	1150	4432	float	R/W	Fan speed
Track read	1292	4754	int	R/O	Bit 0-15 = Track 0-15
Track manual	1158	4440	int	R/W	Bit 0-15 = Track 0-15
Program number	1147	4423	int	R/W	
Start Section Number	1148	4424	int	R/W	
Start program	1149	4425	bool	R/W	Write value = 1
Stop program	114A	4426	bool	R/W	Write value = 2
Pause program	114B	4427	bool	R/W	Write value = 3

Figure 28 : Numéros de registre

Cela me permettra plus tard de vérifier la véracité de la documentation en utilisant Wireshark.

10.3.3 Création de machine sur APT-COM

Pour le matériel, nous avons besoin d'un PC ainsi que de l'application APT-COM installée dessus.

L'application APT-COM se présente comme cela :

Pour

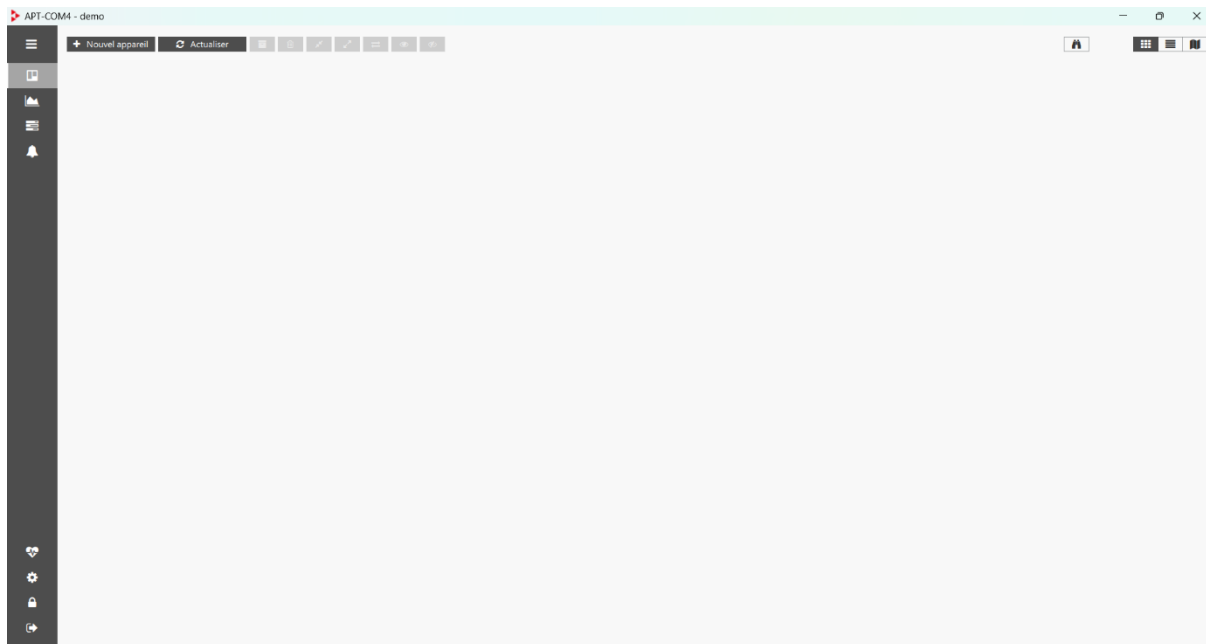
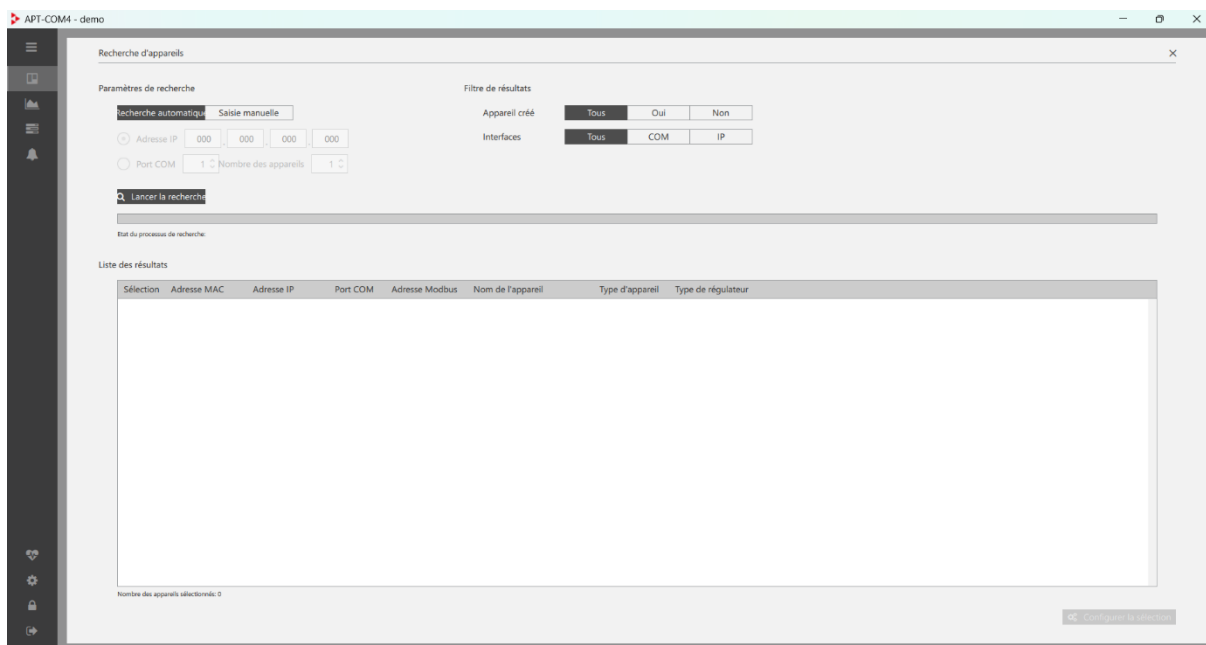


Figure 29 : APT-COM Index

effectuer un quelconque test il faut tout d'abord créer une machine ou en trouver une automatiquement sur le réseau, en l'occurrence le nom de machine est donné comme tel par APT-COM. Une machine est par exemple une étuve.

