Fournier Alan – Boutte Alexis – Konkin Maksim
Dossier commun

Lycée LA FAYETTE Clermont-Ferrand

PROJET NICHOIR

AUTONOME CONNECTÉ

Sommaire

Présentation du système

Un représentant de la LPO Auvergne-Rhône-Alpes est venu nous voir pour exprimer le besoin d'un système permettant le suivi des oiseaux des villes. Notre client est une association comptant plus de 11 440 adhérents et 1500 bénévoles actifs qui effectuent annuellement 300 000 heures de bénévolat au sein de la structure. LPO signifie Ligue de Protection des Oiseaux. Bien que son nom suggère qu'il s'agit d'une association s'occupant uniquement de la protection des oiseaux, la LPO agit pour la biodiversité de manière générale. Par exemple, en Auvergne-Rhône-Alpes, la LPO peut très bien mener des actions de tout type, comme des chantiers nature, des sorties sur le terrain, des conférences, des ateliers, des tenues de stand,… Le tout dans un seul but précis, sensibiliser un public large et varié à la protection de la nature.

Cahier des charges

Le cahier des charges du système nous a été communiqué par le client lors de sa demande. Le périmètre du système s'étend aux villes, pour permettre aux bénévoles de l'association de contrôler plus simplement et efficacement la nidification des oiseaux dans le milieu urbain. La première contrainte concerne la forme du système. Il doit s'agir d'un nichoir qui relève la température et l'humidité intérieure et extérieure, qui prend des photos infrarouges et standards des oiseaux, et enfin qui pèse les oiseaux. Le coût final du produit doit être le plus faible possible, pour être profitable à l'association et être installé en grand nombre. Ensuite, pour permettre un domaine de supervision plus étendu, nous devons permettre une consultation des données sur deux supports, une application sur PC ou smartphone et un site web. Bien entendu, le système doit être sécurisé pour éviter aux données de l'association d'être recueillies par des tiers. Concernant les données, celles-ci doivent être stockées à distance, mais aussi localement en cas de panne de connexion. Enfin, la dernière contrainte est celle du temps. Le produit final doit être prêt en mai, ce qui signifie qu'à partir de la date d'expression du besoin, en début janvier, nous avons 5 mois pour produire un système fonctionnel et complet, répondant au cahier des charges explicité ci-dessus.

Description du système

Pour répondre aux besoins du client, nous avons pris comme base un nichoir en bois qui dispose d'un faux fond. Du point de vue technique, nous allons utiliser un mini-ordinateur comme cœur du système. Nous connecterons à cet appareil plusieurs capteurs. Pour commencer, afin de relever la température et l'humidité à l'intérieur et à l'extérieur du nichoir, nous utiliserons deux capteurs qui permettent la mesure simultanée de ces deux grandeurs. Ensuite, le faux fond du nichoir va nous permettre de mettre en place un système de balance relevant le poids des oiseaux. Pour finir, la capture des images à l'intérieur du nichoir sera faite pour que nous puissions distinguer la présence des oiseaux à l'aide d'une caméra infrarouge et d'une caméra standard. Au final, les données nécessaires au suivi de la nidification des oiseaux seront collectées par 5 capteurs différents. Le mini-ordinateur disposera d'un pare-feu pour interdire les connexions extérieures non souhaitées et ainsi sécuriser le système.

Le système enverra ensuite les données à une base de données hébergée sur un serveur distant, pour leur enregistrement. Le système sera sécurisé par un mot de passe fort, et l'accès sera possible uniquement sur le même réseau local. Les valeurs récupérées par les capteurs seront aussi enregistrées directement sur le mini-ordinateur en cas de panne de connexion. Toutes les acquisitions de données seront datées par la carte à leur envoi dans la base de données dans le but de produire des affichages des valeurs en fonction du temps sur les outils de consultation. Cela servira à suivre la progression des relevés de façon claire.

Objectifs du projet

Objectif principal

L'objectif du projet est de concevoir un service d'observation des oiseaux, afin de contrôler leur nidification. La supervision sera faite avec diffusion des informations en temps réel.

Il s'agira donc de disposer d'un nichoir qui comprend des outils de surveillance. Nous travaillerons sur l'acquisition et la mise en forme des observations pour les rendre facilement accessibles par les utilisateurs.

Objectifs techniques

Le premier objectif technique est d'écrire les programmes principaux sur la carte Raspberry qui permettront de récupérer et d'enregistrer les données issues des capteurs dans la base de données distante et locale avec les bons formats.

En amont, il est nécessaire de mettre en place un serveur de base de données distant qui servira à sauvegarder les informations issues des capteurs : température, humidité, poids, et les images des caméras de surveillance.

Nous devons concevoir l'application sur PC et le site web qui serviront tous les deux à récupérer les informations de la base de données distante et de les afficher de façon claire pour les utilisateurs.

Afin de pouvoir utiliser le site web sur le réseau local, nous devrons mettre en place un serveur web qui hébergera le site, qui servira à visualiser les données collectées par le nichoir au même titre que l'application sur PC.

Pour que les utilisateurs du système n'aient pas la contrainte d'intervenir de manière régulière dans le but de le recharger, l'ensemble du système faisant les acquisitions doit être autonome en énergie.

Enfin, les équipements du projet doivent être sécurisés en tout point. Cela signifie que les deux outils de consultation, la base de données distante ainsi que la partie physique se trouvant au cœur du nichoir sont concernés par le besoin de sécurité.

Choix techniques

Solutions possibles

Le tableau ci-dessous permet d'avoir une vision globale des besoins exprimés par le client et des solutions techniques qui peuvent être réalisées pour les satisfaire.

N° besoin	Besoin	Solutions possibles
1	Contrôler les capteurs	Raspberry Pi 4 Raspberry Pi Zero
2	Récupérer la température et l'humidité relative	DHT22 DHT11
3	Récupérer le poids	HX711 + cellule de charge 780g
4	Capturer des images infrarouges	AMG8833 RPI IR Camera
5	Capturer des images standards	Caméra ToF pour Raspberry Pi 4 Caméra HQ officielle – Monture M12 Raspicam v2.1
6	Gérer l'alimentation du système	Carte Witty Pi 4 Carte Witty Pi 4 Mini
7	Stocker l'énergie nécessaire au fonctionnement du système	Batterie Power Sonic PS1270
8	Contrôler la charge du système	Steca Solsum 6.6c Steca Solsum 8.8c Steca Solsum 10.10c
9	Enregistrer les données à distance	Serveur UwAmp MySQL Serveur MariaDB
10	Enregistrer les données localement	SQLite PostgreSQL
11	Héberger le site web	Serveur web Apache sur Raspberry Serveur web UwAmp distant

Solutions retenues

Le tableau ci-dessous présente les solutions qui ont été retenues pour répondre aux besoins et les raisons qui nous ont poussées à faire ces choix.

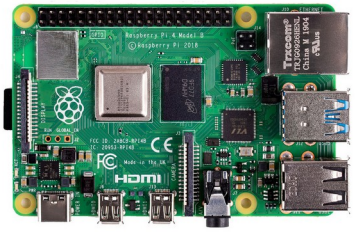
N° besoin	Solution retenue	Raison
1	Raspberry Pi 4	Ce mini-ordinateur dispose d'une plus grande puissance de calcul que la carte Raspberry Pi Zero.
2	DHT22	Le capteur de température du DHT22 peut relever des valeurs négatives, ce qui est utile durant l'hiver.
3	HX711 + cellule de charge 780g	Il s'agit du seul matériel dont nous disposons déjà. La cellule de poids aurait pu permettre de relever un poids plus élevé, mais 780g suffisent.
4	AMG8833	Il s'agit ici aussi du seul matériel dont nous disposons. Bien que sa résolution soit faible, son mode de fonctionnement est similaire aux capteurs avec lesquels nous avons l'habitude de travailler.
5	Raspicam v2.1	Cette caméra standard est la seule dont nous disposons. Cependant, son mode de fonctionnement est simple, notre choix se serait tout de même porté sur cette solution.
6	Carte Witty Pi 4	La carte Witty Pi 4 Mini est trop petite pour être maintenue entre le Raspberry Pi 4 et le shield, nous avons donc choisi la carte Witty Pi 4, dont le montage convient mieux.
7	Batterie Power Sonic PS1270	Il s'agit de la seule batterie dont nous disposons. Nous aurions pu choisir une batterie 5V, mais l'utilisation d'un contrôleur de charge nous permet d'utiliser une batterie 12V.
8	Steca Solsum 6.6c	Ce contrôleur de charge est le seul que nous avons à disposition. Nous n'avons pas cherché un autre modèle, car celui-ci remplit bien son rôle.
9	Serveur UwAmp MySQL	Nous avons l'habitude d'utiliser le serveur UwAmp. Son interface phpMyAdmin est pratique et claire, contrairement au serveur MariaDB qui fonctionne en lignes de commandes.
10	SQLite	La solution PostgreSQL aurait pu être envisagée au même titre que la base de données SQLite. Ces deux possibilités remplissent leur rôle de façon similaire.
11	Serveur web UwAmp distant	Le serveur UwAmp distant permet aussi d'héberger facilement un site web. Héberger le site sur la carte Raspberry n'est pas possible puisqu'elle s'éteint lorsqu'elle n'est pas utilisée.

Matériel utilisé

Nichoir en bois

Notre système physique prendra la forme d'un nichoir en bois dont la forme est similaire à l'image qui vous est présentée. Ce nichoir contiendra l'ensemble des capteurs ainsi que le mini-ordinateur qui s'occupera de l'acquisition et de l'enregistrement des données. À proximité du nichoir seront aussi placés les équipements nécessaires à l'autonomie du système tels que la batterie, la cellule photovoltaïque et le contrôleur de charge.

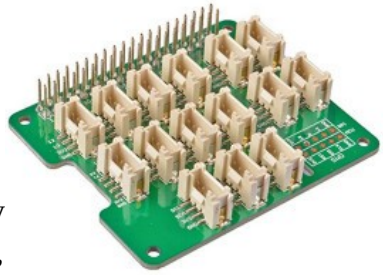
Mini-ordinateur Raspberry Pi 4



Le cœur du système physique est la carte mini-ordinateur Raspberry Pi 4. La Raspberry Pi 4 se différencie des microcontrôleurs tels que la carte Arduino Uno. En effet, elle est composée d'un microcontrôleur : processeur, mémoire vive et morte, interfaces d'entrées-sorties, ainsi que d'un système d'exploitation, lui valant l'appellation de « mini-ordinateur ».

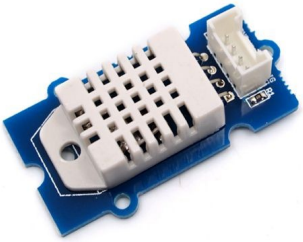
Au sein du projet, la Raspberry Pi 4 va nous servir à récupérer les données des différents capteurs pour les traiter puis les enregistrer dans la base de données distante. Par ailleurs, elle contiendra une base de données locale pour pouvoir tout de même enregistrer les données dans le cas où la connexion à la base de données distante serait interrompue.

Module Grove Base Hat



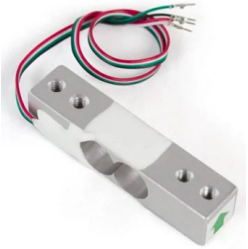
Les connexions entre tous les capteurs et la carte Raspberry Pi 4 sont simplifiées à l'aide du module Grove Base Hat, communément appelé « shield ». Ce module se branche directement sur le mini-ordinateur et met à disposition des ports physiques utilisant différents protocoles de communication en fonction des besoins de l'utilisateur.

Capteur de température et d’humidité DHT22



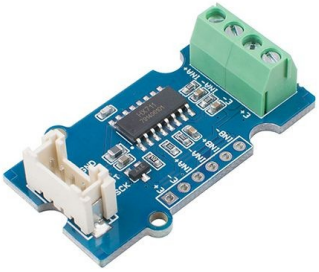
Le capteur DHT22 permet de faire l’acquisition de la température et de l’humidité relative de façon simultanée. Sa sensibilité est intéressante vis à vis des besoins du projet puisqu’elle est faible. Les mesures sont effectuées à 0,1°C près pour la température et 0,1 % près pour l’humidité relative. De même, sa plage de relevés est intéressante. Avec ce capteur, il est possible de mesurer la température de -40°C à 80°C, ce qui est idéal pour les relevés en hiver. L’humidité relative est quant à elle relevée de 0 à 100 %. Ce capteur communique avec la carte Raspberry Pi 4 à l’aide d’un port GPIO (General Purpose Input/Output) qui est un port série d’entrées-sorties.

Cellule de charge micro CZL611CD



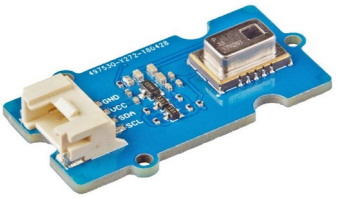
La cellule de charge utilisée peut mesurer une masse maximale de 780g, ce qui est suffisant en considérant que les oiseaux ne pèsent que quelques grammes. Elle est composée de 4 jauges de contraintes, qui sont des fils dont la résistance varie en fonction de la déformation qui leur est appliquée. Montées électriquement selon le schéma du pont de Wheatstone, il est ainsi possible de récupérer une tension de sortie en fonction de la charge appliquée.

Capteur de poids HX711



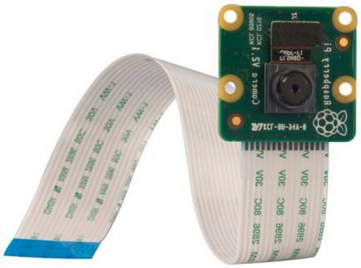
Le capteur HX711 récupère la tension en sortie de la cellule de charge, qui sert à la mesure du poids. La tension de sortie est ensuite traitée par le capteur pour être utilisée par la carte Raspberry Pi 4. Le HX711 communique avec le mini-ordinateur à l’aide d’un port série d’entrées-sorties GPIO.

Caméra infrarouge AMG8833



Cette caméra infrarouge permet de capturer des images à l'aide d'un capteur de résolution 8x8 pixels. Cette résolution n'étant pas suffisante pour interpréter les images capturées, il sera nécessaire de mettre en place une interpolation, qui consiste à générer des valeurs intermédiaires entre les valeurs réellement mesurées pour obtenir une meilleure résolution. La caméra infrarouge communique avec la carte Raspberry Pi 4 par l'intermédiaire du bus série I2C.

Caméra standard Raspicam v2.1



Conçue par les mêmes constructeurs que la carte Raspberry Pi 4, cette caméra standard permet de capturer des images en utilisant des lignes de commandes, contrairement au reste des capteurs qui doivent être programmés. Il est possible de régler plusieurs paramètres lors de la prise de l'image tels que la durée d'exposition et le grain par exemple. Cette caméra est reliée au port MIPI CSI situé sur la partie droite du mini-ordinateur. La caméra utilise une nappe et communique à l'aide de l'interface CSI.

Carte Witty Pi 4

La carte Witty Pi 4 gère l'alimentation de la carte Raspberry Pi 4. Elle permet notamment de définir les arrêts et démarrages programmés du système afin de consommer le moins d'énergie possible. Cette fonction est réalisée à l'aide de l'horloge RTC (Real Time Clock) intégrée à la carte qui conserve la date et l'heure en mémoire lorsque le système est éteint. La carte Witty Pi 4 se connecte directement sur la Raspberry Pi 4 de la même manière que le shield. Une fois connectée, l'alimentation doit être branchée sur la carte et non sur le mini-ordinateur.

Batterie Power Sonic PS1270

L’autonomie du système sera assurée par plusieurs éléments, dont la batterie Power Sonic PS1270. Elle délivre une tension de 12V et dispose d’une capacité de 7Ah, ce qui est amplement suffisant compte tenu de l’énergie consommée par le système en moyenne.

Cellule photovoltaïque Velleman SOL10P

Cette cellule photovoltaïque permettra de délivrer l’énergie nécessaire pour recharger la batterie en évitant les pertes. Son rendement est de 11 %, ce qui reste faible. Cependant, en hiver, dans le cas le plus défavorable, cela suffit à recharger suffisamment la batterie pour que le système ne s’éteigne pas.

Contrôleur de charge Steca Solsum 6.6c

Afin de relier tous les éléments permettant de mettre en autonomie le système, nous aurons besoin du contrôleur de charge Steca Solsum 6.6c. Dessus, nous connecterons la batterie, la cellule photovoltaïque et la carte Witty Pi 4. Notamment, le contrôleur de charge permettra de délivrer les 5V nécessaires en sortie pour le système, avec une batterie de tension 12V.

Outils utilisés

Qt Creator

Afin de développer l'application de consultation sur PC, l'étudiant 2 va se servir de l'environnement de développement Qt Creator. Cet outil est multi-plateforme et dispose d'une multitude de bibliothèques pour mener à bien son projet. Le langage de programmation qui est utilisé est le C++.

Visual Studio Code

Développé par Microsoft, Visual Studio Code est un éditeur de code qui prend en charge un grand nombre de langages de programmation. Notamment, cet outil peut suggérer de compléter notre code, met en évidence la syntaxe, et prend en charge le débogage. Dans le cadre de notre projet, Visual Studio Code a été utilisé avec l'extension Live Server pour développer le site web de consultation en observant les modifications en temps réel.

Serveur UwAmp

Le serveur UwAmp est très simple à mettre en place et à utiliser. Il est disponible sous la forme d'une archive, qu'il suffit de décompresser. Les lettres « Amp » signifient respectivement « Apache », « MySQL » et « PHP ». Ce serveur est composé d'un serveur web Apache pour héberger les sites web, d'un serveur de bases de données MySQL et prend en charge le langage PHP. Cet outil permet donc à la fois d'héberger le site web de consultation ainsi que d'héberger la base de données distante, en proposant une interface claire et simple à utiliser.

GitHub

GitHub est une société qui offre un service d'hébergement de référentiel Git, qui est un système de contrôle de version open source spécifique créé en 2005. Le contrôle de version permet aux développeurs de suivre et gérer les modifications apportées au code d'un projet. Nous avons créé et utilisé un dépôt GitHub pour sauvegarder le code du projet et s'organiser en notant les tâches effectuées dans des fichiers texte.

Coût du projet

Le coût du projet correspond aux dépenses nécessaires à son fonctionnement. Les dépenses se limitent uniquement au matériel, car l'ensemble des outils et des programmes utilisés sont gratuits. Vous trouverez ci-dessous le prix de chaque élément composant du projet.

Composant	Prix
Nichoir en bois	30€
Raspberry Pi 4	60€
Grove Base Hat	10,6€
DHT22 x2	22,8€
HX711	5,9€
Cellule de charge 780g	7,9€
AMG8833	44,5€
Raspicam v2.1	29,9€
Witty Pi 4	37,95€
Power Sonic PS1270	14,9€
Velleman SOL10P	39,96€
Steca Solsum 6.6c	50,66€
Total	377,87€

Le coût total du matériel composant le projet est égal à 377,87€. Si nous considérons que l'ordinateur qui fait office de serveur est aussi à acheter, nous pouvons alors ajouter au total environ 200€, ce qui ferait un coût de 577,87€ pour l'ensemble. Les prix des composants ont été récupérés sur différents sites de fournisseurs. Il n'a pas été possible de récupérer tous les prix sur le même site.

Répartition des tâches

Pour suivre et organiser l'avancement du projet, nous avons mis en place un dépôt GitHub. Chaque élève dispose d'un fichier de suivi où il explique les tâches qu'il a effectuées, concluantes ou non, durant une session de projet. Ces informations nous permettront de fabriquer le diagramme de Gantt réel, pour le comparer au diagramme de Gant prévisionnel, présenté un peu plus tard dans le dossier commun. Le dépôt GitHub permet aussi de sauvegarder les fichiers et programmes du projet.

Étudiant 1 – Alan Fournier

Alan doit réaliser les programmes principaux sur la carte Raspberry qui permettent l'acquisition et la sauvegarde des données issues des capteurs. Il doit aussi implémenter le processus de prise d'images. Toutes les données récupérées doivent être enregistrées dans une base de données locale et distante. Il s'occupe aussi de mettre en place le service sur le système embarqué qui permet la mise en veille et le réveil programmés. Les tâches qu'il doit effectuer sont à la racine du projet puisque sans les données réelles des capteurs, les autres étudiants ne peuvent pas entièrement compléter leurs tâches.

Étudiant 2 – Alexis Boutte

Alexis s'occupe de la partie Interface Homme Machine (IHM) externe sous forme d'une application pour le contrôle distant du nichoir. Il doit mettre en place un serveur de base de données ainsi que créer la base de données distante qui sert au stockage des informations. Les données doivent pouvoir être affichées et consultées par les utilisateurs. L'application qu'il développe doit pouvoir interpréter les données dans le bon format pour les afficher de façon claire.