Dans
cas
créé



notre
on l'a

Figure 30 : APT-COM Recherche machine

manuellement étant donné que nous n'avons pas l'étuve au lycée.

Pour ce faire je coche « saisie manuelle » et entre une adresse IP quelconque.



The screenshot shows a software interface for APT-COM. At the top, there are two tabs: 'Recherche automatique' (highlighted) and 'Saisie manuelle'. Below the tabs, there are two radio buttons. The first is labeled 'Adresse IP' and is selected, followed by four input boxes containing the values '192', '168', '1', and '200' separated by dots. The second radio button is labeled 'Port COM' and is unselected, followed by a spinner box containing the value '1'. To the right of the 'Port COM' section is the text 'Nombre des appareils' followed by another spinner box containing the value '1'. At the bottom, there is a dark button with a white plus sign and the text 'Ajouter'.

Figure 31 : APT-COM Saisie adresse IP

Elle apparait désormais dans la liste des résultats (soit la liste des étuves reconnues par APT-COM) :

Sélection	Adresse MAC	Adresse IP	Pt
<input checked="" type="checkbox"/>		192.168.1.200	

Figure 32 : APT-COM Machine trouvée

Néanmoins il manque une partie de la configuration, nous devons donc continuer de donner des informations.

Infos sur l'appareil Sélection de l'appareil Vérification

Informations sur l'appareil

Nom de l'appareil: Test

Adresse IP: 192.168.1.200

Adresse MAC: aa:aa:aa:aa:aa:aa

Numéro de série: 11-11111

Figure 33 : APT-COM Configuration

L'adresse MAC ainsi que le numéro de série sont mises arbitrairement, et ont uniquement pour but de finir la création manuelle de la machine.

Nous devons désormais choisir le type d'appareil utilisé, en l'occurrence nous avons pris un appareil qui semblait être similaire à notre étuve:

No d'article	Modèle	Taille	Version	Tension	Régulateur
9020-0265	AI	300	E2.1	200V-240V	BCW
9020-0266	AI	300	E2.1	200V-240V	BCW
9020-0318	AI	300	E2.2	200V-240V	BCW
9020-0319	AI	300	E2.2	200V-240V	BCW
9120-0265	AI	300	E2.1	200V-240V	BCW
9120-0266	AI	300	E2.1	200V-240V	BCW
9120-0318	AI	300	E2.2	200V-240V	BCW
9120-0319	AI	300	E2.2	200V-240V	BCW
9010-0073	BD	400	E2	230V	R3
9010-0073	BD	400	E2	230V	R3.2
9010-0074	BD	720	E2	230V	R3
9010-0074	BD	720	E2	230V	R3.2
9010-0081	BD	53	E2	230V	R3
9010-0081	BD	53	E2	230V	R3.2
9010-0088	BD	115	E2	230V	R3
9010-0088	BD	115	E2	230V	R3.2
9010-0095	BD	240	E2	230V	R3

Figure 34 : APT-COM Choix appareil

10.3.4 Tests sur la machine créée



Figure 35 : APT-COM Infos appareil

Nous

avons désormais une machine de créée, elle ne peut néanmoins pas fonctionner mais ces étapes seront utiles pour l'installation de l'étuve sur APT-COM chez Piséo.

La machine étant créée nous pouvons désormais créer un programme pour automatiser les montées en température

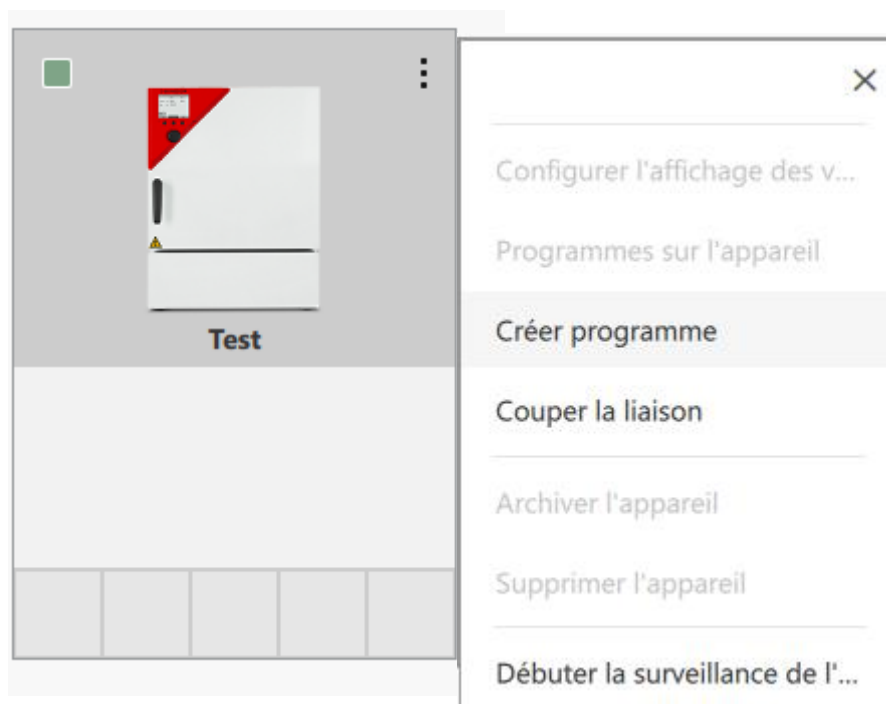


Figure 36 : APT-COM Créer programme

Voici à quoi ressemble les programmes nous pouvons, sur le côté droit de la page changer la température selon le temps d'activité de l'étuve, en l'occurrence nous avons laissé le programme par défaut

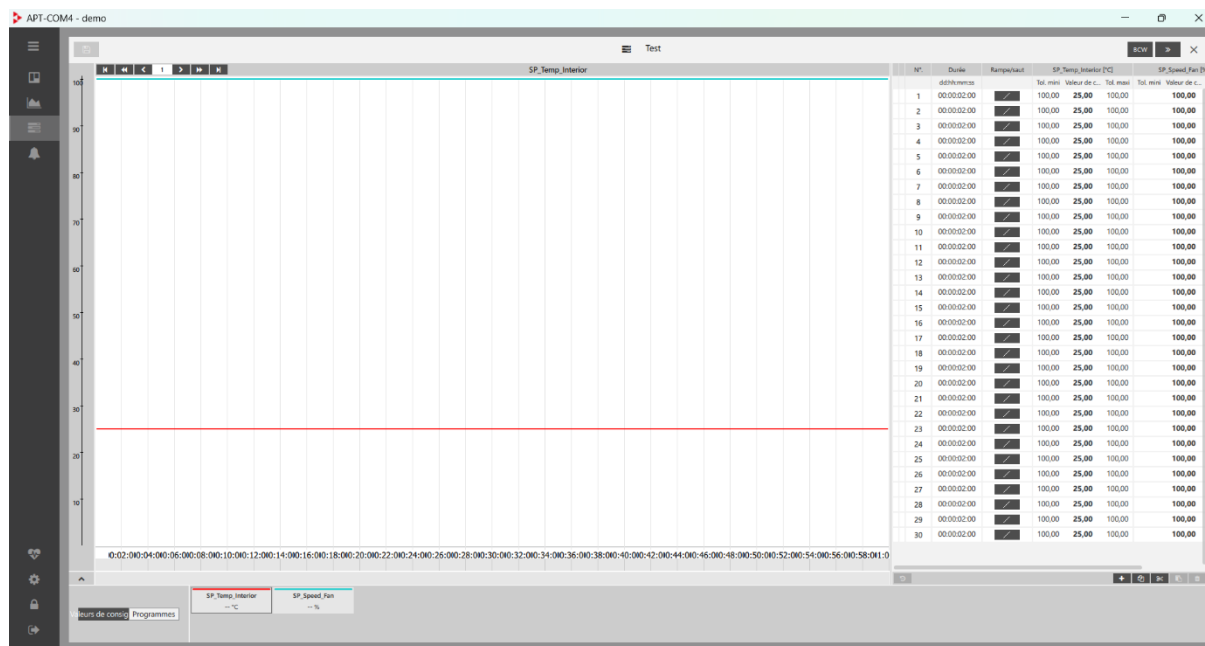


Figure 37 : APT-COM Programmes

10.3.5 APT-COM chez PISEO

Après avoir pris en main APT-COM au lycée, nous allons désormais faire de même, mais cette fois sur la vraie étuve.



Figure 38 : APT-COM Signaux disponibles

Voici les plusieurs signaux disponibles que nous pouvons visionner à la création de l'étuve, nous allons principalement utiliser les « Temp »



Figure 39 : APT-COM Températures

29.9°C signifie la température actuelle à l'intérieur de l'étuve,

30°C signifie la température cible

En faisant un double-clic sur la température cible nous pouvons modifier cette dernière après avoir mis un commentaire de confirmation de changement.

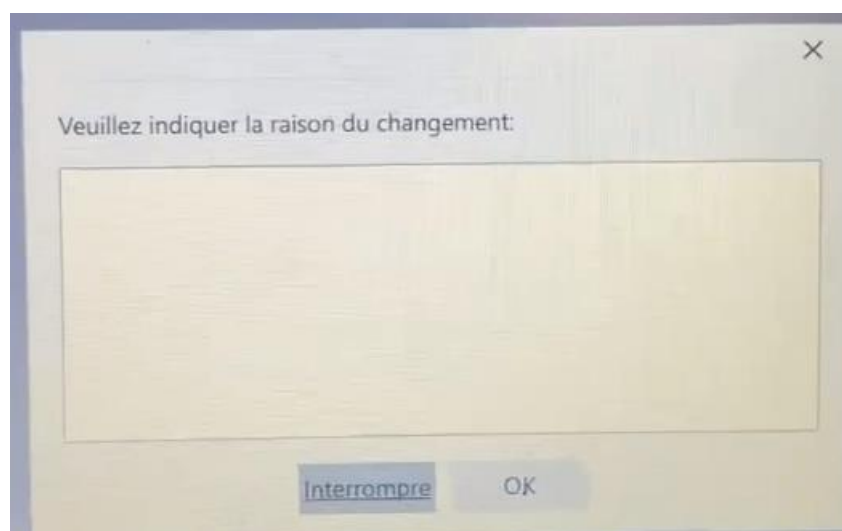


Figure 40 : APT-COM Confirmation

10.3.6 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Utilisation de APT-COM	Création d'une machine et tests sur cette dernière puis tests sur l'étuve	Se former sur APT-COM pour récupérer les informations utiles chez Piséo	Conforme	OK

Cette mission m'a permis d'apprendre et de comprendre l'utilisation d'APT-COM,

Elle m'a également permis de récupérer les trames nécessaires pour ma future analyse.