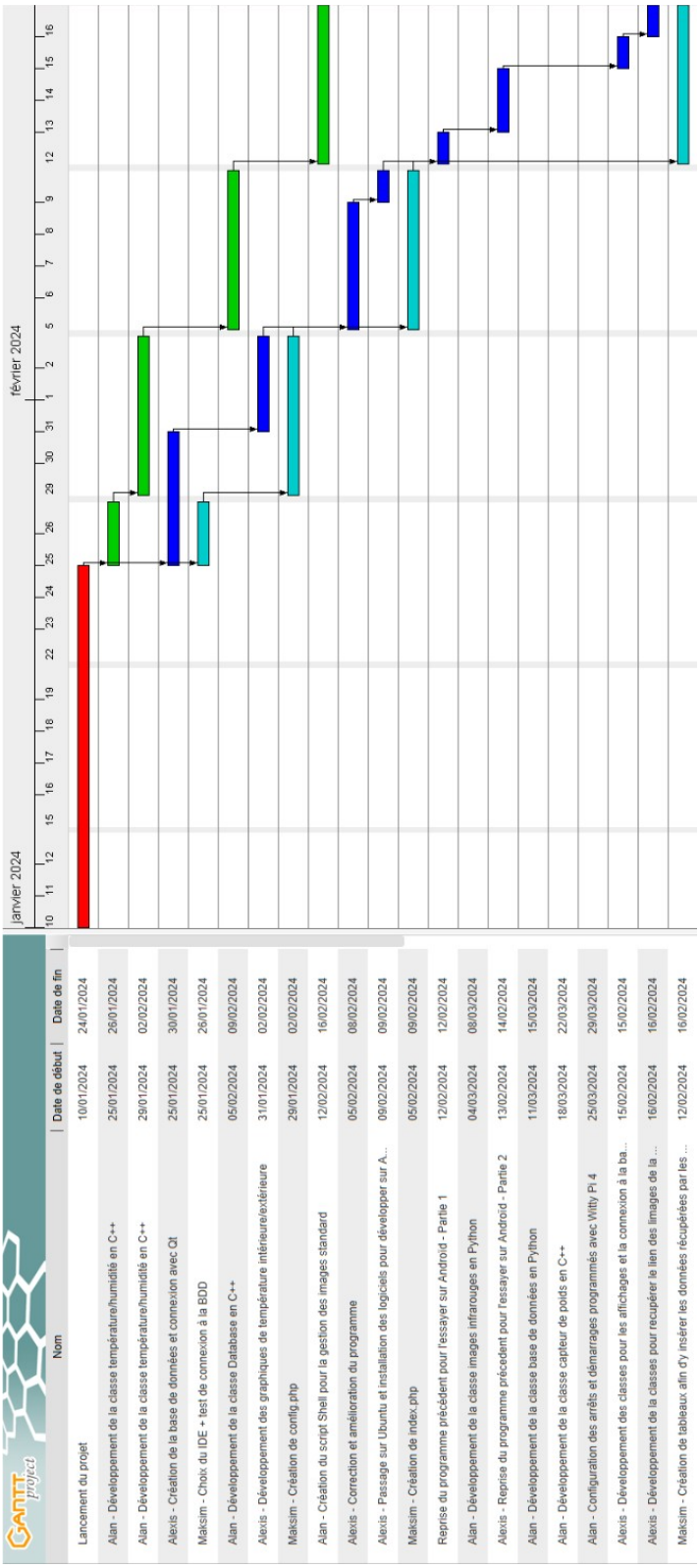
Étudiant 3 – Maksim Konkin

Maksim doit mettre en place un serveur web et développe un site Internet hébergé sur ce serveur permettant de visualiser les données collectées par le nichoir. Tout comme Alexis, l'outil de consultation de Maksim doit pouvoir interpréter les données de manière à ce qu'elles paraissent claires aux utilisateurs et contrôler le nichoir.

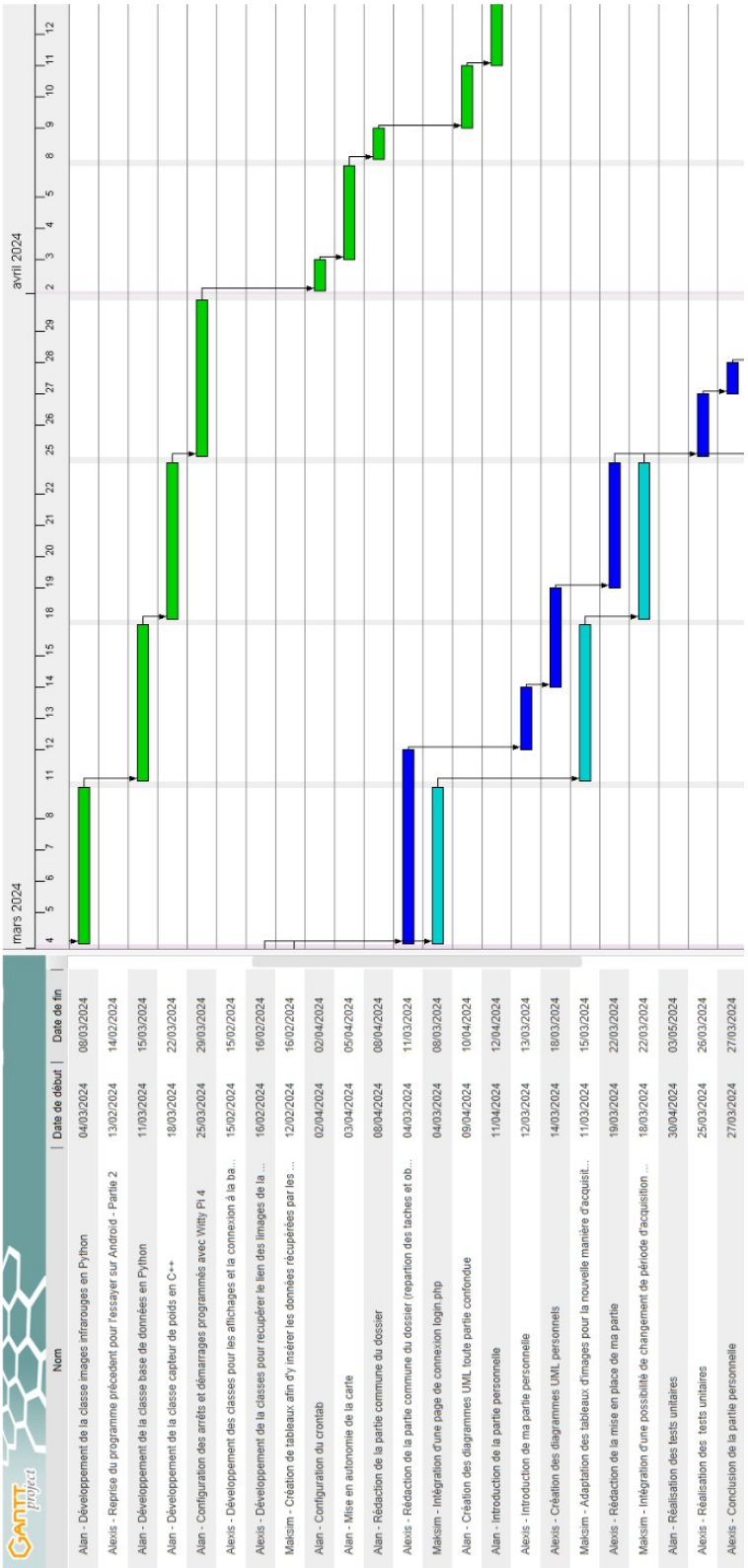
Diagramme de Gantt prévisionnel

Janvier à Février

Les tâches en bleu foncé sont celles d'Alexis, en cyan celles de Maksim et celles en vert sont celles d'Alan.



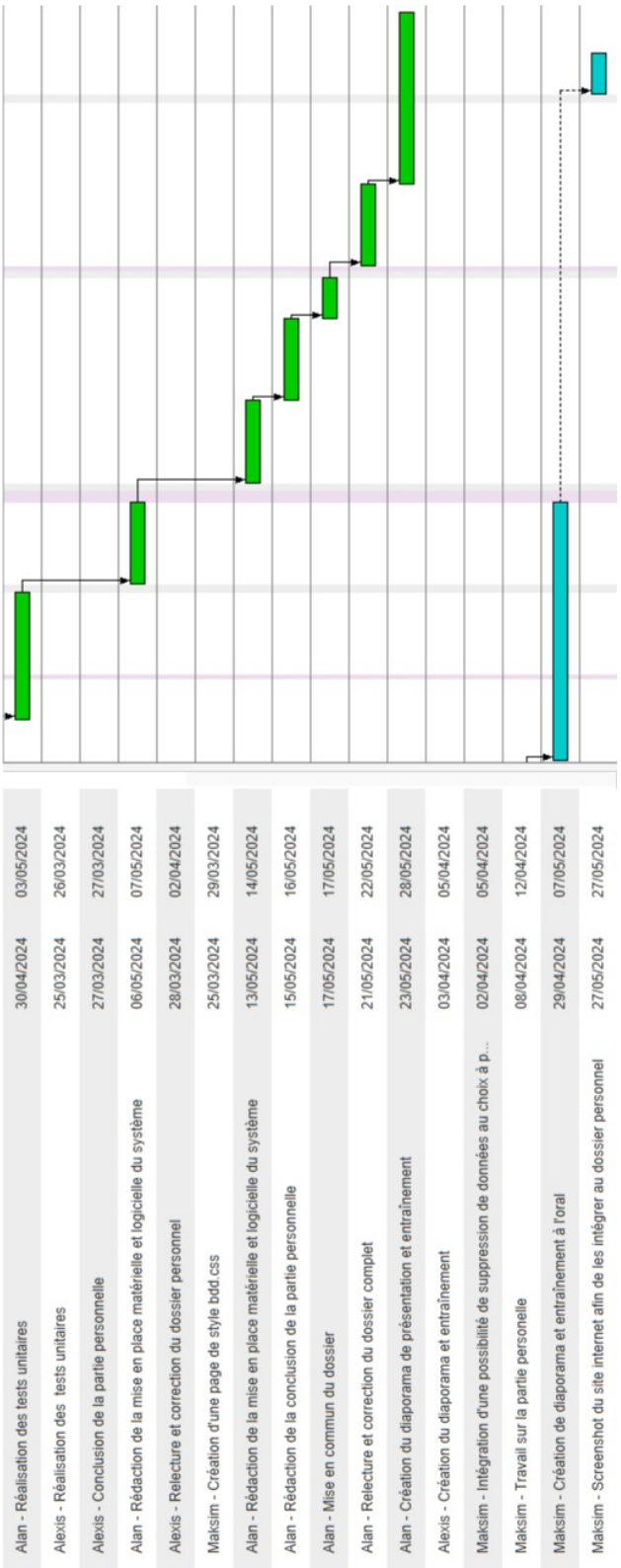
Mars à Avril 1e partie



Mars à Avril 2e partie

Alexis - Relecture et correction du dossier personnel	28/03/2024	02/04/2024
Maksim - Création d'une page de style bdd.css	25/03/2024	29/03/2024
Alan - Rédaction de la mise en place matérielle et logicielle du système	13/05/2024	14/05/2024
Alan - Rédaction de la conclusion de la partie personnelle	15/05/2024	16/05/2024
Alan - Mise en commun du dossier	17/05/2024	17/05/2024
Alan - Relecture et correction du dossier complet	21/05/2024	22/05/2024
Alan - Création du diaporama de présentation et entraînement	23/05/2024	28/05/2024
Alexis - Création du diaporama et entraînement	03/04/2024	05/04/2024
Maksim - Intégration d'une possibilité de suppression de données au choix à p...	02/04/2024	05/04/2024
Maksim - Travail sur la partie personelle	08/04/2024	12/04/2024
Maksim - Création de diaporama et entraînement à l'oral	29/04/2024	07/05/2024

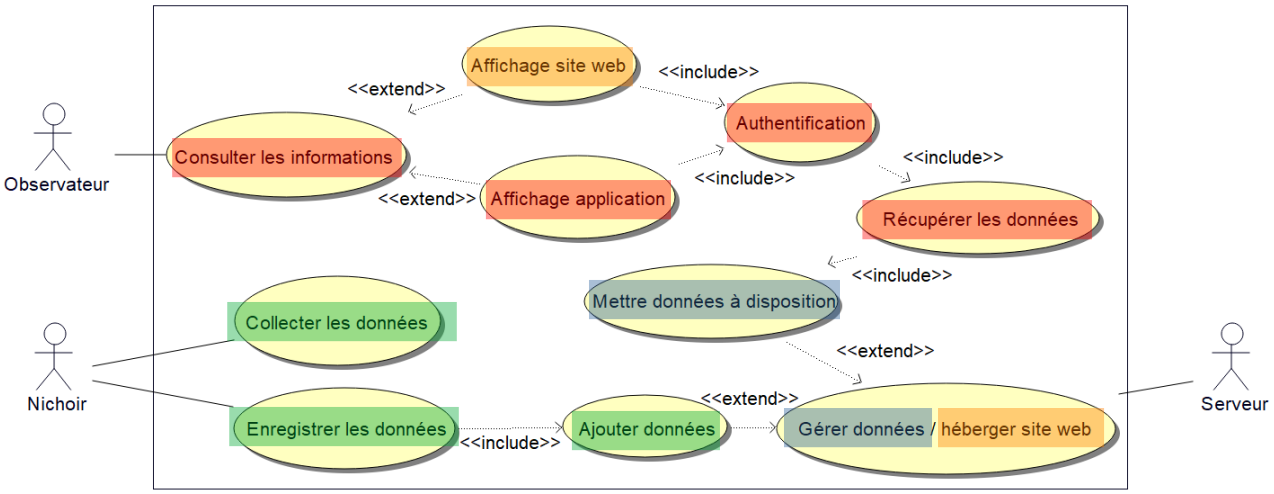
Mai



Alan - Réalisation des tests unitaires	30/04/2024	03/05/2024
Alexis - Réalisation des tests unitaires	25/03/2024	26/03/2024
Alexis - Conclusion de la partie personnelle	27/03/2024	27/03/2024
Alan - Rédaction de la mise en place matérielle et logicielle du système	06/05/2024	07/05/2024
Alexis - Relecture et correction du dossier personnel	28/03/2024	02/04/2024
Maksim - Création d'une page de style bdd.css	25/03/2024	29/03/2024
Alan - Rédaction de la mise en place matérielle et logicielle du système	13/05/2024	14/05/2024
Alan - Rédaction de la conclusion de la partie personnelle	15/05/2024	16/05/2024
Alan - Mise en commun du dossier	17/05/2024	17/05/2024
Alan - Relecture et correction du dossier complet	21/05/2024	22/05/2024
Alan - Création du diaporama de présentation et entraînement	23/05/2024	28/05/2024
Alexis - Création du diaporama et entraînement	03/04/2024	05/04/2024
Maksim - Intégration d'une possibilité de suppression de données au choix à p...	02/04/2024	05/04/2024
Maksim - Travail sur la partie personelle	08/04/2024	12/04/2024
Maksim - Création de diaporama et entraînement à l'oral	29/04/2024	07/05/2024
Maksim - Screenshot du site internet afin de les intégrer au dossier personnel	27/05/2024	27/05/2024

UML

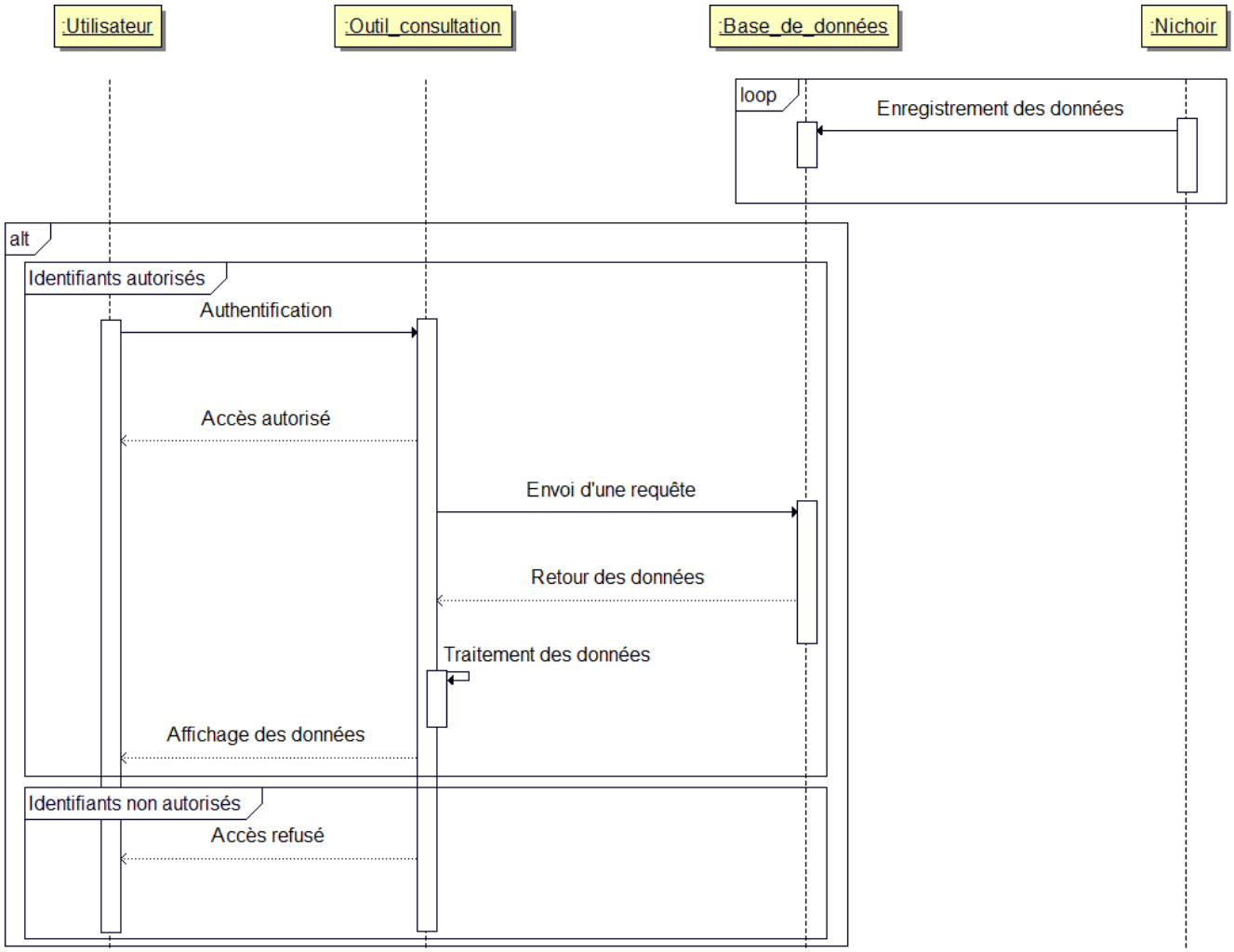
Diagramme des cas d’utilisation



Les zones vertes représentent les tâches effectuées par Alan, l’étudiant 1. Les zones bleues représentent les tâches du projet effectuées par Alexis, l’étudiant 2. Les tâches effectuées par Maksim, l’étudiant 3, sont représentées par les zones oranges. Enfin, les zones rouges représentent les tâches effectuées en commun par Alexis et Maksim, les étudiants 2 et 3.

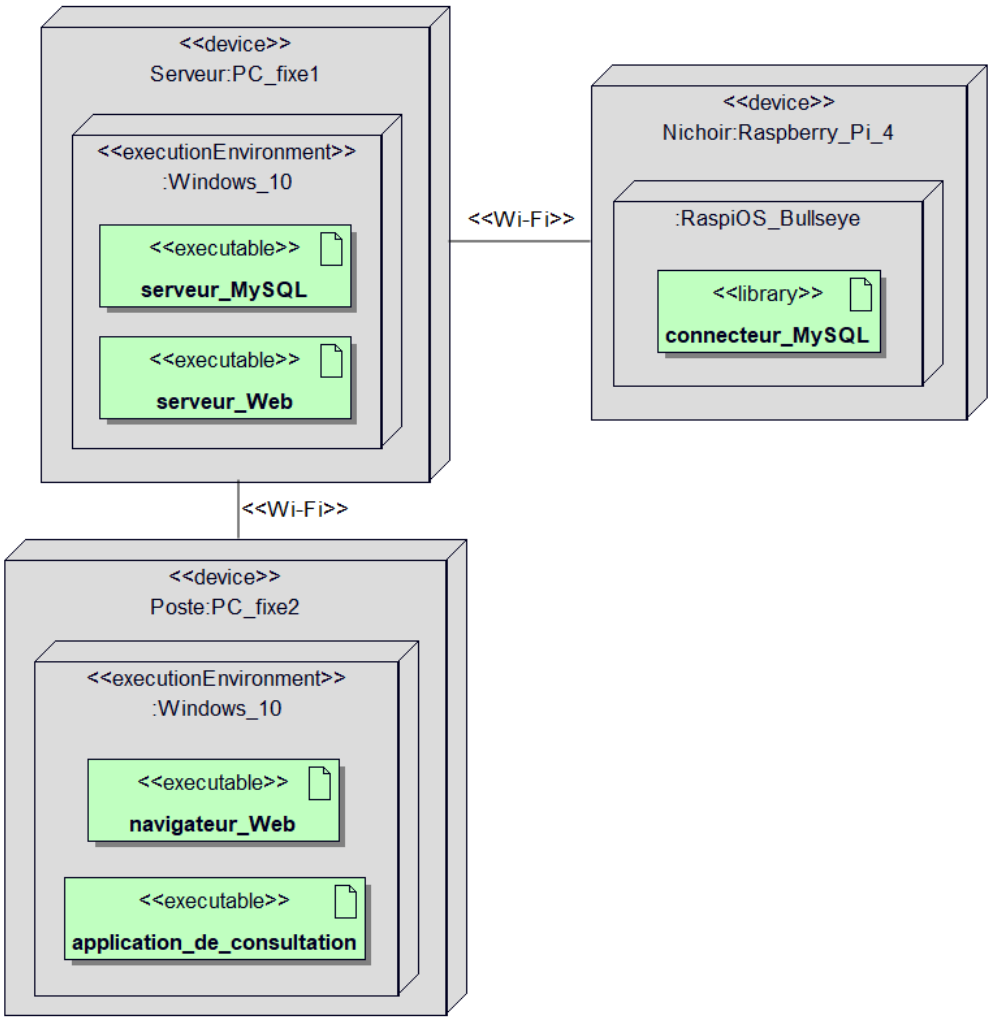
Lorsque l’observateur souhaite consulter les données, elles doivent au préalable avoir été collectées par le nichoir, puis sauvegardées sur le serveur de base de données. Les données sont ensuite affichées sur l’application ou sur le site web en fonction de la méthode souhaitée par l’observateur.

Diagramme de séquence



Si l'utilisateur dispose d'identifiants autorisés, il peut accéder aux données, l'outil de consultation envoie alors une requête à la base de données. Une fois récupérées, les données sont traitées puis affichées par l'outil de consultation. À l'inverse, si ses identifiants de l'utilisateur ne sont pas autorisés, l'utilisateur ne peut pas accéder aux données. L'outil de consultation lui indique que l'accès lui est refusé.

Diagramme de déploiement



La carte Raspberry Pi 4 communique en Wi-Fi avec la base de données distante pour enregistrer les données des capteurs qu'elle récupère. Sur un PC fixe, nous pourrons exécuter une application de consultation qui se connectera à la base de données en Wi-Fi pour accéder aux données et les afficher, ou alors utiliser un navigateur web pour consulter les données sur un site Internet.

Alan Fournier
Dossier personnel

Lycée LA FAYETTE Clermont-Ferrand

PROJET NICHOIR

AUTONOME CONNECTE

Remerciements

Tout d’abord, j’aimerais remercier sincèrement mes enseignants de spécialité, M. Bulcke, M. Barthomeuf, M. Degoute et Mme. Chopin pour leur accompagnement tout au long du projet. Je leur suis reconnaissant pour le soutien, le suivi et les réponses qu’ils m’ont tous les quatre apportés afin de mener à bien mon projet de fin d’année.

Ensuite, je remercie mes coéquipiers de projet, avec qui j’ai eu l’opportunité de travailler en groupe durant les 5 derniers mois. C’est grâce à notre collaboration que l’ensemble du projet a pu aboutir, de l’organisation à la mise en place des solutions envisagées.

Sommaire

Table des figures

Présentation de ma partie

Dans un premier temps, je dois m'occuper de l'acquisition des données des capteurs. Pour ce faire, je vais développer des programmes dans des langages de programmation différents. J'avais initialement prévu d'écrire tous mes programmes dans le même langage, mais pour certaines raisons explicitées plus tard dans mon dossier, cela n'a pas été possible. Ainsi, un premier programme permettra l'acquisition des données du capteur de poids et des capteurs de température et d'humidité, et un autre servira à l'acquisition des images des deux caméras.

Dans un second temps, je dois pouvoir enregistrer les données que les capteurs récupèrent dans une base de données distante, que nous avons mise en place et configurée avec mon coéquipier Alexis. Puisque j'utilise plusieurs langages de programmation pour les capteurs, je dois créer plusieurs programmes dans chaque langage afin de permettre la connexion à la base de données et l'enregistrement des données à l'intérieur. Par ailleurs, une base de données locale sera installée sur le système pour avoir une copie des données et ainsi éviter de les perdre si la connexion à la base de données distante est interrompue.

Ensuite, pour la sécurité du système, des identifiants sécurisés seront utilisés pour toutes les connexions entre les différents appareils, telles qu'entre la base de données et la carte Raspberry Pi 4 ou encore entre le PC exécutant l'application de consultation des données et la base de données. Pour filtrer les connexions extérieures à la carte Raspberry Pi 4, je vais mettre en place un pare-feu pour n'autoriser que les connexions utiles et bloquer toutes les autres.

Enfin, pour que le système soit autonome, je vais mesurer l'énergie qu'il consomme. À partir de cette donnée, je vais pouvoir calculer les caractéristiques de la batterie et de la cellule photovoltaïque pour alimenter le système sans interruption pendant les heures prévues.

Responsabilités dans le projet et dans le groupe

Au sein du projet Nichoir Autonome Connecté, j’ai pour responsabilité de faire fonctionner la partie matérielle. En programmant les capteurs connectés à la carte Raspberry Pi 4, je dois récupérer les données puis les sauvegarder localement et dans la base de données distante.

Mes tâches sont à la racine du projet. En effet, pour que mes coéquipiers puissent récupérer les données pour les afficher à l’aide de leurs outils de consultation, je dois au préalable avoir programmé la carte pour que les capteurs puissent faire des acquisitions, et que chaque valeur soit stockée dans la base de données distante pour l’affichage.

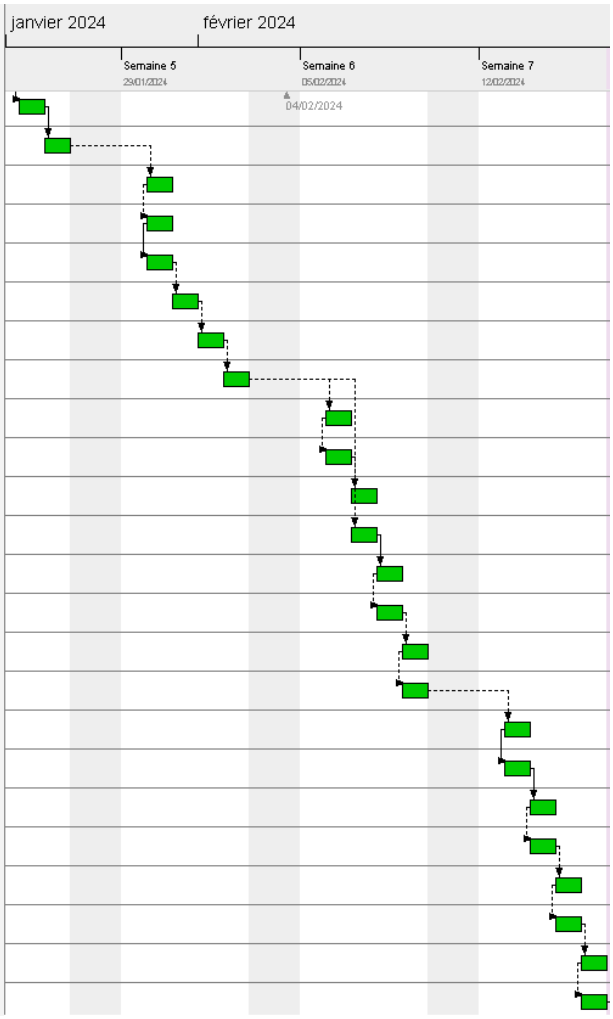
Mon but est aussi de veiller à la sécurité de la carte Raspberry Pi 4, afin d’éviter toute intrusion sur le système. De ce fait, je dois aussi travailler à la mise en place de règles de sécurité, comme le contrôle des flux entrants et sortants de la carte et l’utilisation d’identifiants d’accès sécurisés sur la carte et sur la base de données.

Enfin, pour que le système puisse être autonome, je dois mettre en place une solution de recharge d’une batterie à l’aide d’une cellule photovoltaïque, en tenant compte des heures d’activité de la carte et de sa consommation journalière.

Diagramme de Gantt réel

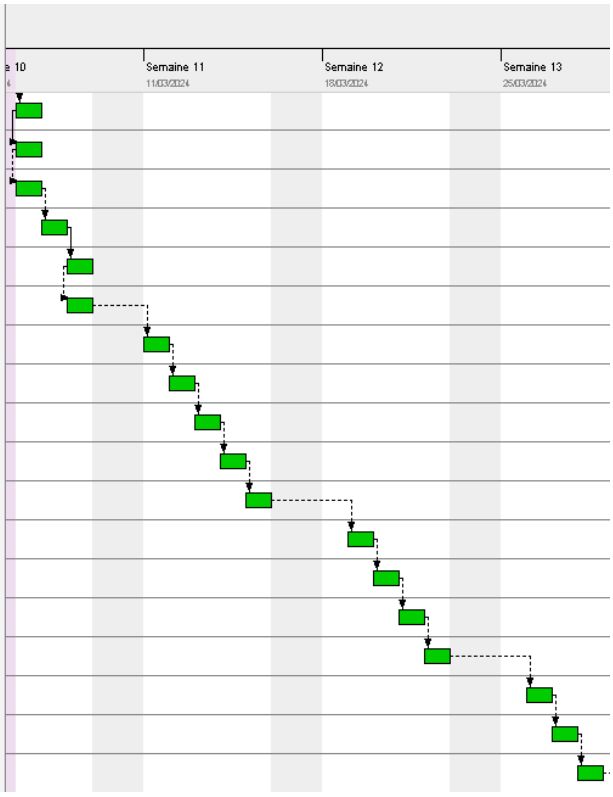
Janvier à Février

Nom	Date de début	Date de fin
Essai capteur DHT11 C++ - non concluant	25/01/2024	25/01/2024
Essai en Python - non concluant	26/01/2024	26/01/2024
Essai DHT22 C++ - fonctionnel	30/01/2024	30/01/2024
Autorisation connexion B.D.D réseau local	30/01/2024	30/01/2024
Connexion carte / base de données	30/01/2024	30/01/2024
Première requête de sauvegarde fonctionnelle	31/01/2024	31/01/2024
Essai HX711 C++ - non concluant	01/02/2024	01/02/2024
Recherches des bibliothèques fonctionnelles	02/02/2024	02/02/2024
Création script Shell caméra standard - fonctionnel	06/02/2024	06/02/2024
Configuration crontab - fonctionnel	06/02/2024	06/02/2024
Essai HX711 Arduino : comprendre fonctionnement	07/02/2024	07/02/2024
Essai AMG8833 concluant	07/02/2024	07/02/2024
Création classes Python	08/02/2024	08/02/2024
Recherche problème HX711	08/02/2024	08/02/2024
Création diagramme prévisionnel	09/02/2024	09/02/2024
Ajustements programmes existants	09/02/2024	09/02/2024
Remplacement cellule de charge	13/02/2024	13/02/2024
HX711 C++ : fonctionnel	13/02/2024	13/02/2024
Ajustement des programmes	14/02/2024	14/02/2024
Branchement / installation Witty Pi 4	14/02/2024	14/02/2024
Début rédaction partie personnelle	15/02/2024	15/02/2024
Ajustements programmes	15/02/2024	15/02/2024
Souci format date et heure B.D.D	16/02/2024	16/02/2024
Création classe Nichoir C++	16/02/2024	16/02/2024



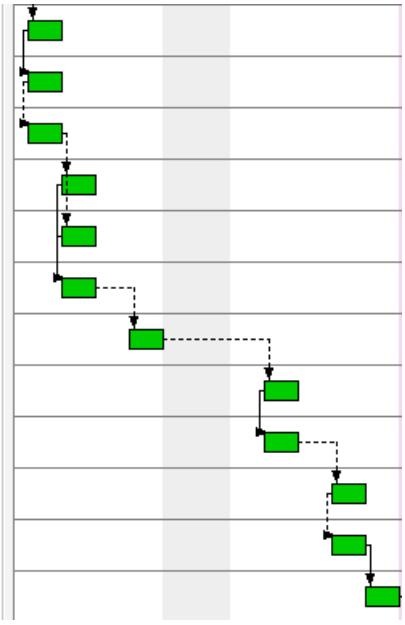
Mars

Nom	Date de début	Date de fin
Tous programmes fonctionnels	06/03/2024	06/03/2024
Nouveau crontab	06/03/2024	06/03/2024
Recherches démarrages / arrêts programmés	06/03/2024	06/03/2024
Ajustements programmes	07/03/2024	07/03/2024
Amélioration programmes	08/03/2024	08/03/2024
Création diagrammes UML personnels	08/03/2024	08/03/2024
Rédaction partie personnelle	11/03/2024	11/03/2024
Calcul consommation pour autonomie	12/03/2024	12/03/2024
Rédaction partie personnelle	13/03/2024	13/03/2024
Paramétrage arrêts et démarrages programmés	14/03/2024	14/03/2024
1e revue de projet : prise de notes remarques	15/03/2024	15/03/2024
Création diagrammes UML communs	19/03/2024	19/03/2024
Rédaction partie personnelle	20/03/2024	20/03/2024
Passage en connexion Wi-Fi : ajustements progr...	21/03/2024	21/03/2024
Montage faux fond	22/03/2024	22/03/2024
Rédaction partie personnelle	26/03/2024	26/03/2024
Création diaporama revue de projet	27/03/2024	27/03/2024
Ajustements programmes et partie personnelle	28/03/2024	28/03/2024



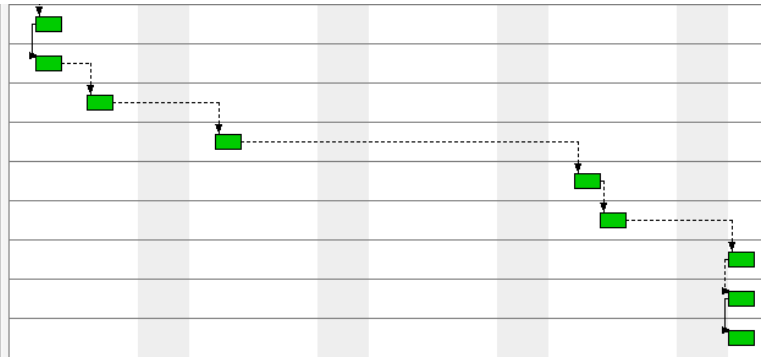
Avril

Modification B.D.D : champ IP	02/04/2024	02/04/2024
Ajout récupération IP Python et Shell	02/04/2024	02/04/2024
Arrêt et démarrage toutes les 30 min	02/04/2024	02/04/2024
Modification script Shell	03/04/2024	03/04/2024
Modification B.D.D	03/04/2024	03/04/2024
Import image dans B.D.D	03/04/2024	03/04/2024
Modification programmes Python	05/04/2024	05/04/2024
Suppression script Shell	09/04/2024	09/04/2024
Utilisation caméra standard en Python	09/04/2024	09/04/2024
Modification B.D.D	11/04/2024	11/04/2024
Ajustements programme Python	11/04/2024	11/04/2024
Écriture partie commune	12/04/2024	12/04/2024



Mai

Ajout attribut u_sched pour gérer l'autonomie	30/04/2024	30/04/2024
Gestion de l'autonomie en Python	30/04/2024	30/04/2024
Montage des capteurs dans le nichoir	02/05/2024	02/05/2024
Implémentation sauvegarde locale des données	07/05/2024	07/05/2024
Relecture de tous les programmes et correctifs	21/05/2024	21/05/2024
Rédaction dossier du projet	22/05/2024	22/05/2024
Relecture et ajustements finaux des programmes	27/05/2024	27/05/2024
Rendu du dossier du projet	27/05/2024	27/05/2024
Projet fonctionnel et terminé	27/05/2024	27/05/2024

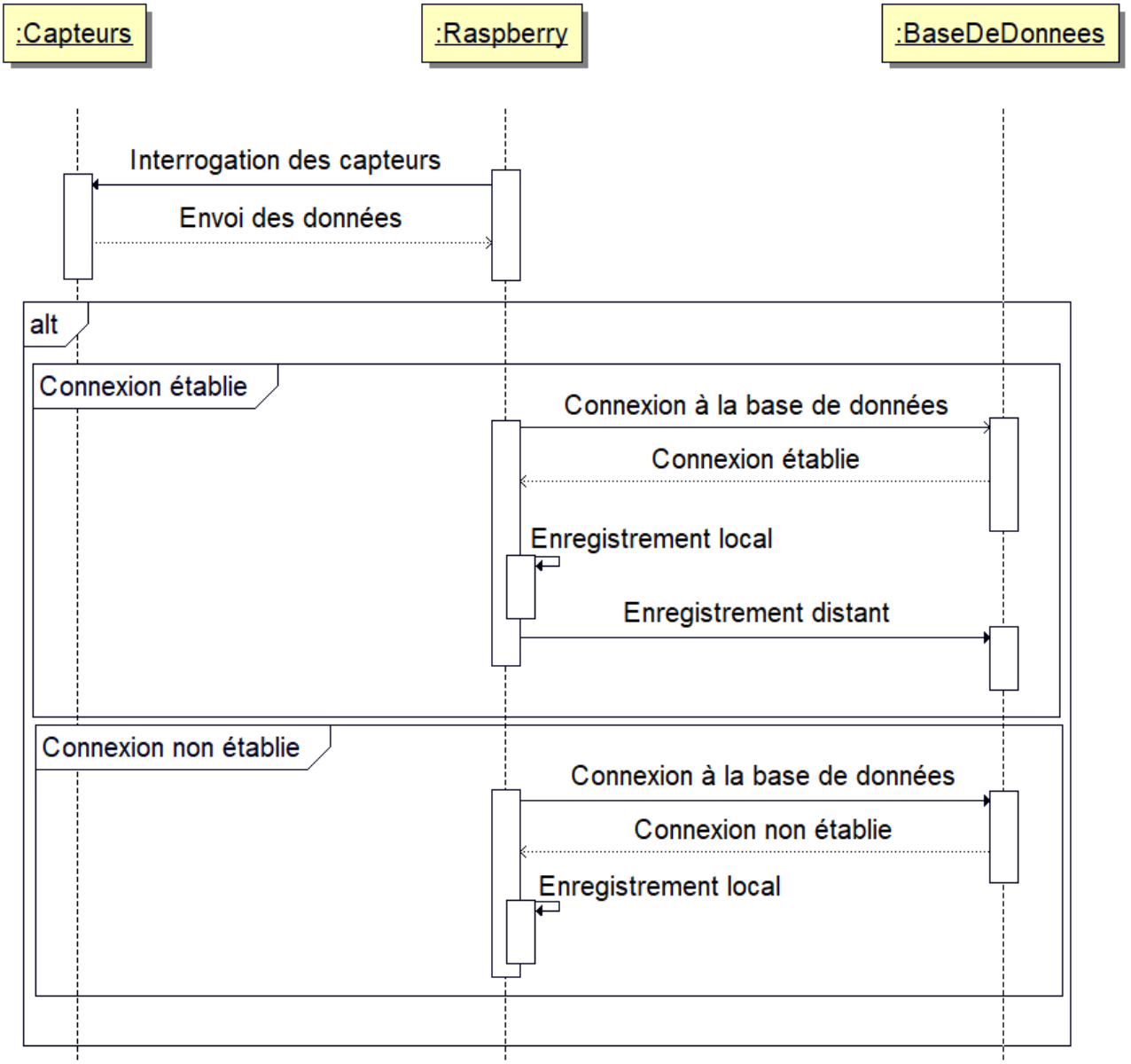


En comparant mon diagramme de Gantt prévisionnel et mon diagramme de Gantt réel, nous constatons que chaque tâche m'a demandé moins de temps que prévu. De manière générale, je me suis organisé afin de gagner du temps pour des possibles modifications et relectures des programmes. À chaque problème rencontré, je suis passé à la tâche suivante et une fois celle-ci terminée, je cherchais à résoudre le problème après avoir fait des recherches supplémentaires en amont ou demandé de l'aide à mes professeurs.

Grâce au temps gagné vis à vis des tâches prévues, j'ai obtenu plus de temps pour revoir certains points. Plusieurs ajustements ont été faits durant le développement du projet en accord avec mes coéquipiers. Toutes les informations seront données dans la partie mise en place de mon dossier personnel.