10.4 Tâche 3 - Wireshark

10.4.1 Introduction

Dans cette partie je vais utiliser Wireshark, un logiciel permettant d'analyser les échanges qui se produisent sur un réseau. Nous allons principalement nous en servir pour capturer des paquets et analyser des protocoles (Modbus).

Notre objectif sera donc de récupérer les trames que s'échangent APT-COM et l'étuve afin d'en comprendre le fonctionnement.

Nous avons besoin de récupérer les trames que s'échangent l'étuve et APT-COM dans l'optique de recréer une application similaire qui permettra d'ordonner à l'étuve d'atteindre des températures cibles ainsi que de lire la température par nos propres moyens (sans utiliser APT-COM).

Pour ce faire nous utiliserons les trames récupérées pendant nos tests précédents, chez Piséo

10.4.2 Eléments de réalisation

Nous aurons donc besoin d'être chez Piséo afin d'utiliser l'étuve, ainsi que le PC qui y est situé (un PC de l'ORT emmené au préalable pour le projet) et y utiliser Wireshark et APT-COM.

Une fois sur place nous avons branché notre PC à l'étuve via une connexion RJ45.

Nous avons modifié la température cible ainsi que lu la température cible et actuelle.

Tout ça dans l'optique de créer plusieurs trames pour ensuite pouvoir les analyser au lycée.

Les protocoles principaux sont Modbus et TCP, mais nous allons surtout nous concentrer sur le protocole Modbus.

10.4.3 Analyse de trames

Mon objectif était maintenant de voir si les trames Wireshark donnaient les mêmes valeurs de registre que dans la documentation selon ce que je faisais avec APT-COM.

Sur cette capture nous pouvons observer le numéro de registre 4428 qui était présent dans la documentation en tant que Set Point Manual pour la température, soit en d'autres termes la température cible que l'on voudra atteindre.

Le problème que nous avons rencontré était que la valeur qui était associé au registre était une valeur qui ne ressemblait pas à une température (39322)

Nous avons donc cherché un moyen de mieux lire les données et nous avons trouvé dans préférence de protocole, la possibilité de changer la façon de lire les trames.

No.	Time	Source	Destination	Protocol	Length	Info
11	12.652031	172.17.100.201	172.17.100.108	Modbus...	66	Query: Trans: 158; Unit: 255, Func: 4: Read Input Registers
13	12.694294	172.17.100.108	172.17.100.201	Modbus...	65	Response: Trans: 158; Unit: 255, Func: 4: Read Input Registers
29	25.175977	172.17.100.201	172.17.100.108	Modbus...	71	Query: Trans: 159; Unit: 255, Func: 16: Write Multiple Registers
31	25.214592	172.17.100.108	172.17.100.201	Modbus...	66	Response: Trans: 159; Unit: 255, Func: 16: Write Multiple Registers

Frame 29: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface 0

Ethernet II, Src: Hewlett-Packard (08:00:27:00:00:00), Dst: MKJuch (08:00:27:00:00:00)

Internet Protocol Version 4, Src: 172.17.100.201, Dst: 172.17.100.108

Transmission Control Protocol, Src Port: 55824, Dst Port: 502, Seq: 13, Len: 71

Modbus/TCP

Modbus

.001 0000 = Function Code: Write Multiple Registers (16)