UML

Diagramme de séquence

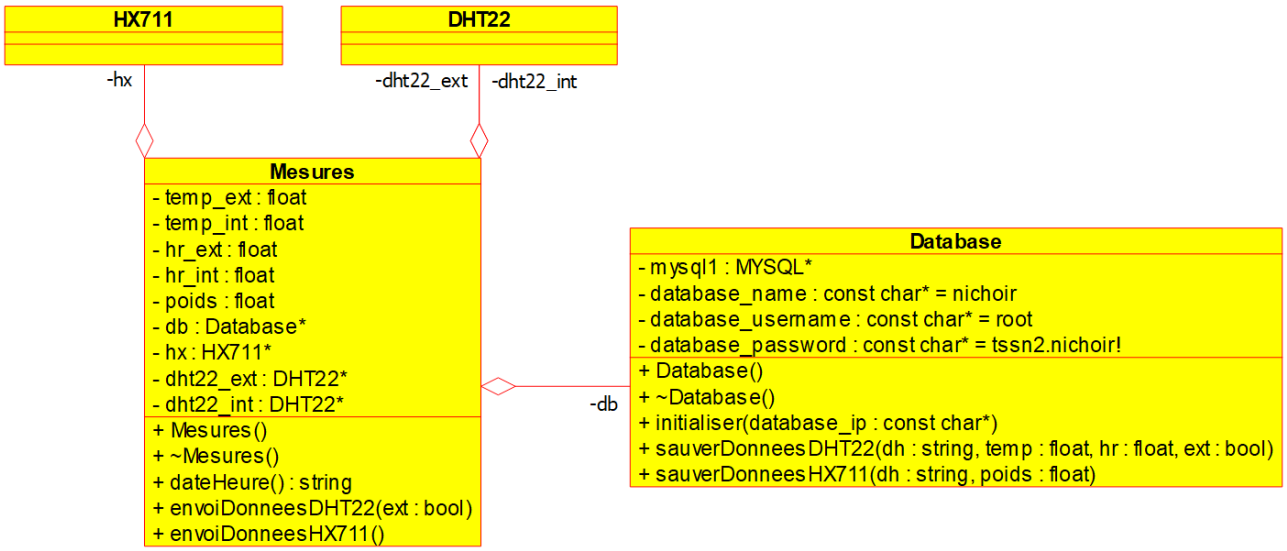


L'enregistrement des données est effectué dans une base de données distante pour que les outils de consultation puissent les récupérer. De plus, les données sont aussi enregistrées dans une base de données locale. Dans le cas où la connexion à la base de données distante serait établie, les données sont enregistrées localement et à distance. Sinon, les données sont uniquement enregistrées localement afin de ne pas les perdre.

Diagrammes des classes

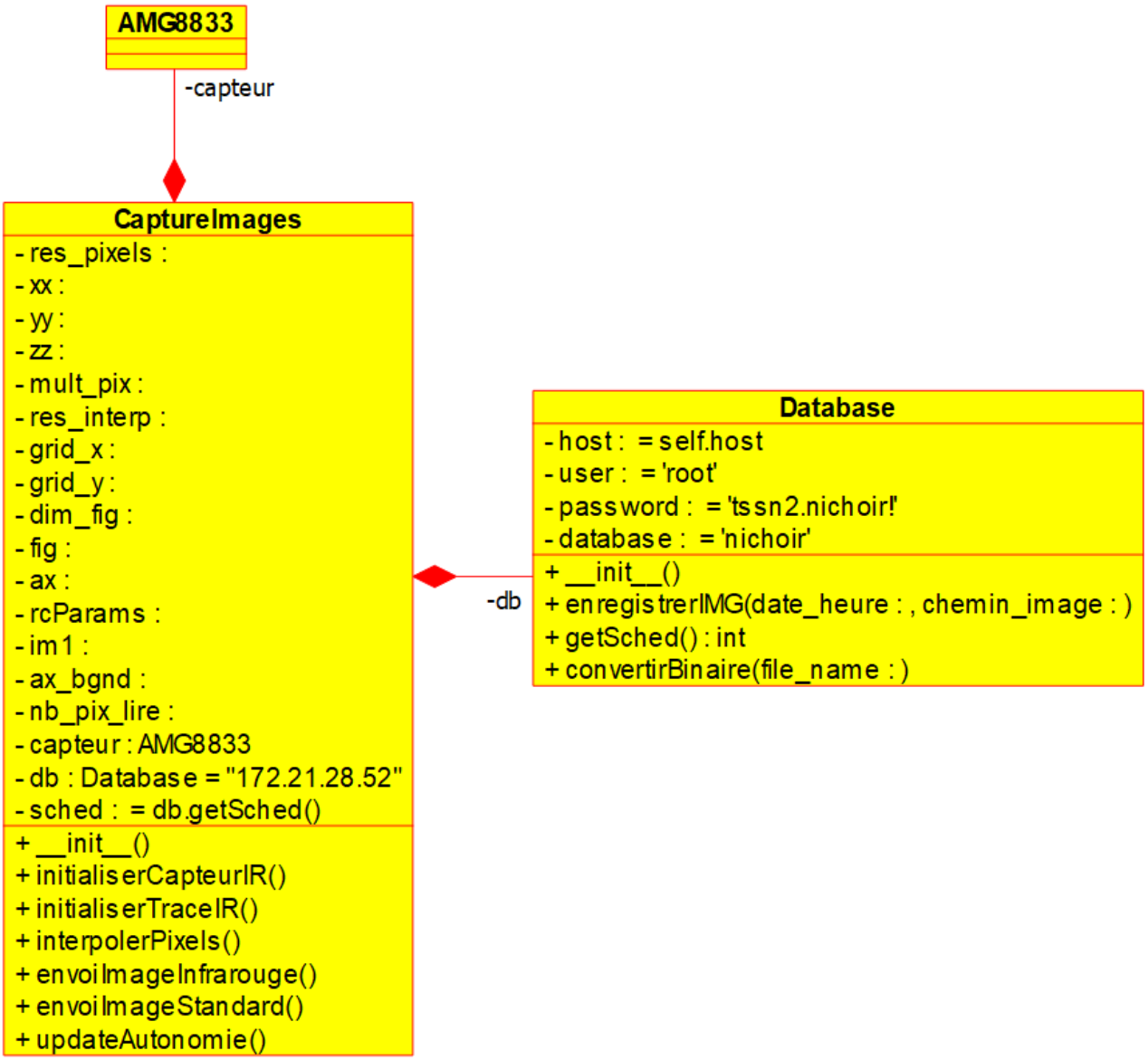
Pour récupérer les données des capteurs, j'ai besoin de bibliothèques qui connaissent le fonctionnement précis de chaque capteur, et qui utilisent des méthodes spécifiques pour relever les données. Toutes les bibliothèques que j'ai utilisées sont libres de droits et gratuites. Initialement, j'avais prévu de développer l'ensemble de mes programmes en C++ pour la récupération et l'enregistrement des données. Cependant, la bibliothèque en C++ de la caméra infrarouge AMG8833 était obsolète et donc inutilisable. J'ai trouvé une alternative en Python, ce qui explique la raison pour laquelle j'ai créé un programme en Python.

Diagramme des classes en C++



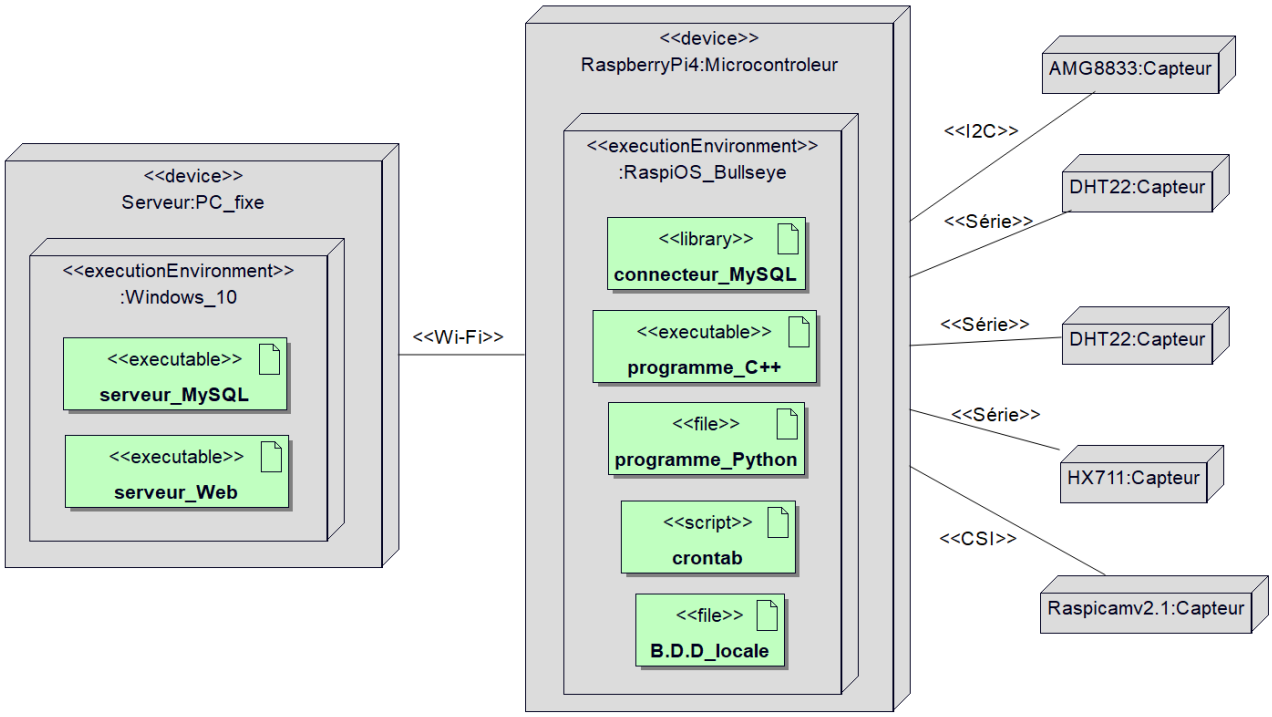
Les classes HX711 et DHT22 sont des bibliothèques de contrôle des capteurs dont je crée des objets à l'intérieur de la classe Mesures, qui permet de récupérer les données des capteurs et de les envoyer dans la base de données. Cela est possible en faisant le lien avec la classe Database, qui s'occupe de la connexion à la base de données et de l'enregistrement des données.

Diagramme des classes en Python



J'ai créé la classe CaptureImages en découpant le programme exemple de la bibliothèque AMG8833 pour capturer des images infrarouges. Cette classe permet aussi de capturer des images avec la caméra standard. Tout comme pour la classe Database en C++, j'ai développé une classe Database en Python qui s'occupe de la gestion de la connexion à la base de données, et l'enregistrement des données des caméras. Ce diagramme de classe est particulier, car Python est un langage non typé. Les attributs n'ont donc pas de types spécifiés, car ils sont du type de la valeur que nous leur attribuons.

Diagramme de déploiement



La carte Raspberry Pi 4 est connectée en Wi-Fi au PC fixe Windows 10 de la salle qui exécute le serveur de base de données. De son côté, la carte communique avec la base de données à l'aide d'un programme qui établit la connexion. Les capteurs de température et d'humidité DHT22 et de poids HX711 sont contrôlés par la carte à l'aide d'un fichier exécutable en C++. La caméra infrarouge AMG8833 et standard Raspicam v2.1 sont contrôlées à l'aide d'un programme en Python. Les supports de communication ne sont pas les mêmes en fonction des capteurs. Les deux DHT22 et le HX711 communiquent avec la carte à l'aide d'un protocole série. La caméra infrarouge AMG8833 communique en utilisant le protocole I2C et la caméra standard Raspicam v2.1 utilise l'interface CSI.

Mise en place

Carte Raspberry Pi 4

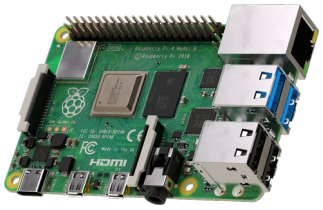


Figure 1: photo de la carte Raspberry Pi 4

La première chose à faire est d'avoir une nouvelle installation sur ma carte pour partir d'une base saine. La carte Raspberry Pi 4 a comme moyen de stockage une carte SD de 32Go, qui contient son système d'exploitation et l'ensemble des autres fichiers. Pour réinstaller la carte, il me suffit de récupérer la carte SD, la brancher à un lecteur sur un PC, et d'utiliser l'outil fourni par Raspberry pour installer la version du système d'exploitation que je souhaite utiliser. Les fichiers sont alors écrasés et le nouveau système est installé.

Puisque la carte est comme neuve, avant de commencer à l'utiliser pour piloter les capteurs, je dois activer certains paramètres. Je sais que ma carte devra communiquer en Wi-Fi sur le réseau, alors je la connecte au réseau Wi-Fi de la salle. Ensuite, certains bus de communication seront utilisés pour récupérer les données des capteurs, comme le bus I2C. Depuis les paramètres de la carte, j'active cette option. J'installe ensuite le pare-feu UFW qui permet de bloquer toutes les connexions par défaut, et j'autorise seulement les connexions en SSH. En effet, pour une utilisation plus pratique, je me connecte à la carte pour l'utiliser depuis un PC de la salle avec le protocole SSH. Enfin, je la mets à jour et la carte est prête à être utilisée.

Base de données distante et locale

Pour stocker les données de mes capteurs, je dois disposer d'une base de données distante. Avec mon coéquipier Alexis, nous décidons de l'installer sur un serveur MySQL UwAmp sur un PC fixe de la salle. Cette base de données dispose alors d'une table par capteur, et une pour les identifiants des utilisateurs.

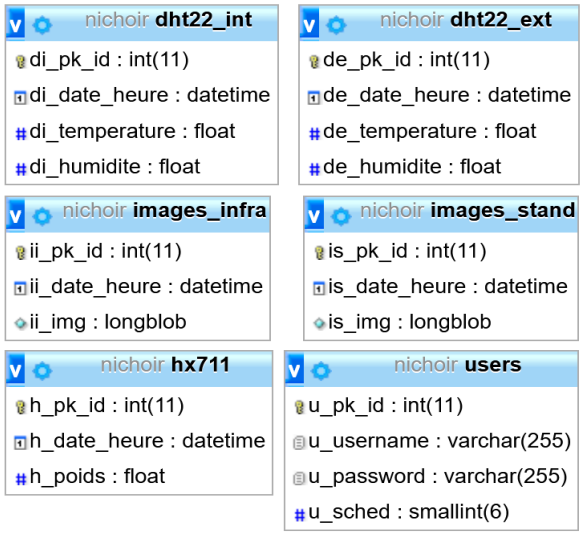


Figure 2: schéma de notre base de données distante

Ensuite, pour que ma carte puisse y accéder, je configure les droits de connexion à distance à la base de données. Enfin, pour des raisons de sécurité, je change le mot de passe par défaut par un mot de passe plus fort. Toutes les commandes détaillées sont disponibles en annexe 1.

Pour que les données récupérées ne soient pas perdues lorsque la connexion à la base de données distante est interrompue, j'installe sur la carte Raspberry une base de données locale SQLite qui permet de créer un fichier qui se comporte comme une base de données. Toutes les données récupérées par les capteurs seront enregistrées dans la base de données locale. Plus de détails sont donnés dans l'annexe 2.

Programmation des capteurs

L'ensemble des programmes que je vais écrire vont utiliser le principe de la programmation orientée objet. Il s'agit d'une façon de programmer qui consiste à créer un moule, qui va à son tour créer des objets. Le moule porte le nom de classe et les objets restent des objets. Par exemple, nous pourrions créer un moule « Véhicules », il s'agirait donc d'une classe. Ce moule peut créer des véhicules tels que des voitures, des camions ou des motos, ce sont les objets de la classe. Chaque objet a ses propres attributs, comme sa couleur, son prix, son type, etc. Par ailleurs, une classe dispose de méthodes qui sont des actions que peuvent réaliser ses objets. Par exemple, une méthode « Accélérer » pourrait permettre aux véhicules d'avancer et « Freiner » de s'arrêter. Dans mon cas, les classes, ou les moules, que je vais programmer vont servir à créer des capteurs, qui seront leurs objets, afin de les piloter en leur donnant des méthodes, ou actions qu'ils doivent réaliser, comme relever la température par exemple.

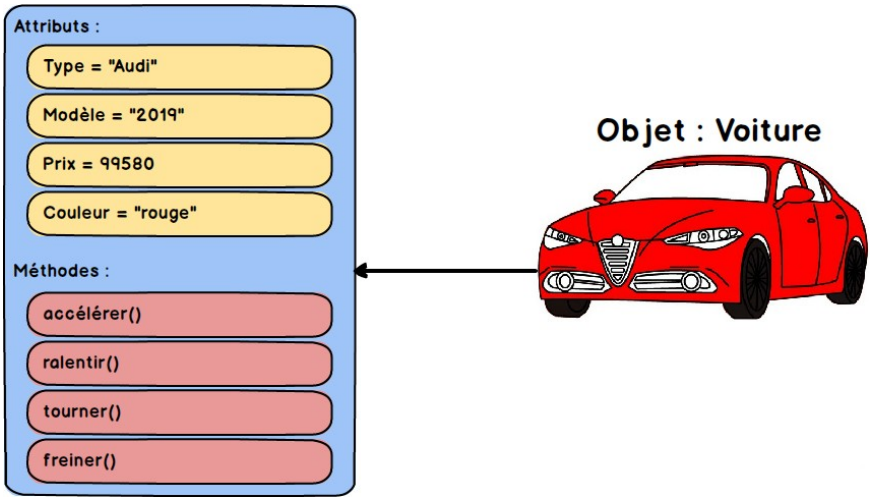


Figure 3: principe de la programmation orientée objet

Pour piloter mes capteurs, je vais créer une classe principale qui saura piloter les capteurs, récupérer leurs données, et communiquer avec une autre classe qui quant à elle permettra de se connecter à la base de données distante et d'enregistrer les données collectées.

Capteurs de température et d’humidité

Figure 4: photo du capteur DHT22

Les premiers capteurs que je vais utiliser sont les DHT22. Ils mesurent la température et l’humidité relative. Un sera utilisé à l’extérieur du nichoir et un à l’intérieur.

Sensibilité	Humidité relative : 0.1 % Température : 0.1°C
Plage de fonctionnement	Humidité relative : 0 – 100 % Température : -40 - 80°C

Pour les brancher sur la carte, j’utilise un shield. Il s’agit d’un module que je connecte sur la carte Raspberry pour utiliser des ports d’entrées-sorties plus simplement. Je branche mes deux capteurs sur deux ports série GPIO mis à disposition.

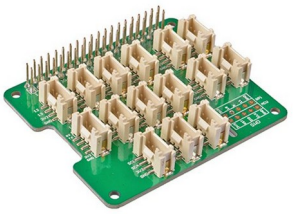


Figure 5: photo du shield de la carte Raspberry Pi 4

Afin d’utiliser les capteurs, je dois permettre à ma carte de décoder les valeurs qu’ils lui transmettent en communiquant avec elle. Je décide d’utiliser le langage de programmation C++, alors je cherche sur Internet une bibliothèque en C++, qui permet de traduire les données des capteurs en valeurs compréhensibles, et je l’installe. Je passe ensuite à l’écriture des programmes de récupération des valeurs. Enfin, j’installe une autre bibliothèque en C++ qui permet de créer une connexion entre ma carte et la base de données distante pour stocker les mesures et l’implémente dans un autre programme. Ce programme sert aussi à enregistrer les données dans la base de données locale. Je remarque que les données sont récupérées et enregistrées correctement. Le fonctionnement de mes programmes est validé, je passe alors au prochain capteur. En annexe 3 et 4 sont disponibles des parties des programmes commentés.

Capteur de poids

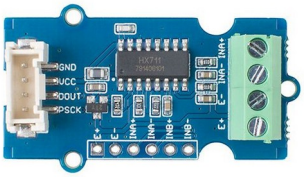


Figure 6: photo du capteur de poids HX711

Le capteur HX711 sert à mesurer le poids du nid. Il fonctionne à l'aide d'une cellule de charge 780g, ce qui suffit puisque les oiseaux pèsent peu. Cette cellule de charge est un rectangle d'aluminium composé de 4 jauges de contraintes. Une jauge de contrainte est une résistance dont la valeur varie en fonction de la déformation de la cellule de charge. La relation de la résistance d'une jauge de contrainte est indiquée ci-dessous.

$$R = \rho \frac{l}{S}$$

Figure 7: relation de la résistance d'une jauge de contrainte

p : résistivité du matériau en ohm-mètre

l : longueur du fil de résistance en mètres

S : section du fil en mètres carrés

Pour générer une tension de sortie, ces jauges de contraintes sont utilisées sur un pont de Wheatstone.

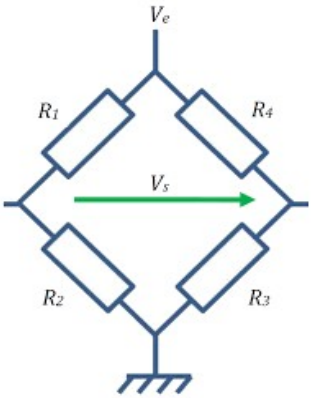


Figure 8: schéma électrique du pont de Wheatstone

La tension de sortie « Vs » est récupérée et traitée par le HX711. La cellule de charge est mise en équilibre entre deux planches en bois qui servent de faux fond et de vrai fond au nichoir, comme une balance.

Pour finir, je branche le capteur sur un port série GPIO du shield et j'installe une bibliothèque en C++ comme pour les capteurs DHT22. Une fois installée, j'utilise le même procédé. En m'aidant de la bibliothèque, j'écris un programme qui récupère la valeur du poids. J'agrémente aussi mon programme pour permettre l'enregistrement des données du capteur dans la base de données distante et locale. Je remarque que les données sont bien récupérées et enregistrées, son fonctionnement est alors validé. Je détaille des parties de mon programme pour ce capteur dans l'annexe 3.

Caméra infrarouge

L’intérieur du nichoir étant sombre, je vais alors utiliser la caméra infrarouge AMG8833, qui capte les ondes de chaleur. La seule bibliothèque fonctionnelle est en Python. Je prends le programme donné en exemple et l’adapte en fonction de mes besoins. Mon programme permet alors de générer des images avec la caméra infrarouge.

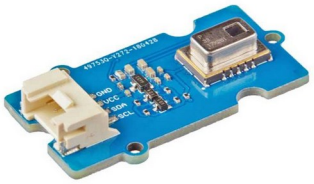


Figure 9: photo de la caméra infrarouge AMG8833

Cependant, la résolution du capteur est de 8x8 pixels et ne permet pas de capturer des images faciles à interpréter. Pour résoudre ce problème, la bibliothèque que j’ai choisie dispose d’une méthode qui permet d’interpoler l’image. L’interpolation consiste à générer des valeurs intermédiaires entre les valeurs qui ont été réellement prélevées pour obtenir une meilleure résolution sur l’image et ainsi la rendre plus facilement interprétable par l’utilisateur.

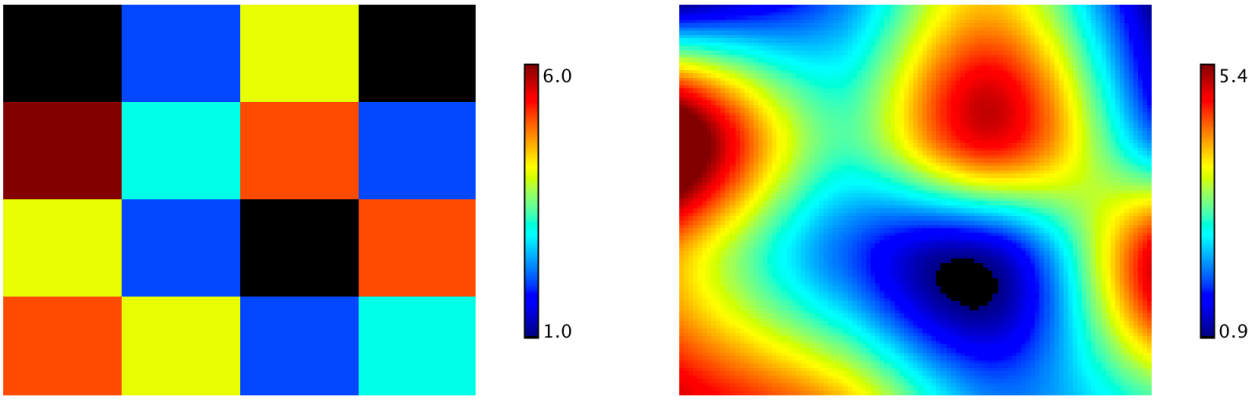


Figure 10: image infrarouge avant et après interpolation

Sur cet exemple, nous pouvons constater la différence entre une image infrarouge dont la résolution est de 4x4 pixels et la même image après interpolation, permettant ainsi d’obtenir une résolution bien plus élevée.

Comme pour les capteurs précédents, je dois aussi écrire un programme pour enregistrer les images dans la base de données distante et locale. Mon programme reprend le même mode de fonctionnement que celui utilisé pour les autres capteurs. Il me permet d’enregistrer les images infrarouges dans les bases de données. Des détails des programmes sont donnés dans les annexes 5 et 6.

Caméra standard

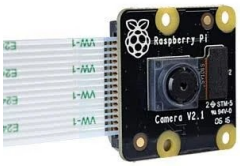


Figure 11: photo de la caméra standard Raspicam v2.1

Le système doit aussi permettre de récupérer des images standards à l'intérieur du nichoir. Pour faire cela, j'utilise une caméra Raspicam v2.1. Elle peut être utilisée grâce à des lignes de commandes, mais cela ne suffit pas pour l'enregistrer dans les bases de données. J'utilise alors le langage Python.

Puisque cette caméra est sensible à la lumière et non aux ondes de chaleur comme la caméra infrarouge, je dois régler sa sensibilité ISO afin que les images qu'elle capture ne soient pas trop sombres de façon à ce que nous puissions distinguer les oiseaux. Cette sensibilité contrôle la quantité de lumière que la caméra capture. Plus ce paramètre est élevé, plus la caméra capture la lumière. Il est possible de régler la sensibilité ISO à l'aide des lignes de commandes. Cependant, le terminal de la carte Raspberry m'indique que l'image générée par la caméra lorsque l'ISO est plus élevé est trop lourde pour être enregistrée. Je n'ai donc pas pu régler ce paramètre.

Figure 12: différence entre deux sensibilités ISO

Pour utiliser la caméra standard, j'agrémente le programme utilisé pour la caméra infrarouge. Le procédé est plus simple puisque cette caméra n'a pas besoin d'une bibliothèque pour fonctionner. L'enregistrement des images s'effectue sans erreur. La caméra standard est alors fonctionnelle.

Durant le développement des programmes des caméras, j'ai dû modifier la méthode d'enregistrement des images dans les bases de données. Initialement, j'avais prévu d'installer un serveur web sur la carte Raspberry et d'enregistrer le lien d'accès à l'image plutôt que l'image elle-même. Cependant, afin que le système soit autonome, la carte doit s'éteindre à intervalle régulier. La solution du serveur web n'était donc plus viable et j'ai choisi d'enregistrer les images dans les bases de données. Comme pour la caméra infrarouge, des détails des programmes sont donnés dans les annexes 5 et 6.

Récupération automatique des données

Maintenant que les programmes qui permettent de piloter les capteurs et les caméras sont écrits, il faut que je puisse les exécuter suivant les besoins. Il existe sur le système d'exploitation de la carte un programme, appelé crontab, qui permet d'exécuter des tâches de façon planifiée. Je synchronise les mesures et les prises d'images au redémarrage de la carte pour mettre en place le système d'autonomie.

```
# ┌──────────── minute (0 - 59)
# │ ┌──────────── heure (0 - 23)
# │ │ ┌──────────── jour du mois (1 - 31)
# │ │ │ ┌──────────── mois (1 - 12)
# │ │ │ │ ┌──────────── jour de la semaine (6 - 0) (du dimanche au samedi ;
# │ │ │ │ │                                    7 est aussi le dimanche sur certains systèmes)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * commande à exécuter
```

Figure 13: syntaxe d'exécution d'une tâche crontab.

Voici un exemple clair et simple de la syntaxe du fichier de configuration crontab pour l'exécution d'une commande de manière programmée. Chaque étoile correspond à un champ pour l'heure, les minutes, les secondes ou la date. En faisant des essais à quelques minutes d'intervalles seulement, j'ai remarqué que les données sont récupérées et enregistrées sans erreur dans la base de données distante et locale de façon automatique, ce qui me permet de valider le fonctionnement de ce programme. La syntaxe du fichier crontab que j'ai utilisé est détaillé dans l'annexe 7.

Gestion de l’alimentation

Pour la gestion de l’arrêt et de l’allumage programmés de la carte Raspberry Pi 4, j’utilise le module Witty Pi 4. Il s’agit d’une carte que je viens brancher entre la carte Raspberry et le shield.

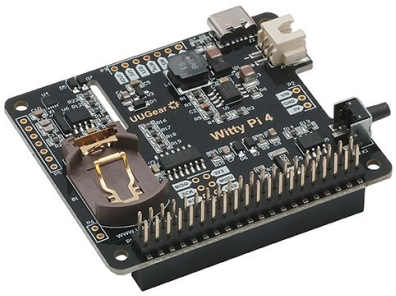


Figure 14: photo de la carte Witty Pi 4

J’installe les fichiers de configuration et me rends sur l’interface web de la carte Witty Pi 4 pour paramétrer les allumages et les arrêts de la carte Raspberry. Je décide que par défaut le système s’allumera toutes les 25 minutes pendant 5 minutes afin de relever les données en consommant le moins possible. L’utilisateur pourra aussi choisir entre d’autres options d’allumage. Tous les détails sont donnés dans l’annexe 8.

Ensuite, à l’aide d’un module USB, je relève en temps réel la consommation de la carte Raspberry pendant une heure, sans avoir lancé les arrêts et démarrages programmés. Grâce à ces données, je vais pouvoir calculer la capacité de la batterie nécessaire ainsi que la taille de la cellule photovoltaïque.

La carte est alimentée par une tension 5V et consomme 3W en une heure. En une journée, la carte consomme donc 72W.

Puisque la carte est allumée seulement 2 heures par jour, je divise la consommation par 12. En un jour, la carte consomme donc réellement 6Wh.

Pour 5 jours d’autonomie, l’énergie nécessaire est donc de 30Wh. Afin de connaître la capacité de la batterie 12V, j’utilise la formule C = E/U avec E l’énergie et U la tension. Cela donne 2,5Ah, il faut donc une batterie de 2500mAh minimum pour tenir 5 jours sans interruption et sans recharge. La batterie dont je dispose a une capacité de 7000mAh à 12V, ce qui suffit.

La cellule photovoltaïque dont je dispose mesure 35,6cm par 25,3cm. Cette surface permet de récupérer 1000W/m². Avec sa surface, je peux récupérer 0.356*0.253*1000 = 90,068W.

Sa puissance max est P = 10W, son rendement est alors le suivant 10/90,068 = 0,11. Le rendement de cette cellule photovoltaïque est de 11 %. Puisque je récupère 0,2kWh/m² par jour en décembre, je multiplie cette valeur par le rendement, 200*0,11 = 22Wh.

Puisque le système a besoin de 6Wh par jour pour fonctionner, l’énergie que peut produire cette cellule est suffisante.

Mise en autonomie

Figure 15: photo du contrôleur de charge

Pour que le système soit entièrement autonome, je dois relier la batterie, la cellule photovoltaïque et la carte Witty Pi 4. J'utilise alors le contrôleur de charge Steca Solsum 6.6c. Il dispose d'un pôle positif et d'une masse pour chacun des éléments concernés.

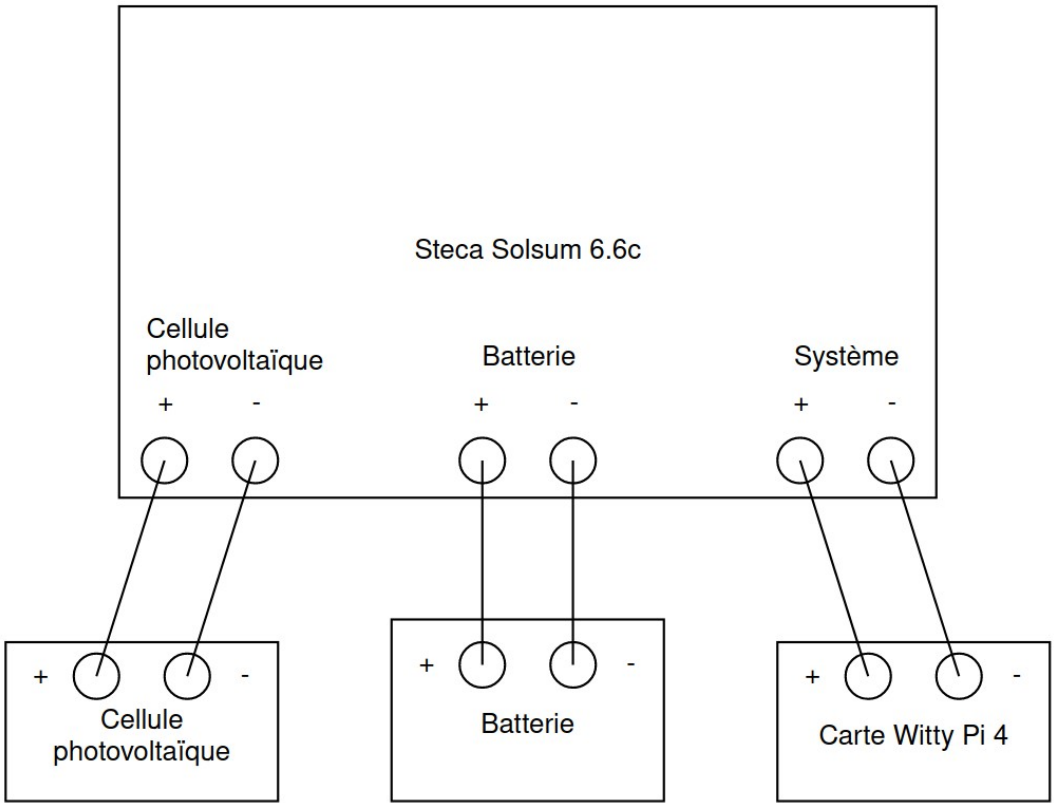


Figure 16: schéma des branchements des éléments gérant l'autonomie

Les branchements sont simples, tout comme le fonctionnement. Le contrôleur de charge est équipé de deux DEL, dont la couleur varie entre vert jaune et rouge, en fonction de la charge de la batterie et de l'énergie délivrée par la cellule photovoltaïque. Cela permet un suivi clair de l'autonomie du système.

Plan de tests

Tests unitaires

Système concerné : Montage carte Raspberry Pi 4 et capteurs

Unités testées : Capteurs DHT22 et HX711

N°	Action	Attendu	Résultat
1	Exécution du programme d'acquisition et d'enregistrement des données issues des capteurs.	Les mesures sont récupérées et enregistrées dans la base de données locale et distante.	Le résultat s'est déroulé comme attendu.
2	Exécution du programme d'acquisition et d'enregistrement des données issues des capteurs lorsque la carte est hors connexion.	Les mesures sont récupérées et uniquement enregistrées dans la base de données locale.	Le résultat s'est déroulé comme attendu.

Système concerné : Montage carte Raspberry Pi 4 et caméras

Unités testées : Caméra standard Raspicam v2.1 et caméra infrarouge AMG8833

N°	Action	Attendu	Résultat
1	Capturer une image standard.	L'image permet d'avoir une bonne visibilité dans le noir.	Le résultat ne s'est pas déroulé comme attendu. Le fichier est trop lourd pour être enregistré.
2	Capturer une image infrarouge.	L'image a une résolution suffisante.	Le résultat ne s'est pas déroulé comme attendu, la résolution est mauvaise.
3	Lancement du programme de prise et d'enregistrement des images.	Les images sont capturées et enregistrées dans la base de données locale et distante.	Le résultat s'est déroulé comme attendu.
4	Lancement du programme de prise et d'enregistrement des images lorsque la carte est hors connexion.	Les images sont capturées et uniquement enregistrées dans la base de données locale.	Le résultat s'est déroulé comme attendu.

Tests d’intégration

Systèmes concernés : Carte Raspberry Pi 4 et serveur de base de données distant

Unités testées : Capteurs DHT22 et HX711

N°	Action	Attendu	Résultat
1	Exécution du programme d’acquisition et d’enregistrement des données issues des capteurs avec des identifiants autorisés.	La base de données distante valide la connexion et reçoit les requêtes envoyées par la carte Raspberry Pi 4. La console affiche les requêtes et valide l’enregistrement des données.	Le résultat s’est déroulé comme attendu.
2	Exécution du programme d’acquisition et d’enregistrement des données issues des capteurs avec des identifiants erronés.	La base de données distante envoie un message d’erreur lors de la connexion de la carte Raspberry Pi 4. À chaque requête dans la console, une erreur s’affiche pour signaler le problème. Les données ne sont pas enregistrées dans la base de données distante.	Le résultat s’est déroulé comme attendu.
3	Exécution du programme d’acquisition et d’enregistrement des données issues des capteurs lorsque la carte est hors connexion.	La console affiche un message d’erreur indiquant que la connexion est impossible. À chaque requête dans la console, une erreur s’affiche pour signaler le problème. Les données ne sont pas enregistrées dans la base de données distante.	Le résultat s’est déroulé comme attendu.

Systèmes concernés : Carte Raspberry Pi 4 et serveur de base de données distant

Unités testées : Caméra standard Raspicam v2.1 et caméra infrarouge AMG8833

N°	Action	Attendu	Résultat
1	Lancement du programme de prise et d'enregistrement des images avec des identifiants autorisés.	La base de données valide la connexion et reçoit les requêtes envoyées par la carte Raspberry Pi 4. La console valide l'enregistrement des images.	Le résultat s'est déroulé comme attendu.
2	Lancement du programme de prise et d'enregistrement des images avec des identifiants erronés.	La base de données envoie un message d'erreur à la carte Raspberry Pi 4. La console indique que les images n'ont pas été enregistrées à distance.	Le résultat s'est déroulé comme attendu.
3	Exécution du programme d'acquisition et d'enregistrement des images lorsque la carte est hors connexion.	La console affiche un message d'erreur expliquant que la connexion est impossible et indique que les images n'ont pas été enregistrées à distance.	Le résultat s'est déroulé comme attendu.

Finalité

Ma partie du projet s'est terminée plus tôt que la durée estimée dans le diagramme de Gantt prévisionnel. Il m'a fallu moins de 3 mois pour faire fonctionner le système à la base du projet, permettant ainsi à mes coéquipiers d'accéder aux données que relèvent les différents capteurs.

Le cahier des charges est respecté. Le système fait le relevé de la température et de l'humidité à l'intérieur et à l'extérieur du nichoir, ainsi que du poids des oiseaux. Des images infrarouges et standard sont aussi capturées. Toutes les données sont récupérées à intervalles réguliers et sont sauvegardées dans une base de données distante et locale sécurisée. Le système est lui aussi sécurisé pour éviter les intrusions.

Enfin, le système est autonome et peut ainsi fonctionner sans interruption. Ses arrêts et démarrages sont programmés, ce qui permet de faire des économies d'énergie. La cellule photovoltaïque et la batterie ont des caractéristiques calculées pour que le système ne soit pas interrompu.

Le coût a été pensé pour être le plus faible possible. Les dépenses concernent le matériel et non la programmation et le paramétrage logiciel. En effet, aucune des bibliothèques que j'ai utilisée n'est payante. De ce fait, le coût estimé du projet se situe entre 377,87€ et 577,87€.

Conclusion

En conclusion, durant les 5 mois précédents, le projet Nichoir Autonome Connecté m'a beaucoup apporté. J'ai acquis de nouvelles connaissances en programmant de nouveaux capteurs et j'ai pu approfondir celles que j'avais déjà acquises en utilisant des capteurs que je connaissais déjà. Par ailleurs, j'ai appris à mettre un système en autonomie ainsi qu'à assurer ses connexions avec d'autres appareils de façon sécurisée.

Ce projet m'a aussi permis d'en apprendre plus sur les moyens d'organisation en équipe. J'ai appris à confectionner un diagramme de Gantt prévisionnel, qui nous a guidé mes coéquipiers et moi dans l'élaboration de notre projet. De plus, j'ai appris à me servir de GitHub, sur navigateur web et en lignes de commandes, un outil qui s'est montré indispensable au suivi de nos tâches et à la sauvegarde en ligne des différentes parties de notre projet.

Par dessus tout, ce projet représente l'aboutissement de ma deuxième et dernière année de BTS Systèmes Numériques. Il s'agit du dernier examen de l'année, qui vient donc conclure mes deux années d'études au sein de cette formation. Le projet a donc été une expérience enrichissante.

Annexes

Annexe 1 – Commandes de la base de données distante

```
mysql> GRANT ALL PRIVILEGES ON nichoir.* TO 'root'@'172.21.28.%' IDENTIFIED BY 'root';
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

Pour autoriser l'accès à la base de données distante depuis le même réseau, j'utilise la commande ci-dessus. J'accorde ainsi tous les droits (connexion, lecture, écriture) à l'utilisateur dont l'identifiant est root et le mot de passe par défaut est root sur le réseau local.

```
mysql> SELECT User, Host FROM mysql.user;
+-----------+--------------+
| User      | Host         |
+-----------+--------------+
| root      | 172.21.28.%  |
| root      | 172.21.28.70 |
| mysql.sys | localhost    |
| root      | localhost    |
| uwamp     | localhost    |
+-----------+--------------+
5 rows in set (0.00 sec)
```

Ensuite, pour modifier les mots de passe, j'établis la liste des utilisateurs avec la commande ci-dessus. Je remarque trois utilisateurs root, chacun pour un accès différent, mais dont les mots de passe doivent être modifiés.