Reference Number: 4428

Word Count: 2

Byte Count: 4

Register 4428 (UINT16): 39322

Register 4429 (UINT16): 16935

0000 00 0c d8 0a 8d 48 70 5a 0f 85 07 04 08 00 45 00HpZ.....E

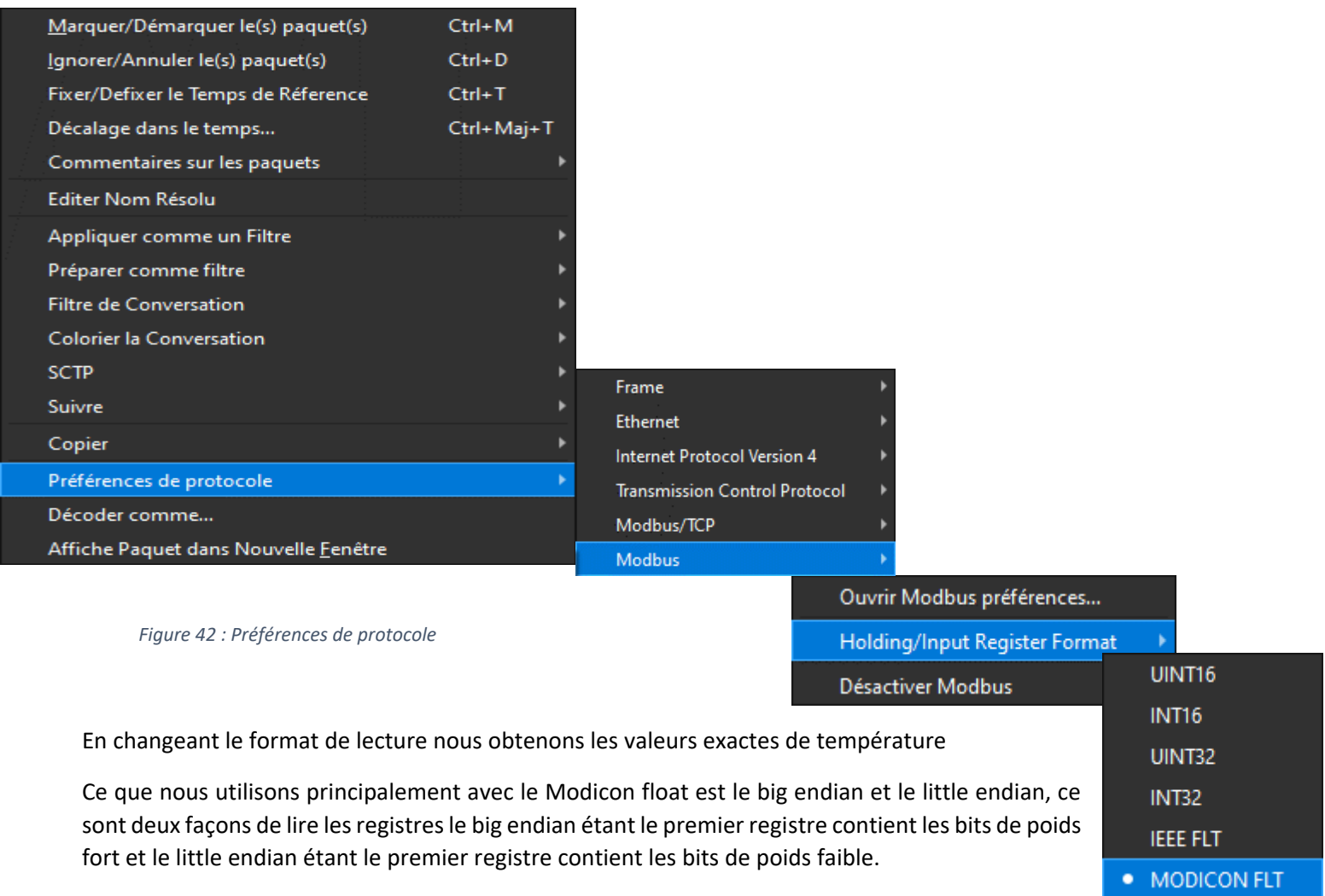
0010 00 39 78 39 40 00 80 06 00 00 ac 11 64 c9 ac 11 ..9x9@.....d...

0020 64 6c da 10 01 f6 e6 0d e4 07 f7 2a 12 f0 50 18 d!.....*...P

0030 f9 64 21 84 00 00 00 9f 00 00 00 0b ff 10 11 4c d!.....l

0040 00 02 04 99 9a 42 27B'

Figure 41 : Trames APT-COM



11	12.652031	172.17.100.201	172.17.100.108	Modbus...	66	Query: Trans: 158; Unit: 255, Func: 4: Read Input Registers
13	12.694294	172.17.100.108	172.17.100.201	Modbus...	65	Response: Trans: 158; Unit: 255, Func: 4: Read Input Registers
29	25.175977	172.17.100.201	172.17.100.108	Modbus...	71	Query: Trans: 159; Unit: 255, Func: 16: Write Multiple Registers
31	25.214592	172.17.100.108	172.17.100.201	Modbus...	66	Response: Trans: 159; Unit: 255, Func: 16: Write Multiple Registers

Frame 29: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on i	0000	00 0c d8 0a 8d 48	70 5a 0f 85 07 04 08 00 45 00HpZ
Ethernet II, Src: HewlettPacka_85:07:04 (70:5a:0f:85:07:04), Dst: MKJuch	0010	00 39 78 39 40 00 80 06	00 00 ac 11 64 c9 ac 11	..9x9@...
Internet Protocol Version 4, Src: 172.17.100.201, Dst: 172.17.100.108	0020	64 6c da 10 01 f6 e6 0d	e4 07 f7 2a 12 f0 50 18	d!..... ..*
Transmission Control Protocol, Src Port: 55824, Dst Port: 502, Seq: 13,	0030	f9 64 21 84 00 00 00 9f	00 00 00 0b ff 10 11 4c	..d!..... ..L
Modbus/TCP	0040	00 02 04 99 9a 42 27	B'

Modbus	
0001 0000 = Function Code: Write Multiple Registers (16)	
Reference Number: 4428	
Word Count: 2	
Byte Count: 4	
Register 4428 (Modicon Float): 41,900002	

Figure 43 : Trames de lecture de température cible

En l'occurrence la température cible a été établie à 41.9°

Nous avons donc récupéré le bon registre pour écrire la température cible, il ne manquait plus qu'à trouver celui pour lire la température cible et celui pour lire la température actuelle qui devraient être normalement respectivement 4274 et 4100.

Nous sommes donc à la recherche des registres énoncés précédemment parmi les trames que nous avons capturées.

```

> Frame 24: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{44C5DD63-1815-4385-8F55-6A121B17280E}, :
> Ethernet II, Src: MKJuchheim_0a:8d:48 (00:0c:d8:0a:8d:48), Dst: HewlettPacka_85:07:04 (70:5a:0f:85:07:04)
> Internet Protocol Version 4, Src: 172.17.100.108, Dst: 172.17.100.201
> Transmission Control Protocol, Src Port: 502, Dst Port: 55824, Seq: 38, Ack: 49, Len: 13
> Modbus/TCP
  > Modbus
    .000 0100 = Function Code: Read Input Registers (4)
    [Request Frame: 22]
    [Time from request: 0.055534000 seconds]
    Byte Count: 4
    > Register 4274 (Modicon Float): 30,000000

```

Figure 44 : Trame lecture température cible

Nous avons commencé par chercher dans une capture de trames que nous avons renommée « 26 à 30° », nous regardons les trames une par une en ouvrant l'onglet « Modbus » pour voir les numéros de registre. Nous finissons par trouver une trame qui correspond à ce que l'on recherche et la valeur « 30 » nous valide bien que le 4274 est la lecture de la température cible.