```
mysql> ALTER USER root@localhost
    -> IDENTIFIED BY 'tssn2.nichoir!';
Query OK, 0 rows affected (0.00 sec)
```

Voici un exemple de modification du mot de passe de l'utilisateur root sur la machine locale. Le mot de passe par défaut est changé par un mot de passe plus sécurisé « tssn2.nichoir! ». Enfin, pour valider les opérations, je dois écrire la commande « FLUSH PRIVILEGES; » pour mettre à jour les droits des utilisateurs.

Annexe 2 – Fonctionnement de la base de données locale

Comme expliqué précédemment, la base de données locale SQLite est une copie conforme de la base de données distante.

```
sudo apt install sqlite3
```

L'installation sur la carte se fait simplement avec la commande ci-dessus. Par la suite, nous pouvons utiliser des commandes pour insérer des données dans la base de données locale, qui correspond à un fichier dont l'extension est « db » pour « Database » qui signifie « Base de données ». Dans mon cas, j'ai nommé la base de données locale « dbLocale.db ».

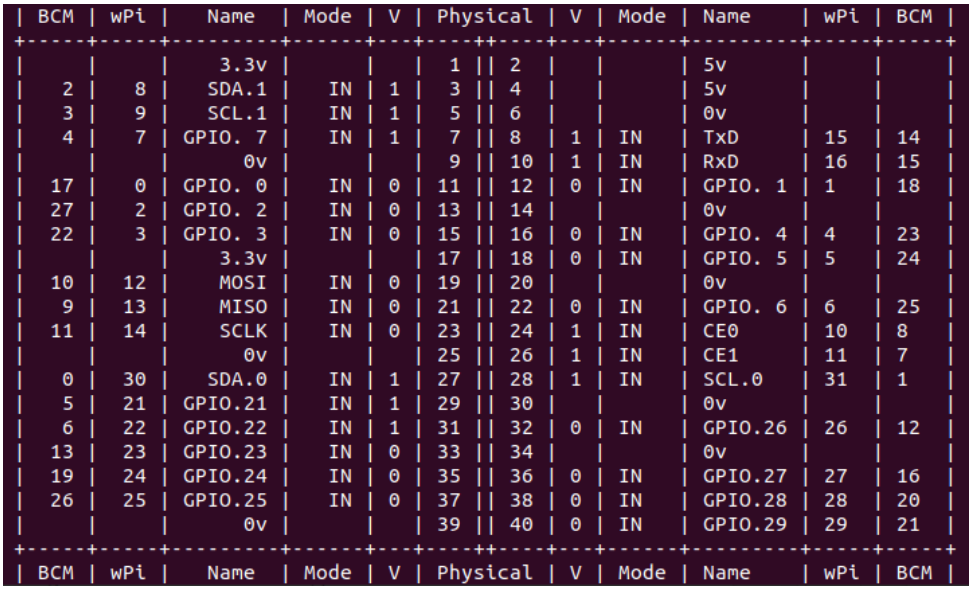
```
sqlite3 dbLocale.db
```

Une fois la base de données ouverte, je peux exécuter des commandes SQL pour créer les tables conformément à la base de données distante afin d'ajouter les données récupérées par mes programmes.

```
sqlite> CREATE TABLE images_infra(ii_pk_id INT PRIMARY KEY NOT
NULL AUTOINCREMENT, ii_date_heure DATETIME, ii_img LONGBLOB);
```

Par exemple, voici la commande qui permet de créer la table « images_infra » qui sert à stocker les images infrarouges. Mon programme devra donc renseigner la date et l'heure ainsi que les données de l'image. L'identifiant « ii_pk_id » ne sera pas à renseigner, car comme dans la base de données distante, il s'auto-incrémente.

Annexe 3 – Programme d’acquisition des données des capteurs



BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1 || 2			5v		
2	8	SDA.1	IN	1	3 || 4			5v		
3	9	SCL.1	IN	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	1	IN	TxD	15	14
		0v			9 || 10	1	IN	RxD	16	15
17	0	GPIO. 0	IN	0	11 || 12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13 || 14			0v		
22	3	GPIO. 3	IN	0	15 || 16	0	IN	GPIO. 4	4	23
		3.3v			17 || 18	0	IN	GPIO. 5	5	24
10	12	MOSI	IN	0	19 || 20			0v		
9	13	MISO	IN	0	21 || 22	0	IN	GPIO. 6	6	25
11	14	SCLK	IN	0	23 || 24	1	IN	CE0	10	8
		0v			25 || 26	1	IN	CE1	11	7
0	30	SDA.0	IN	1	27 || 28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29 || 30			0v		
6	22	GPIO.22	IN	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	IN	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37 || 38	0	IN	GPIO.28	28	20
		0v			39 || 40	0	IN	GPIO.29	29	21

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM

Puisque j’utilise la bibliothèque WiringPi, qui permet de communiquer avec les capteurs dans mon programme, je récupère les numéros correspondants aux ports physiques sur lesquels j’ai connecté les deux capteurs DHT22 afin de m’en servir dans mon programme. Mes capteurs se trouvent sur les ports 5 et 16 du shield, qui correspondent aux ports WiringPi 21 et 27.

```
26          TDHT22 *dht22_ext = new TDHT22(21);
27          TDHT22 *dht22_int = new TDHT22(27);
```

Dans mon programme, j’instancie donc les objets représentant mes capteurs à l’aide de la bibliothèque DHT22 sous la forme de pointeurs. Cela me permet de faire de l’allocation dynamique, qui consiste à réserver manuellement de l’espace en mémoire pour mes capteurs et ainsi d’avoir un meilleur contrôle de la mémoire utilisée.

```
41              dht22_ext->Init(); //Initialisation
42              dht22_ext->Fetch(); //Récupération des données
43              temp_ext = dht22_ext->Temp; // Affectation dans les variables attributs
```

Toujours à l’aide de la bibliothèque, je récupère la température et l’humidité et les stocke dans des variables pour les traiter.

```
28          SimpleHX711 *hx = new SimpleHX711(19, 18, 2589, 1435385);
```

Pour le capteur de poids HX711, j'inclus la bibliothèque et je crée l'objet qui me permet de piloter le capteur sous la forme d'un pointeur toujours pour gérer l'espace mémoire. À l'aide d'un programme fourni dans la bibliothèque, j'ai pu calibrer le capteur. Dans l'ordre, la valeur 19 correspond au pin d'envoi des données, 18 au signal d'horloge pour synchroniser l'envoi des données, 2589 est une valeur de référence donnée lors du calibrage et 1435385 est la valeur qui correspond à 0, aussi appelée offset.

```
44              hr_ext = dht22_ext->Hum;
45              if(temp_ext == 0 || hr_ext == 0){
46                      temp_ext = dht22_ext->Temp;
47                      hr_ext = dht22_ext->Hum;
48              }
```

Ensuite, tout comme pour les capteurs DHT22, je récupère la valeur du poids. Pour éviter les valeurs incohérentes, qui peuvent apparaître au redémarrage du système, j'indique que si la valeur vaut 0, la mesure doit être reprise pour s'assurer qu'elle est correcte.

```
14      Mesures::~Mesures()
15      {
16          delete dht22_ext;
17          delete dht22_int;
18          delete hx;
```

Enfin, lorsque l'acquisition des données est terminée, je supprime les capteurs pour libérer l'espace mémoire.

Annexe 4 – Programme d'enregistrement des mesures dans les bases de données

```
11              MYSQL *mysql1;
12              const char *database_name = "nichoir";
13              const char *database_username = "root";
14              const char *database_password = "tssn2.nichoir!";
```

Tout d'abord, je crée une classe Database en C++ avec comme attributs les identifiants de connexion à la base de données. La base de données est ici représentée par le pointeur sur un objet mysql1, crée à l'aide du connecteur C++/MySQL, qui est une bibliothèque permettant d'établir la connexion à la base de données distante et l'enregistrement des données.

```
19              void initialiser(const char* database_ip);
20              void sauverDonneesDHT22(std::string dh, float temp, float hr, bool ext);
21              void sauverDonneesHX711(std::string dh, float poids);
```

La classe dispose de 3 méthodes, qui permettent d'initialiser la base de données en fonction de l'adresse IP du serveur, ainsi que de sauvegarder localement et à distance les données des capteurs DHT22 et HX711 à l'aide de requêtes SQL.

```
30          Database *db = new Database; // Création de la base de données
```

```
 8      Mesures::Mesures(){
 9          db->initialiser("172.21.28.52");
10      }
```

```
19              delete db;
20      }
```

Dans ma classe principale, j'initialise donc ma connexion à la base de données sous la forme d'un pointeur. La connexion est alors établie lorsque les mesures sont lancées (constructeur), et supprimée lorsque les mesures sont terminées (destructeur).

Annexe 5 – Programme de capture des images des caméras

```
66              chemin = "/home/nichoir/images/img_infra.jpg"
67              plt.savefig(chemin)
```

Après avoir été interpolée, l'image de la caméra est enregistrée avec la méthode savefig(chemin) du module plt dans le répertoire /home/nichoir/images/img_infra.jpg.

```
72              chemin = "/home/nichoir/images/img_stand.jpg"
73              os.system("libcamera-still -o {}".format(chemin))
```

Le procédé est similaire pour la caméra standard, mais nécessite d'utiliser une ligne de commande, qui est exécutée par la méthode system(command) du module os. La commande « libcamera-still -o [chemin] » permet donc d'enregistrer l'image dans le répertoire souhaité, qui est /home/nichoir/images/img_stand.jpg ».

```
11          def convertirBinaire(self,file_name):
12              with open(file_name, 'rb') as file:
13                  binary_data = file.read()
14              return binary_data
```

Puisque les images doivent être converties en binaire pour être enregistrées dans la base de données, j'ai créé une méthode qui permet d'ouvrir une image et d'écrire son contenu dans une chaîne de caractères, ici appelée « binary_data ». Cette variable contient donc les données binaires d'une image et sera envoyée dans la base de données pour stocker l'image.

Annexe 6 – Programme d'enregistrement des images dans les bases de données

```
4   class Database:
5       def __init__(self, host):
6           self.host = host
7           self.user = "root"
8           self.password = "tssn2.nichoir!"
9           self.database = "nichoir"
```

Tout comme pour les capteurs développés en C++, afin de gérer l'enregistrement des images dans la base de données distante, j'utilise une classe que j'appelle Database à laquelle je renseigne les informations de connexion.

```
32          query = "INSERT INTO images_infra(ii_date_heure, ii_img) VALUES(%s,%s)"
33          cursor1.execute(query, (date_heure,self.convertirBinaire(chemin_image)))
```

```
35          req_sqlite = "INSERT INTO images_infra(ii_date_heure, ii_img) VALUES(?,?)"
36          donnees = (date_heure, self.convertirBinaire(chemin_image))
37          cursor2.execute(req_sqlite, donnees) # Enregistrement local
```

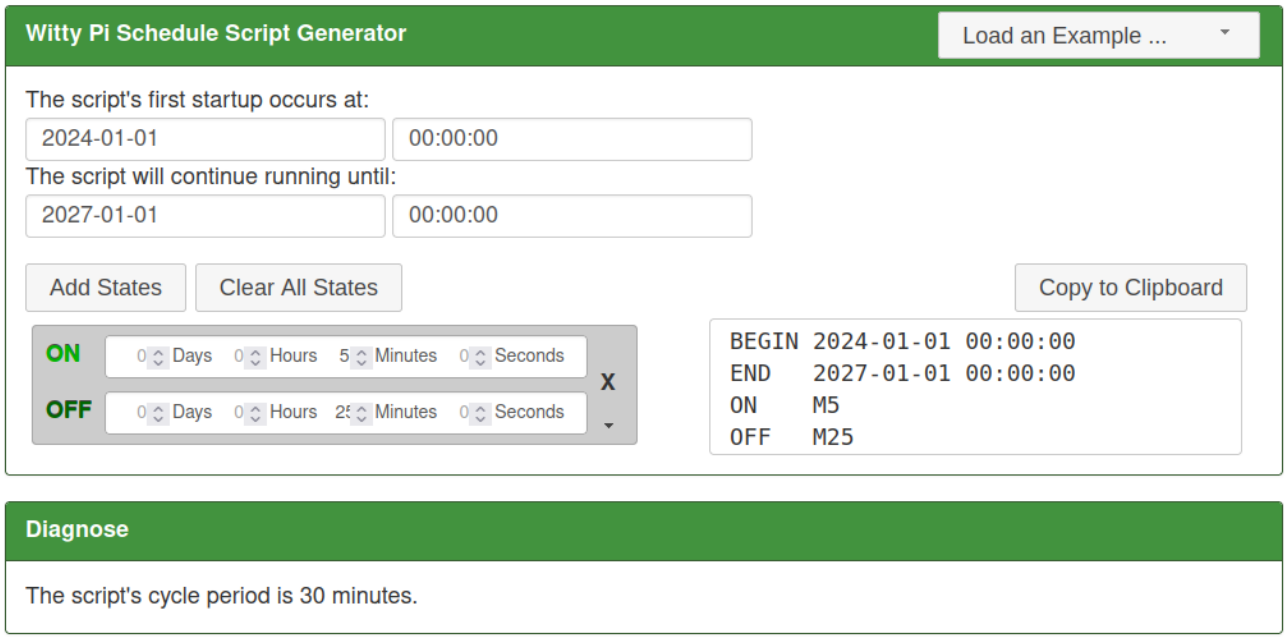
Enfin, la classe dispose d'une méthode qui contient les requêtes qui permettent l'enregistrement des images dans la base de données locale et distante et les exécute par l'intermédiaire d'un curseur qui établit la connexion aux bases de données. Le curseur « cursor1 » lance la connexion à la base de données distante et « cursor2 » à la base de données locale.

Annexe 7 – Syntaxe de la récupération automatique des données

```
# ----- Prise d'image infrarouge et standard au redémarrage -----
@reboot sleep 60;/usr/bin/python3 /home/nichoir/projet_nichoir/python/main.py

# ----- Récupération et envoi des données des capteurs au redémarrage -----
@reboot sleep 60;/home/nichoir/projet_nichoir/cpp/./make
```

Ce programme crontab permet de lancer la capture d'images, la récupération des mesures et l'enregistrement de l'ensemble des données dans la base de données distante et locale. Avec la commande '@reboot sleep 60;', le programme s'exécute 60 secondes après le démarrage de la carte pour laisser le temps à tous les services de s'activer.

Annexe 8 – Script d’allumage et d’arrêt programmés



Je choisis la durée durant laquelle la carte est allumée et elle est éteinte à partir d’une date. Je récupère dans la section de droite le script sous forme de texte, que je rentre dans le gestionnaire de la carte Witty Pi 4, à l’endroit prévu. L’état passe alors en « in use », ce qui signifie que le script est pris en compte.

Dans cet exemple, la carte Witty Pi 4 s’occupera d’allumer le système pendant 5 minutes toutes les 25 minutes, ce qui correspond à un cycle de 30 minutes. Cette configuration permet de solliciter le système seulement 2 heures par jour, réduisant considérablement l’énergie consommée.

```
76                query = "SELECT u_sched FROM users WHERE u_username = 'admin'"
77                cursor.execute(query)
```

La classe Database en Python dispose d’une méthode qui peut récupérer l’attribut « u_sched » qui correspond à un entier et qui fait partie de la table « users » de la base de données. Cet attribut est propre au compte administrateur et permet de faire varier le script de démarrage et d’arrêt programmé.

Ensuite, la classe principale de capture des images récupère cet attribut par l’intermédiaire de sa connexion à la base de données et fait varier les redémarrages de la carte. Si l’attribut vaut 1, la carte s’allume 5 minutes toutes les 25 minutes. S’il vaut 2, la carte s’allume 5 minutes toutes les 55 minutes. Enfin, si l’attribut vaut 3, la carte s’allume 5 minutes toutes les 115 minutes.

Programmes complets

C++

mesures.h

```
#ifndef MESURES
#define MESURES

#include <hx711/common.h>
#include <iostream>
#include <fstream>
#include "DHT22.h"
#include "database.h"
#include <sys/time.h>
#include <cmath>

using namespace HX711;

class Mesures
{

private:
// ----- Déclaration des attributs pour les mesures -----
    float temp_ext;
    float temp_int;
    float hr_ext;
    float hr_int;
    float poids;

// ----- Création des capteurs -----
    TDHT22 *dht22_ext = new TDHT22(21);
    TDHT22 *dht22_int = new TDHT22(27);
    SimpleHX711 *hx = new SimpleHX711(19, 18, 2589, 1435385);

    Database *db = new Database; // Création de la base de données

public:
    Mesures();
    ~Mesures();

    std::string dateHeure();
    void envoiDonneesDHT22(bool ext);
    void envoiDonneesHX711();
};

#endif
```

mesures.cpp

```
#include "mesures.h"

using namespace std;
using namespace HX711;

// --- Partie à modifier par l'utilisateur si besoin ---

Mesures::Mesures(){
    db->initialiser("172.21.28.52"); //Modifier par l'adresse IP de votre
serveur de base de données
}

// --- Cette partie ne doit pas être modifiée ---

Mesures::~Mesures()
{
    delete dht22_ext;
    delete dht22_int;
    delete hx;
    delete db;
}

string Mesures::dateHeure() // Méthode qui récupère la date et l'heure dans le
bon format
{
    static int seconds_last = 99;
    char TimeString[128];

    timeval curTime;
    gettimeofday(&curTime, NULL);

    seconds_last = curTime.tv_sec;
    strftime(TimeString, 80, "%Y-%m-%d %H:%M:%S", localtime(&curTime.tv_sec));
    string dateHeure(TimeString);
    return dateHeure;
}


void Mesures::envoiDonneesDHT22(bool ext)
{
    string date_Heure(dateHeure());
    if(ext == true){
        dht22_ext->Init(); //Initialisation
        dht22_ext->Fetch(); //Récupération des données
        temp_ext = dht22_ext->Temp; // Affectation dans les variables attributs
        hr_ext = dht22_ext->Hum;
        if(temp_ext == 0 || hr_ext == 0){
                temp_ext = dht22_ext->Temp;
                hr_ext = dht22_ext->Hum;
        }
        db->sauverDonneesDHT22(date_Heure,temp_ext,hr_ext, ext);
//Enregistrement dans la base de données locale et distante
    }
    if(ext == false){
        dht22_int->Init();
        dht22_int->Fetch();
        temp_int = dht22_int->Temp;
        hr_int = dht22_int->Hum;
        if(temp_int == 0 || hr_int == 0){
```

```
                temp_int = dht22_int->Temp;
                hr_int = dht22_int->Hum;
        }
        db->sauverDonneesDHT22(date_Heure,temp_int,hr_int, ext);
//Enregistrement dans la base de données locale et distante
    }
}

void Mesures::envoiDonneesHX711()
{
// ----- Récupération de la valeur du poids en grammes sur 21 échantillons
-----
    hx->setUnit(Mass::Unit::G);
    poids = hx->weight(21);
    if(poids <= 0){poids = 0;}

// ----- Enregistrement dans la base de données locale et distante -----
    db->sauverDonneesHX711(dateHeure(),ceil(poids*100)/100);
    db->sauverDonneesLocalesHX711(dateHeure(),ceil(poids*100)/100);
}
```

database.h

```
#ifndef DATABASE
#define DATABASE

#include <mysql/mysql.h>
#include <sqlite3.h>
#include <string>

class Database{
        private:

        MYSQL *mysql1;
        const char *database_name = "nichoir";
        const char *database_username = "root";
        const char *database_password = "tssn2.nichoir!";

        public:
        Database();
        ~Database();
        void initialiser(const char* database_ip);
        void sauverDonneesDHT22(std::string dh, float temp, float hr, bool
ext);
        void sauverDonneesHX711(std::string dh, float poids);
};

#endif
```

database.cpp

```
#include "database.h"
#include <cstddef>
#include <iostream>

using namespace std;

Database::Database()
{
```

```
  //On initialise l'objet de la classe pour les connexions à la base de données
        mysql1 = mysql_init(NULL);
}

Database::~Database()
{
    mysql_close(mysql1);
    cout << endl << "Déconnexion de la base de données." << endl;
}


void Database::initialiser(const char* database_ip)
{
 if(mysql1 == NULL)
    {
        cout << mysql_error(mysql1);
        return;
    }

    //Connexion à la base de données
    if(mysql_real_connect(mysql1, database_ip, database_username,
database_password, database_name, 0, NULL, 0) == NULL)
    {
        cout << "La connexion à la base de données à échoué. Code d'erreur ci-
dessous." << endl;
        cout << mysql_error(mysql1) << endl << endl;
        cout << "Fin du programme." << endl;
        exit(1);
    }
    else
    {
        cout << "Connexion à la base de données effectuée avec succès." <<
endl;
    }
}

void Database::sauverDonneesDHT22(string dh, float temp, float hr,bool ext)
{
    sqlite3 *db;
    sqlite3_stmt *stmt;

    if(ext == true)
    {
        //Création de la requête dans le bon format
        string req = "INSERT INTO
dht22_ext(de_date_heure,de_temperature,de_humidite) VALUES ('" + dh + "'," +
to_string(temp) + "," + to_string(hr) + ");";
        //Affichage de la requête dans le terminal
        cout << endl << "Requête en direction de la table : dht22_ext" << endl
<< ">> " <<  req << endl;
        const char *requete = req.c_str();
        if(mysql1 != NULL)
        {
                //Exécution de la requête
                if(mysql_query(mysql1, requete ))
                {
                //Affichage du code d'erreur si un problème est survenu
                cout << stderr << "%s\n" << mysql_error(mysql1);
                }
         }
```

```
         if (sqlite3_open("/home/nichoir/projet_nichoir/dbLocale.db", &db) ==
SQLITE_OK)
         {
             sqlite3_prepare(db, req.c_str(), -1, &stmt, NULL ); //Préparation
de la requête
             sqlite3_step( stmt ); //Exécution de la requête
             cout << endl << "Requête en direction de la table locale :
dht22_ext" << endl << ">> " <<  req << endl;
         }
         else{cout << "Impossible d'ouvrir la table\n";}

         sqlite3_finalize(stmt);
         sqlite3_close(db);
    }
    if(ext == false)
    {
        //Création de la requête dans le bon format
        string req = "INSERT INTO
dht22_int(di_date_heure,di_temperature,di_humidite) VALUES ('" + dh + "'," +
to_string(temp) + "," + to_string(hr) + ");";
        //Afficahge de la requête dans le terminal
        cout << endl << "Requête en direction de la table : dht22_int" << endl
<< ">> " <<  req << endl;
        const char *requete = req.c_str();
        if(mysql1 != NULL)
        {
                //Exécution de la requête
                if(mysql_query(mysql1, requete ))
                {
                //Affichage du code d'erreur si un problème est survenu
                cout << stderr << "%s\n" << mysql_error(mysql1);
                }
         }

         if (sqlite3_open("/home/nichoir/projet_nichoir/dbLocale.db", &db) ==
SQLITE_OK)
         {
             sqlite3_prepare( db, req.c_str(), -1, &stmt, NULL ); //Préparation
de la requête
             sqlite3_step( stmt ); //Exécution de la requête
             cout << endl << "Requête en direction de la table locale :
dht22_int" << endl << ">> " <<  req << endl;
         }
         else{cout << "Impossible d'ouvrir la table\n";}

         sqlite3_finalize(stmt);
         sqlite3_close(db);
    }
}

void Database::sauverDonneesHX711(string dh, float poids)
{
    sqlite3 *db;
    sqlite3_stmt *stmt;

    string req = "INSERT INTO hx711(h_date_heure,h_poids) VALUES ('" + dh +
"'," + to_string(poids) + ");";
    cout << endl << "Requête en direction de la table : hx711 " << endl << ">>
" <<  req << endl;
    const char *requete = req.c_str();
    if(mysql1 != NULL)
```

```
        {
                //Exécution de la requête
                if(mysql_query(mysql1, requete ))
                {
                cout << stderr << "%s\n" << mysql_error(mysql1);
                }
         }
    if (sqlite3_open("/home/nichoir/projet_nichoir/dbLocale.db", &db) ==
SQLITE_OK)
        {
            sqlite3_prepare( db, req.c_str(), -1, &stmt, NULL ); //Préparation
de la requête
            sqlite3_step( stmt ); //Exécution de la requête
            cout << endl << "Requête en direction de la table locale : hx711 "
<< endl << ">> " <<  req << endl;
        }
    else{cout << "Impossible d'ouvrir la table\n";}

    sqlite3_finalize(stmt);
    sqlite3_close(db);
}
```

Python

capture_images.py

```
iimport time
import sys
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
import amg8833_i2c
from database import Database
import os

date_heure = time.strftime('%Y-%m-%d %H:%M:%S')

class CaptureImages:
    def __init__(self):
        self.res_pixels = (8, 8)
        self.xx, self.yy = (np.linspace(0, self.res_pixels[0],
self.res_pixels[0]),
                            np.linspace(0, self.res_pixels[1],
self.res_pixels[1]))
        self.zz = np.zeros(self.res_pixels)
        self.mult_pix = 6
        self.res_interp = (int(self.mult_pix * self.res_pixels[0]),
int(self.mult_pix * self.res_pixels[1]))
        self.grid_x, self.grid_y = (np.linspace(0, self.res_pixels[0],
self.res_interp[0]),
                                    np.linspace(0, self.res_pixels[1],
self.res_interp[1]))
        self.dim_fig = (10, 10)
        self.fig, self.ax = plt.subplots(figsize=self.dim_fig)
        plt.rcParams.update({'font.size': 16})
        self.im1 = None
        self.ax_bgnd = None
        self.nb_pix_lire = 64
        self.db = Database("172.21.28.52") # Modifier l'adresse IP du serveur
de bases de données si besoin
        self.sched = self.db.getSched();

    def initialiserCapteurIR(self):
        # Initialise la caméra infrarouge à l'aide de la bibliothèque
        t0 = time.time()
        while (time.time() - t0) < 1:
            try:
                self.capteur = amg8833_i2c.AMG8833(addr=0x69)
            except:
                self.capteur = amg8833_i2c.AMG8833(addr=0x68)
            finally:
                pass
        time.sleep(0.1)
        if not self.capteur:
            print("Capteur AMG8833 non trouvé - Veuillez vérifier votre
connexion")
            sys.exit()

    def initialiserTraceIR(self):
        # Initialise la figure pour tracer l'image infrarouge
        self.ax.axis('off')  # Désactive les axes x et y
        self.im1 = self.ax.imshow(self.zz, vmin=18, vmax=37,
```

```
cmap=plt.cm.RdBu_r)
        self.fig.tight_layout()
        self.fig.show()

    def interpolerPixels(self, pixels):
        # Interpole les pixels pour avoir une meilleure résolution
        f = interpolate.interp2d(self.xx, self.yy, pixels, kind='cubic')
        return f(self.grid_x, self.grid_y)

    def envoiImageInfrarouge(self):
        # Capture une image avec la caméra infrarouge et l'enregistre dans la
base de données
        status, pixels = self.capteur.read_temp(self.nb_pix_lire)
        new_z = self.interpolerPixels(np.reshape(pixels, self.res_pixels))
        self.im1.set_data(new_z)
        self.fig.canvas.draw()
        self.fig.canvas.flush_events()

        chemin = "/home/nichoir/images/img_infra.jpg"
        plt.savefig(chemin)
        self.db.enregistrer_img("I",date_heure,
chemin,'/home/nichoir/projet_nichoir/dbLocale.db')

    def envoiImageStandard(self):
        # Capture une image avec la caméra standard et l'enregistre dans la
base de données
        chemin = "/home/nichoir/images/img_stand.jpg"
        os.system("libcamera-still -o {}".format(chemin))
        self.db.enregistrer_img("S",date_heure,
chemin,'/home/nichoir/projet_nichoir/dbLocale.db')

    def updateAutonomie(self):
        # Modifie les arrêts et démarrages programmés en fonction de la
configuration souhaitée par l'administrateur
        if self.sched == 1:
            print("Sched = 1")
            f = open("/home/nichoir/wittypi/schedule.wpi", "w")
            f.write("BEGIN 2015-08-01 00:00:00\nEND   2025-07-31 00:00:00\nON
M5\nOFF   M25")
        if self.sched == 2:
            print("Sched = 2")
            f = open("/home/nichoir/wittypi/schedule.wpi", "w")
            f.write("BEGIN 2015-08-01 00:00:00\nEND   2025-07-31 00:00:00\nON
M5\nOFF   M55")
        if self.sched == 3:
            print("Sched = 3")
            f = open("/home/nichoir/wittypi/schedule.wpi", "w")
            f.write("BEGIN 2015-08-01 00:00:00\nEND   2025-07-31 00:00:00\nON
M5\nOFF   M115")
```

database.py

```
import mysql.connector
import sqlite3

class Database:
    def __init__(self, host):
        self.host = host
        self.user = "root"
```

```
        self.password = "tssn2.nichoir!"
        self.database = "nichoir"

    def convertirBinaire(self,file_name):
        with open(file_name, 'rb') as file:
            binary_data = file.read()
        return binary_data

    def enregistrerIMG(self, type, date_heure, chemin_image,db_file):
        # Enregistre les images dans la base de données en fonction de leur
type

        connection = None
        try:
            connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            cursor1 = connection.cursor()

            if type == "I" :
                query = "INSERT INTO images_infra(ii_date_heure, ii_img)
VALUES(%s,%s)"
                cursor1.execute(query,
(date_heure,self.convertirBinaire(chemin_image))) # Enregistrement distant

            if type == "S" :
                query = "INSERT INTO images_stand(is_date_heure, is_img)
VALUES(%s,%s)"
                cursor1.execute(query,
(date_heure,self.convertirBinaire(chemin_image))) # Enregistrement distant

            connection.commit()
            print("Images envoyées à la base de données distante avec succès.")

        except mysql.connector.Error as error:
            print("Erreur lors de l'envoi de l'image à la base de données : ",
error)

        finally:
            if connection is not None and connection.is_connected():
                cursor1.close()
                connection.close()

        try:
            sqliteConnection =
sqlite3.connect('/home/nichoir/projet_nichoir/dbLocale.db')
            cursor2 = sqliteConnection.cursor()

            if type == "I":
                req_sqlite = "INSERT INTO images_infra(ii_date_heure, ii_img)
VALUES(?,?)"
                donnees = (date_heure, self.convertirBinaire(chemin_image))
                cursor2.execute(req_sqlite, donnees) # Enregistrement local
                sqliteConnection.commit()
                print("Enregistrement local image_infra effectué")

            if type == "S":
                req_sqlite = "INSERT INTO images_stand(is_date_heure, is_img)
```

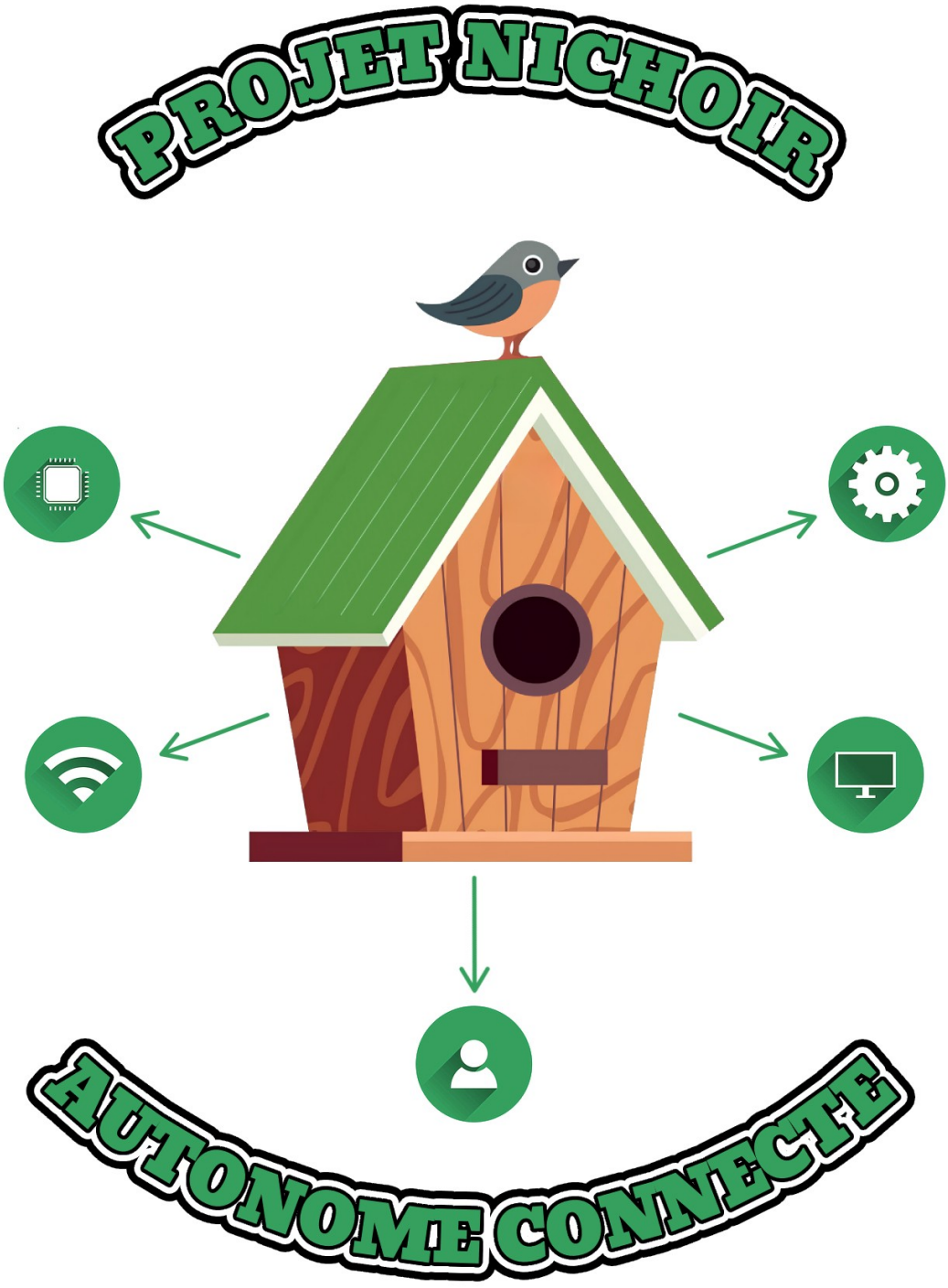
```
VALUES(?,?)"
                donnees = (date_heure, self.convertirBinaire(chemin_image))
                cursor2.execute(req_sqlite, donnees) # Enregistrement local
                sqliteConnection.commit()
                print("Enregistrement local image_stand effectué")

        except sqlite3.Error as er:
            print(er.sqlite_errorcode)
            print(er.sqlite_errorname)

        finally:
            if sqliteConnection:
                cursor2.close()
                sqliteConnection.close()

    def getSched(self):
    # Récupère l'attribut 'sched' du compte administrateur dans la table
'users' de la base de données
        connection = None
        try:
            connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            cursor = connection.cursor()
            query = "SELECT u_sched FROM users WHERE u_username = 'admin'"
            cursor.execute(query)

            result = cursor.fetchone()
            my_value = result[0]
            return my_value

        except mysql.connector.Error as error:
            print("Une erreur est survenue dans l'obtention de l'attribut sched
: ", error)

        finally:
            if connection is not None and connection.is_connected():
                cursor.close()
                connection.close()
```

main.py

```
from capture_images import CaptureImages
import os

if __name__ == "__main__":
    # Initialisation de la caméra infrarouge
    capture_images = CaptureImages()
    capture_images.initialiserCapteurIR()
    capture_images.initialiserTraceIR()