Nous devons désormais chercher à lire la température actuelle de l'étuve, pour ce faire nous opérons comme précédemment en regardant les trames et leur contenu jusqu'à trouver le registre « 4100 ».

Après avoir cherché dans toutes les trames le registre 4100 nous nous sommes rendu compte qu'il n'y avait pas de n° de registre 4100 dans nos trames, nous avons donc opéré d'une façon différente qui était d'essayer de trouver une corrélation entre la valeur des registres et la température que nous cherchons à atteindre dans la capture de trame.

Dans ce cas nous avons une capture de trames de 26° à 30° :

```

> Frame 344: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{44C5DD63-1815-4385-8F55-6A121B17280E}, :
> Ethernet II, Src: MKJuchheim_0a:8d:48 (00:0c:d8:0a:8d:48), Dst: HewlettPacka_85:07:04 (70:5a:0f:85:07:04)
> Internet Protocol Version 4, Src: 172.17.100.108, Dst: 172.17.100.201
> Transmission Control Protocol, Src Port: 502, Dst Port: 55824, Seq: 506, Ack: 529, Len: 13
> Modbus/TCP
  > Modbus
    .000 0100 = Function Code: Read Input Registers (4)
    [Request Frame: 342]
    [Time from request: 0.052517000 seconds]
    Byte Count: 4
    > Register 4306 (Modicon Float): 29,757389

```

Figure 45 : Trame lecture température actuelle

Le résultat 29.7 pourrait s'apparenter à la température que l'étuve avait atteinte.

Nous gardons donc en tête la possibilité que le registre « 4306 » soit le registre de lecture de température.

Pour en être sûr nous avons vérifié une autre capture.

Sur cette capture l'étuve était en train de monter à 20°

```
▶ Frame 15: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{44C5DD63-1815-4385-8F55-6A121B17280E},
▶ Ethernet II, Src: MKJuchheim_0a:8d:48 (00:0c:d8:0a:8d:48), Dst: HewlettPacka_85:07:04 (70:5a:0f:85:07:04)
▶ Internet Protocol Version 4, Src: 172.17.100.108, Dst: 172.17.100.201
▶ Transmission Control Protocol, Src Port: 502, Dst Port: 55824, Seq: 27, Ack: 37, Len: 13
▶ Modbus/TCP
▼ Modbus
  .000 0100 = Function Code: Read Input Registers (4)
    [Request Frame: 13]
    [Time from request: 0.062599000 seconds]
    Byte Count: 4
    ▶ Register 4306 (Modicon Float): 19,517708
```

Figure 46 : Trame lecture de température actuelle

Encore une fois le 4306 était proche de la température cible, nous en avons donc déduit que ce registre permettait bien de lire la température actuelle.

Nous pouvons conclure que le numéro de registre 4306 permet de lire la température actuelle.

Nous pouvons également conclure que la documentation nous donnait bien les bons numéros de registre / fonctions pour le 4428 (température cible) et 4274 (température actuelle).

Maintenant que nous connaissons les registres à utiliser pour faire ce que nous devons faire nous allons pouvoir passer à la programmation.

10.5 Tâche 4 – Programmation en python

Désormais nous devons faire un code qui permettra de recréer des trames et de les envoyer à l'étuve pour lire / changer son état.

Nous devons donc recréer 3 trames types afin de les réutiliser dans notre programme.

10.5.1 Éléments de réalisation

Pour ce faire nous avons utilisé les trames découvertes précédemment, ainsi que la bibliothèque Modbus Python.

Nous avons donc commencé à nous renseigner sur la façon d'utiliser cette bibliothèque mais après plusieurs tentatives nous avons rencontré un problème au niveau de la bibliothèque, après une mise à jour de cette dernière certaines fonctions ont été totalement retirées.

Nous avons donc cherché un autre moyen de recréer des trames.

L'idée est donc de faire un code ayant toutes les informations nécessaires à une trame pour communiquer avec l'étuve, soit coder les trames en dur.

Néanmoins nous utiliserons la bibliothèque python « socket » pour gérer les connexions et l'envoi / réception de données entre le PC et l'étuve / la fausse étuve ainsi que la bibliothèque python « struct » qui nous permettra de pack et de unpack les données de la chaîne d'octets retrouvée dans les trames.

Nous aurons besoin d'un transaction id, protocol id, la taille de l'entête modbus, l'adresse registre de début ainsi que le nombre d'éléments à lire à partir de la « start address »

```
def send_modbus_request_read(self, unit_id, function_code, start_address, count):
    transaction_id = 1
    protocol_id = 0
    length = 6 # Longueur de l'entête Modbus
    header = struct.pack('>HHH', transaction_id, protocol_id, length)
    pdu = struct.pack('>BBHH', unit_id, function_code, start_address, count)
    modbus_request = header + pdu

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((self.host, self.port))
        s.sendall(modbus_request)
        response = s.recv(1024)
```

Figure 47 : Fonction `send_modbus_request_read`

Le host et le port sont ceux de l'étuve, le `s.connect` permet de se connecter à l'étuve, le `s.sendall` permet d'envoyer notre trame faite de toute pièce et la réponse est la réponse de l'étuve qui est capturée par `s.recv(1024)`

Pour l'écriture

```
def send_modbus_request_write(self, unit_id, function_code, start_address, value):
    transaction_id = 1
    protocol_id = 0
    # Pas de décalage sur l'adresse de départ car on suppose qu'elle est déjà correcte
    # Convertir directement la valeur flottante avec l'ordre big endian pour l'alignement correct
    float_bytes = struct.pack('>f', value)
    # Gardons l'ordre des registres tel que nécessaire pour le serveur (pas d'inversion cette fois)
    reg2, reg1 = struct.unpack('>HH', float_bytes)

    length = 11 # Longueur pour l'en-tête, l'adresse, le nombre de registres, le nombre d'octets, et les données
    header = struct.pack('>HHH', transaction_id, protocol_id, length)
    # Construction du PDU avec les valeurs correctes
    pdu = struct.pack('>BBHHBH', unit_id, function_code, start_address, 2, 4, reg1, reg2)

    modbus_request = header + pdu
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((self.host, self.port))
        s.sendall(modbus_request)
        response = s.recv(1024)
```

Figure 48 : Fonction `send_modbus_request_read`

Même chose que pour la lecture, dans pdu (protocol data unit) le 2 est le nombre de registres que l'on va écrire et le 4, le nombre d'octets de données, reg1 et 2 les deux registres contenant les valeurs flottantes.

```
def parse_modbus_float_response(self, response):
    if not response or len(response) < 9:
        print("Réponse invalide ou trop courte.")
        return None

    _, _, _, _, _, byte_count = struct.unpack('>HHHBBB', response[:9])
    data = response[9:9+byte_count]

    if len(data) == 4:
        # Interprétation des registres en little-endian
        reg1, reg2 = struct.unpack('>HH', data[:4])
        # Conversion des registres en flottant, aussi en little-endian
        modicon_float = struct.unpack('<f', struct.pack('<HH', reg1, reg2))[0]
        return modicon_float
    else:
        print("Longueur de données inattendue pour un float.")
        return None
```

Figure 49 : Fonction parse_modbus_float_response

On établit une vérification de la réponse, puis on extrait les entêtes d'octets de données >HHHBBB indique que nous utilisons l'ordre big endian, les H sont en fait 3x2 octets et les B sont 3x1 octets non signés, ce qui nous fait un total de 9 octets.

9:9 signifie que les données réelles commencent après les 9 premiers octets, la quantité de données à lire est dans le byte count

>HH signifie que l'on utilise l'ordre big endian (les octets de poids fort sont stockés du plus significatif au moins significatif)

<HH signifie que l'on utilise l'ordre little endian les octets de poids faible sont stockés du moins significatif au plus significatif)

Le signe supérieur = big endian, inférieur little endian et le nombre de H signifie le nombre d'octets non signés sur lequel on va travailler.

Nous avons ensuite créé un programme simulant une étuve et répondant aux attentes de notre code précédent.

```
if function_code == 4: # Lecture de registre
    if start_address == 0x10D2: # Demande de lecture de la température actuelle
        temperature_actual = 23.5
        data = struct.pack('>HH', *float_to_modbus_registers(temperature_actual))
    elif start_address == 0x10B2: # Demande de lecture du setpoint de température
        data = struct.pack('>HH', *float_to_modbus_registers(setpoint_temperature))
    length = 3 + len(data)
    response = struct.pack('>HHHBBB', transaction_id, protocol_id, length, unit_id, function_code) + struct.pack('B', len(data)) + data

elif function_code == 16: # Écriture de plusieurs registres
    if start_address == 0x114C: # Écriture du setpoint de température
        byte_count = request[12] # Directly get the byte count from the request
        # Assuming 4 bytes of data follow for a float value, adjust if your data structure differs
        reg_values = struct.unpack('>HH', request[13:17])
        setpoint_temperature = modbus_registers_to_float(reg_values[0], reg_values[1]) # Adjust based on your server's expected byte order for floats
        response = struct.pack('>HHHBBHH', transaction_id, protocol_id, 6, unit_id, function_code, start_address, 2)
```

Figure 50 : Simulation d'étuve

Des données sont mises en dur pour les tests, l'idée étant de préparer notre code pour le tester par la suite sur l'étuve directement, la lecture de registres ne pose aucun problème mais l'écriture dans un registre pourrait en poser, typiquement cela pourrait altérer le bon fonctionnement de l'étuve, nous devons donc prendre beaucoup de précautions.

Maintenant pour tester notre classe CEtuve nous devons faire un fichier pour l'appeler :

```
from CEtuve import Etuve

mkbinder115 = Etuve("127.0.0.1",502)
print("Température cible actuelle : " + str(mkbinder115.readSetPoint()))
print("Température actuelle : " + str(mkbinder115.readTemperature()))
mkbinder115.setTemperature(65)
print("Nouvelle température cible : " + str(mkbinder115.readSetPoint()))
```

Figure 51 : Exemple d'utilisation classe Etuve

On importe Etuve, dans le cas présent j'ai laissé le localhost afin de pouvoir communiquer avec le serveur, et j'ai laissé le port de l'étuve qui est le 502

Je fais appelle aux méthode readSetPoint, Temperature et setTemperature pour tester le code, les trames sont bien créées dans Wireshark.

48	12.688803	127.0.0.1	127.0.0.1	Modbus...	56	Query: Trans:	1; Unit: 255, Func:	4: Read Input Registers
50	12.693864	127.0.0.1	127.0.0.1	Modbus...	57	Response: Trans:	1; Unit: 255, Func:	4: Read Input Registers
57	12.694588	127.0.0.1	127.0.0.1	Modbus...	56	Query: Trans:	1; Unit: 255, Func:	4: Read Input Registers
61	12.694998	127.0.0.1	127.0.0.1	Modbus...	57	Response: Trans:	1; Unit: 255, Func:	4: Read Input Registers
68	12.695453	127.0.0.1	127.0.0.1	Modbus...	61	Query: Trans:	1; Unit: 255, Func:	16: Write Multiple Registers
72	12.695747	127.0.0.1	127.0.0.1	Modbus...	56	Response: Trans:	1; Unit: 255, Func:	16: Write Multiple Registers
79	12.696105	127.0.0.1	127.0.0.1	Modbus...	56	Query: Trans:	1; Unit: 255, Func:	4: Read Input Registers
83	12.696349	127.0.0.1	127.0.0.1	Modbus...	57	Response: Trans:	1; Unit: 255, Func:	4: Read Input Registers

Figure 52 : Trame classe Etuve

Nous avons donc des trames fonctionnelles sur notre étuve de test, nous allons pouvoir le tester sur la vraie.

Nous sommes allés chez Piséo et avons effectué des tests, tous les tests étaient concluants.

Nous avons donc réussi à piloter l'étuve.

10.5.2 Bilan

Nom du test	Description du test	Résultat attendu	Résultat obtenu	Validé ?
Création de trames via le code	Recréer des trames similaires à celles que APT-COM et l'étuve s'envoie	Les trames sont fonctionnelles entre notre application et notre serveur de test, elles apparaissent sur Wireshark	Conforme	OK
Création d'une fausse étuve	Créer un serveur modbus pour simuler l'étuve	Nous pouvons effectuer des tests sur une machine ressemblant à l'étuve en toute sécurité	Conforme	OK
Changement du code	Dans la première version de mon code je faisais appel directement dans le fichier, j'ai donc dû faire passer mon fichier en tant que classe et fait un fichier d'appel de ses fonctions	Le fichier d'appel de la classe fonctionne et permet toujours de communiquer avec le serveur	Conforme	OK
Communication entre notre application et l'étuve	Le test final, celui de tester si notre code permet de piloter l'étuve	Nous pouvons lui demander sa température actuelle, sa température cible et de changer sa température cible	Conforme	OK

Nous avons donc réussi à atteindre notre objectif avec quelques difficultés telles que la bibliothèque pymodbus qui ne fonctionnait plus comme on le voulait, la compréhension des méthodes pack et unpack de la bibliothèque struct ainsi que l'inversement de certains registres pour avoir des valeurs non erronées.

Grâce à ce projet, j'ai pu me familiariser avec Python. Etant donné que je devais utiliser Wireshark j'ai pu compléter les connaissances que j'avais acquies en cours, ce projet m'a un peu redonné goût à la programmation bien que je préfère toujours l'infrastructure réseau.

11 Tableau des figures

Figure 1 : Logo de Piséo	1
Figure 2 : La photonique	6
Figure 3 : Laboratoire Piséo	7
Figure 4 : Bureau Piséo	7
Figure 5 : Contexte Piséo	8
Figure 6 : Contexte Piséo	8
Figure 7 : Contexte Piséo	9
Figure 8 : Contexte Piséo	9
Figure 9 : Diagramme de déploiement	11
Figure 10: Diagramme de cas d'utilisation	12
Figure 11: Diagramme de cas d'utilisation (API)	12
Figure 12 : Diagramme de Séquence	13
Figure 13 : Logo Discord.....	14
Figure 14 : Logo Visual Code Studio.....	14
Figure 15 : Logo Gitlab	15
Figure 16 : Logo Python	15
Figure 17 : Logo de PHP	16
Figure 18 : Prototype IHM.....	21
Figure 19 : Planning Léandre P.....	32
Figure 20 : Diagramme de classes UML (Léandre P.).....	35
Figure 21 : Création Projet apiEtuve	46
Figure 22 : Création application Pages.....	46
Figure 23 : Planning Maxime K.....	56
Figure 24 : Installation du setup	58
Figure 25 : Settings Ultraviewer.....	59
Figure 26 : Affichage d'UltraViewer	59
Figure 27 : Configuration mot de passe Ultraviewer	60
Figure 28 : Numéros de registre	61
Figure 29 : APT-COM Index	62
Figure 30 : APT-COM Recherche machine	62
Figure 31 : APT-COM Saisie adresse IP.....	63
Figure 32 : APT-COM Machine trouvée	64
Figure 33 : APT-COM Configuration.....	64
Figure 34 : APT-COM Choix appareil.....	64
Figure 35 : APT-COM Infos appareil.....	65
Figure 36 : APT-COM Créer programme	65
Figure 37 : APT-COM Programmes	66
Figure 38 : APT-COM Signaux disponibles	66
Figure 39 : APT-COM Températures	67
Figure 40 : APT-COM Confirmation.....	67
Figure 41 : Trames APT-COM	69
Figure 42 : Préférences de protocole.....	70
Figure 43 : Trames de lecture de température cible	70
Figure 44 : Trame lecture température cible.....	71
Figure 45 : Trame lecture température actuelle.....	71
Figure 46 : Trame lecture de température actuelle.....	72
Figure 47 : Fonction send_modbus_request_read	73
Figure 48 : Fonction send_modbus_request_read	73

Figure 49 : Fonction parse_modbus_float_response	74
Figure 50 : Simulation d'étuve	74
Figure 51 : Exemple d'utilisation classe Etuve	75
Figure 52 : Trame classe Etuve.....	75