    # Capture et envoi des images
    capture_images.envoiImageStandard()
    capture_images.envoiImageInfrarouge()

    capture_images.updateAutonomie() # Mise à jour des arrêts et démarrages
```

Fournier Alan – Boutte Alexis – Konkin Maksim

Lycée LA FAYETTE Clermont-Ferrand



Guide d’installation

Version de mai 2024

Sommaire

Table des figures

Serveur UwAmp

Connectez vous sur un PC Windows sur le réseau sur lequel se situera le Nichoir Autonome Connecté.

Étape 1. Installez UwAmp en vous rendant sur le site suivant https://www.uwamp.com/fr/ et en suivant la procédure d'installation.

Étape 2. Récupérez le fichier de la base de données et les fichiers du site web à l'aide du lien GitHub suivant : https://github.com/suziksiya/src_nichoir

Étape 3. Lancez UwAmp. Rendez vous dans le dossier www et remplacez les fichiers existants par les fichiers du site web.

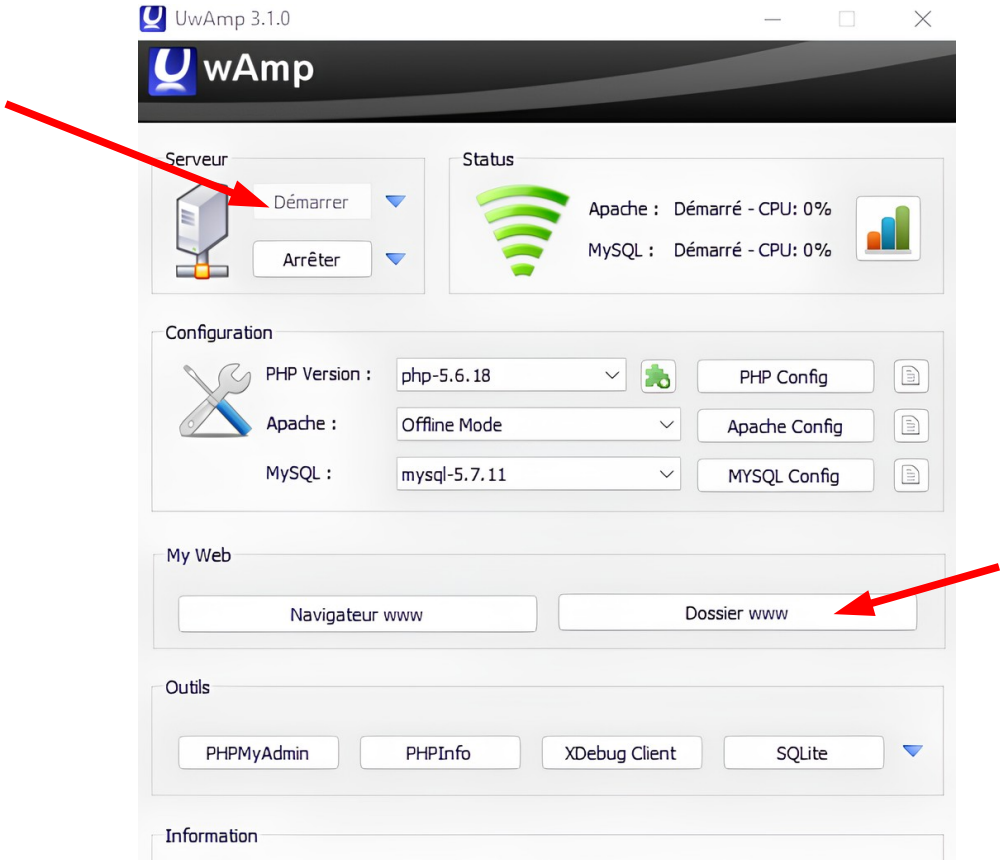


Figure 1: dossier www UwAmp

Étape 4. Démarrez les services Apache et MySQL via la fenêtre UwAmp.

MySQL

Étape 1. Rendez vous dans le dossier d'installation de UwAmp puis dans `\bin\database\mysql-5.7.11\bin`, copiez le chemin relatif.

Étape 2. Ouvrez un invite de commande en tant qu'administrateur et placez vous dans le chemin précédent avec la commande `cd [chemin_relatif]`.

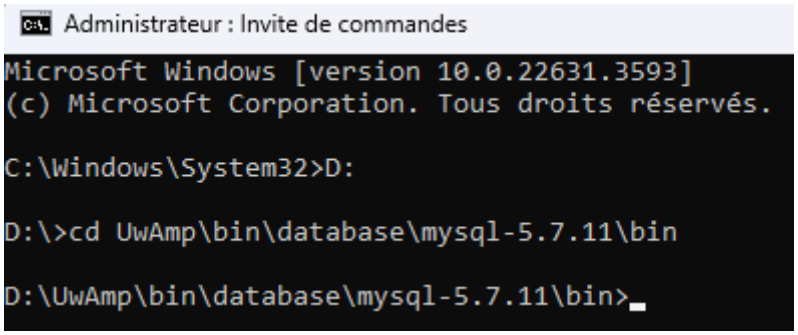


Figure 2: Répertoire MySQL

Étape 3. Connectez vous à MySQL avec la commande `mysql.exe -u root -p`, le mot de passe par défaut est root.

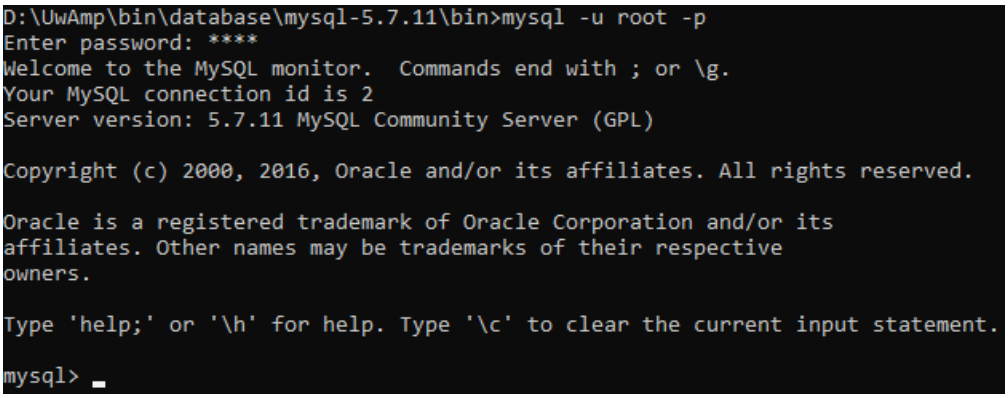


Figure 3: Connexion à MySQL

Étape 4. Donnez tous les droits à l'utilisateur root pour permettre les échanges de données entre le nichoir et la base de données. Exécutez la commande `GRANT ALL PRIVILEGES ON nichoir.* TO 'root'@'[adresse_réseau]' IDENTIFIED BY 'root';`

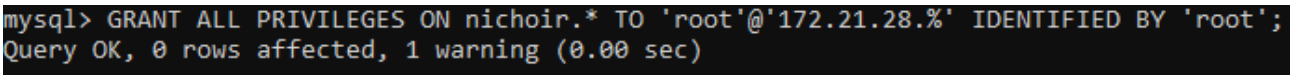


Figure 4: Commande de droits MySQL

Étape 5. Remplacez le mot de passe de l'utilisateur root par un mot de passe plus sécurisé. Il est recommandé de le modifier par 'tssn2.nichoir!'. Si vous souhaitez mettre un mot de passe

différent, des modifications supplémentaires devront être effectuées dans les programmes, cf. partie ‘Paramétrage des programmes’. Veillez à mettre un mot de passe fort. Les commandes à utiliser sont `ALTER USER root@localhost` puis `IDENTIFIED BY ‘[mot_de_passe]’;`.

Étape 6. Sauvegardez les modifications avec la commande `FLUSH PRIVILEGES;`.

Étape 7. Connectez vous avec l’utilisateur root et le mot de passe que vous avez modifié sur l’interface PHPMyAdmin depuis la fenêtre du serveur UwAmp et créez une base de données intitulée « nichoir ».

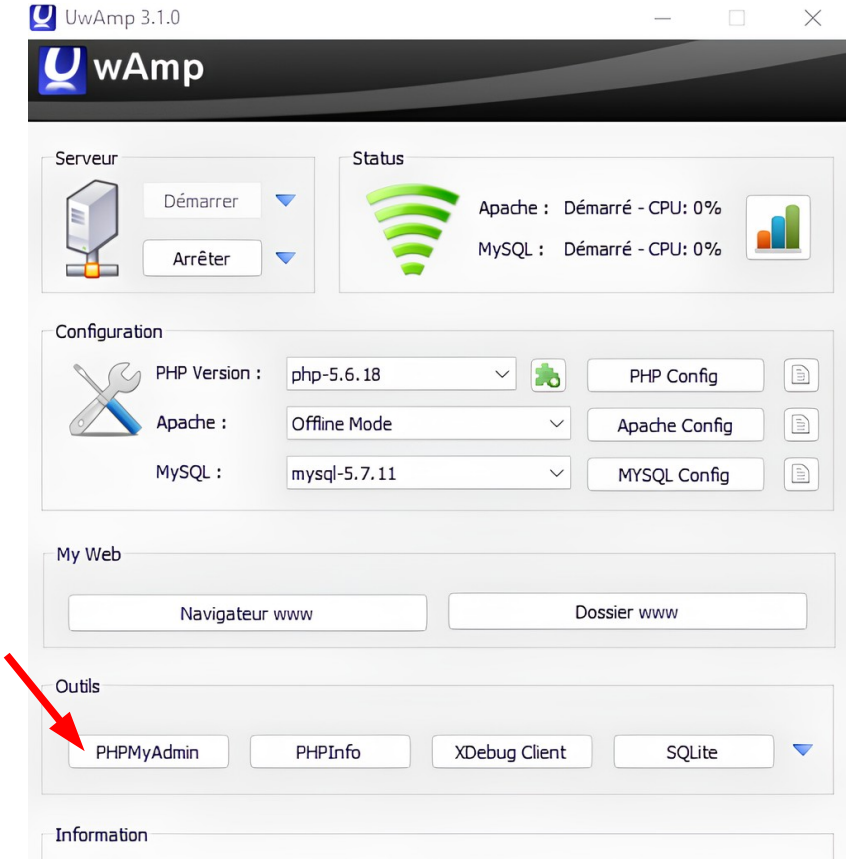


Figure 5: interface PHPMyAdmin

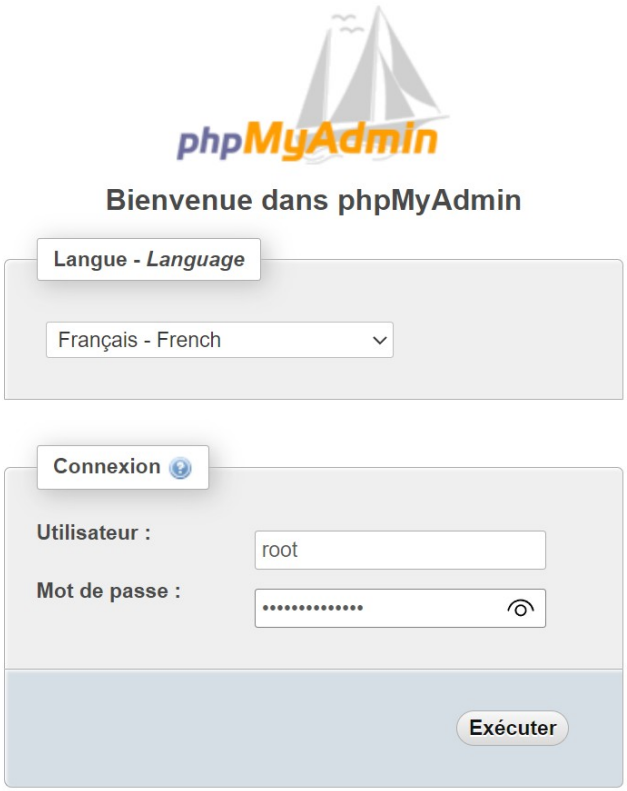


Figure 6: connexion à PHPMyAdmin

Figure 7: création de la base de données nichoir

Étape 8. Copiez le chemin relatif du fichier *nichoir.sql* récupéré dans le dossier *base_de_donnees* sur le dépôt GitHub https://github.com/suziksiya/src_nichoir

Étape 9. Dans la console mysql, tapez la commande *use nichoir*

Étape 10. Entrez ensuite la commande *source chemin/vers_le_fichier/nichoir.sql*, ce qui va importer la structure de la base de données dont vous aurez besoin pour le système. Vous pouvez quitter le terminal mysql avec la commande *exit;*

Étape 11. Par défaut, la table users contient un utilisateur *admin* dont le mot de passe est *admin*. Pour modifier le mot de passe par défaut (recommandé), rendez vous dans l'interface PHPMyAdmin puis sur la base de données *nichoir* et la table *users.*

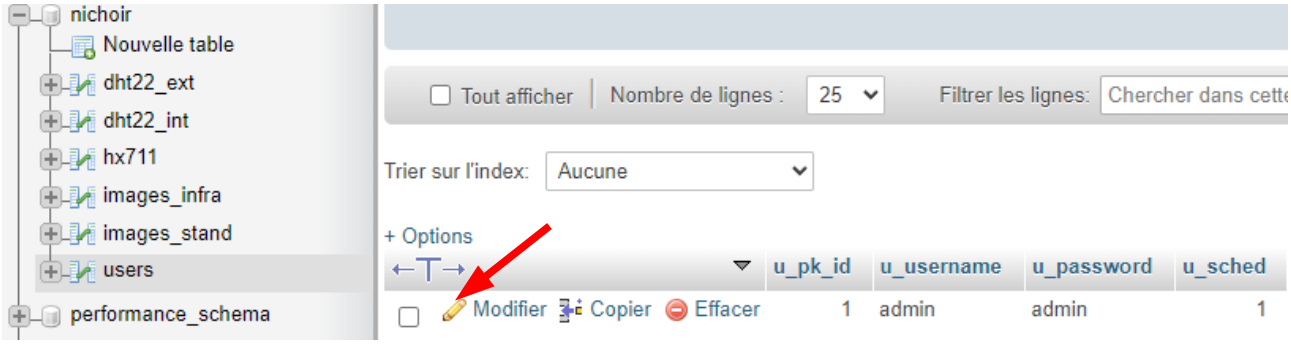


Figure 8: modifier l'utilisateur admin de la table users

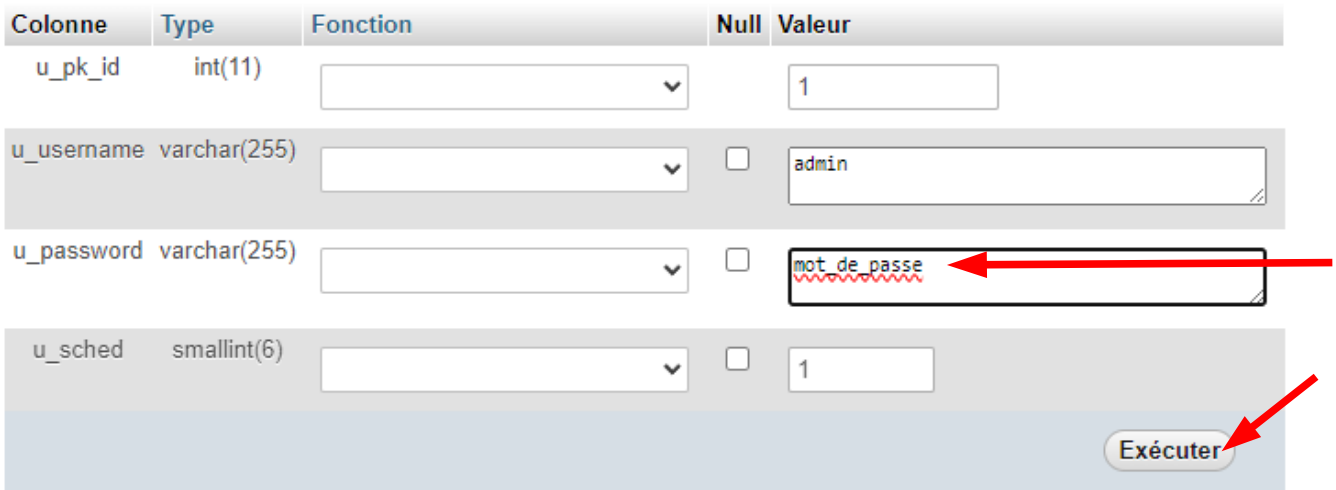


Figure 9: modification du mot de passe de l'utilisateur admin

Étape 12. Pour ajouter un utilisateur standard, rendez vous dans la console SQL en haut de la page PHPMyAdmin et exécutez la commande suivante en modifiant "utilisateur1" par le nom d'utilisateur à ajouter et "mot_de_passe" par son mot de passe. Commande : *INSERT INTO users(u_username, u_password) VALUES("utilisateur1", "mot_de_passe");*

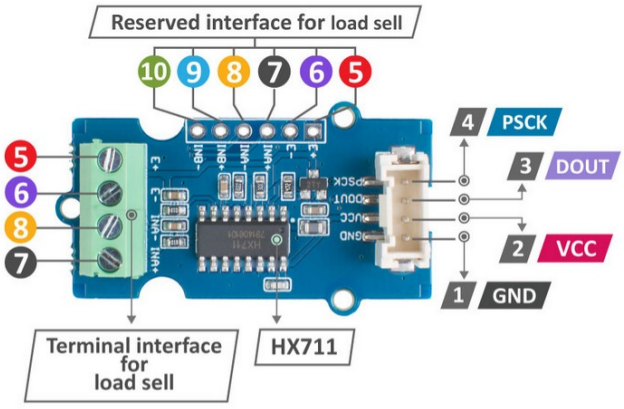
Branchements

Si vous venez de vous procurer un nichoir autonome connecté, vous disposez des éléments suivants : nichoir en bois avec balance installée, carte Raspberry Pi 4, 2 capteurs DHT22, capteur HX711, caméra infrarouge AMG8833, caméra standard Raspicam v2.1, carte Witty Pi 4, module Grove Base Hat, batterie PS1270, Velleman SOL10P, Steca Solsum 6.6c ainsi que les câbles nécessaires aux branchements.

Étape 1. Ajoutez une pile ronde dans l'encoche prévue à cet effet sur la carte Witty Pi 4 afin d'alimenter l'horloge RTC.

Étape 2. Branchez la caméra Raspicam v2.1, puis la carte Witty Pi 4 et le module Grove Base Hat sur la carte Raspberry Pi 4 dans cet ordre.

Étape 3. Branchez le capteur HX711 à la cellule de charge selon les schémas ci-dessous.



1 : Connected to the system GND
2 : Power supply from grove 5V/3.3V*2
3 : Serial data output
4 : Serial clock input
5 : Load sell excitation power supply positive
6 : Load sell excitation power supply negative
7 : Channel A positive input
8 : Channel A negative input
9 : Channel B positive input
10 : Channel B negative input

Amplifier board	Load cell
VCC	-
GND	-
SCK	-
DT	-
E+	Red
E-	Black
A-	Green
A+	White

Voir les liens suivants pour plus d'informations :

https://www.seeedstudio.com/Grove-ADC-for-Load-Cell-HX711-p-4361.html?utm_source=blog&utm_medium=blog

https://www.gotronic.fr/pj2-sen-hx711-manual-2021-10-13-2718.pdf

Étape 4. Suivez le schéma ci-dessous pour le branchement de chacun des capteurs.

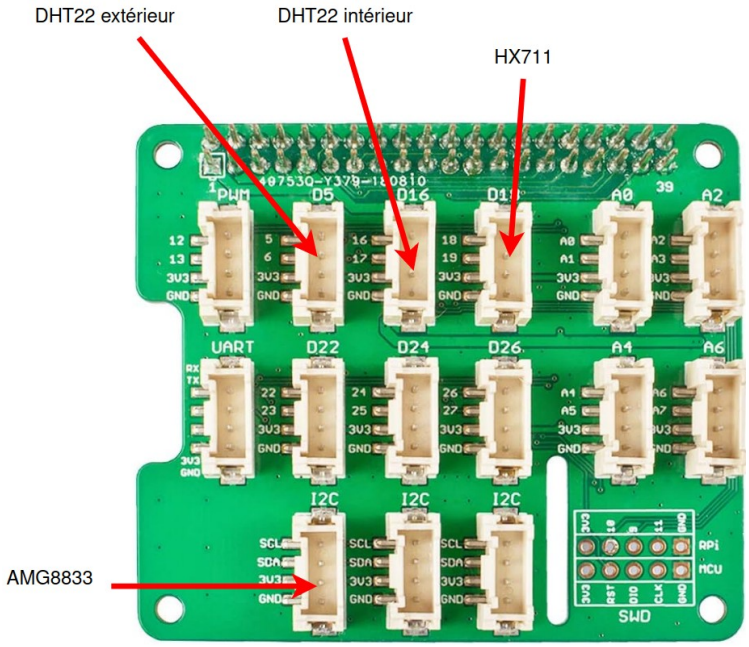


Figure 10: schéma de branchement des capteurs sur le module Grove Base Hat

Vous êtes maintenant libre d'installer le système de la manière que vous souhaitez à l'intérieur du nichoir en bois. De préférence, l'ensemble du système doit être protégé des intempéries pour éviter les dysfonctionnements.

Paramétrage de la carte Raspberry Pi 4

Connectez un écran, une souris et un clavier à votre carte Raspberry Pi 4. Branchez le câble d'alimentation sur la carte Witty Pi 4 et appuyez sur le bouton d'allumage de la carte. Une fois sur le bureau de la carte Raspberry, connectez-la au Wi-Fi et utilisez les touches `Ctrl + Alt + T` pour lancer un terminal.

Étape 1. Mettez à jour la carte avec la commande `sudo apt update && sudo apt upgrade -y`.

Étape 2. Exécutez la commande `sudo raspi-config`. Dans le menu qui s'affiche, rendez vous dans la section 3, `Interface Options`. Dans cette section, activez les protocoles `I2C` et `Serial Port`.

Étape 3. Installez le pare-feu UFW avec la commande `sudo apt install ufw`. Activez-le avec la commande `sudo ufw enable`.

Installation des programmes de gestion des capteurs

Étape 1. Rendez vous dans le répertoire racine en exécutant la commande `cd`. Créez un dossier `projet_nichoir` avec la commande `sudo mkdir projet_nichoir`. Rendez vous dans ce nouveau répertoire avec la commande `cd projet_nichoir`.

Étape 2. Téléchargez l'ensemble des programmes avec la commande `wget https://github.com/suziksiya/nichoir`. Vous disposez maintenant de tous les répertoires et programmes nécessaires pour faire fonctionner les capteurs.

Installation des bibliothèques

Certaines bibliothèques sont nécessaires au fonctionnement des programmes. Placez vous dans le répertoire racine à l'aide de la commande `cd`, puis entrez les commandes ci-dessous à la suite.

- Connecteur MySQL/C++ : `sudo apt install libmysql++-dev`
- WiringPi :
  - `cd /tmp`

    - *wget https://project-downloads.drogon.net/wiringpi-latest.deb*
    - *sudo dpkg -i wiringpi-latest.deb*
    - *cd ..*
- Caméra extérieure :
    - *wget -O install_pivariety_pkgs.sh https://github.com/ArduCAM/Arducam-Pivariety-V4L2-Driver/releases/download/install_script/install_pivariety_pkgs.sh*
    - *chmod +x install_pivariety_pkgs.sh*
    - *./install_pivariety_pkgs.sh -p libcamera_dev*
    - *./install_pivariety_pkgs.sh -p libcamera_apps*
- Python 3 : *sudo apt-get install python3*
- Modules Python :
    - *pip3 install netifaces*
    - *pip3 install scipy*
    - *pip3 install matplotlib*
    - *pip3 install numpy*
    - *pip3 install mysql-connector-python*
- Witty Pi 4 :
    - *wget https://www.uugear.com/repo/WittyPi4/install.sh*
    - *sudo sh install.sh*
- SQLite :
    - *sudo apt install sqlite3*

Paramétrage des programmes

Des modifications doivent être effectuées dans les programmes du Nichoir Autonome Connecté selon votre installation.

Changement d’IP du serveur

Tout d’abord, récupérez l’adresse IPv4 de votre serveur de base de données en vous rendant dans un invite de commande et en exécutant la commande *ipconfig*.

Étape 1. Rendez vous dans le dossier */projet_nichoir/cpp* avec la commande *cd /projet_nichoir/cpp*. Par la suite, éditez le fichier *mesures.cpp avec la commande sudo nano mesures.cpp*.

Étape 2. Remplacez l'adresse IP indiquée dans le programme par celle de votre serveur en veillant à conserver les guillemets. Enfin, sauvegardez le fichier en appuyant sur *Ctrl + X* puis *O* et *Entrer*.

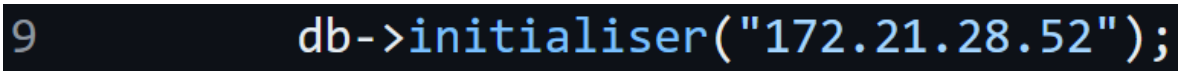


Figure 11: Modification IP serveur, fichier mesures.cpp

Étape 3. Rendez vous dans le dossier */projet_nichoir/python* avec la commande *cd ../python*. Par la suite, éditez le fichier *capture_images.py avec la commande sudo nano capture_images.py*.

Étape 4. Remplacez l'adresse IP indiquée dans le programme par celle de votre serveur en veillant à conserver les guillemets. Enfin, sauvegardez le fichier en appuyant sur *Ctrl + X* puis *O* et *Entrer*.

Figure 12: Modification IP serveur, fichier capture_images.py

Changement d'identifiants de la base de données

Si durant l'installation du serveur de base de données vous avez fait le choix de changer les identifiants de connexion, voici les étapes que vous devez suivre pour que le programme fasse le lien.

Étape 1. Rendez vous dans le dossier */projet_nichoir/cpp* avec la commande *cd /projet_nichoir/cpp*. Par la suite, éditez le fichier *database.h avec la commande sudo nano database.h*.

Étape 2. Remplacez les identifiants indiqués dans le programme par ceux de votre serveur en veillant à conserver les guillemets. Enfin, sauvegardez le fichier en appuyant sur *Ctrl + X* puis *O* et *Entrer*.

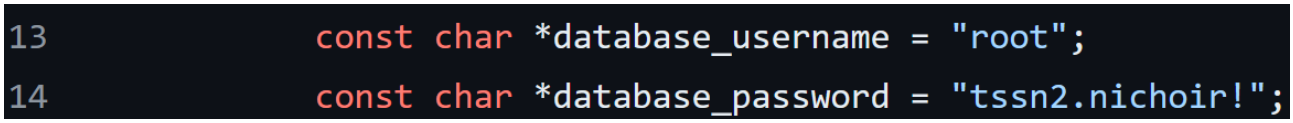


Figure 13: Modification identifiants serveur, fichier database.h

Étape 3. Rendez vous dans le dossier `/projet_nichoir/python` avec la commande `cd /projet_nichoir/cpp`. Par la suite, éditez le fichier `database.py` *avec la commande* `sudo nano database.py`.

Étape 4. Remplacez les identifiants indiqués dans le programme par ceux de votre serveur en veillant à conserver les guillemets. Enfin, sauvegardez le fichier en appuyant sur `Ctrl + X` puis `O` et `Entrer`.

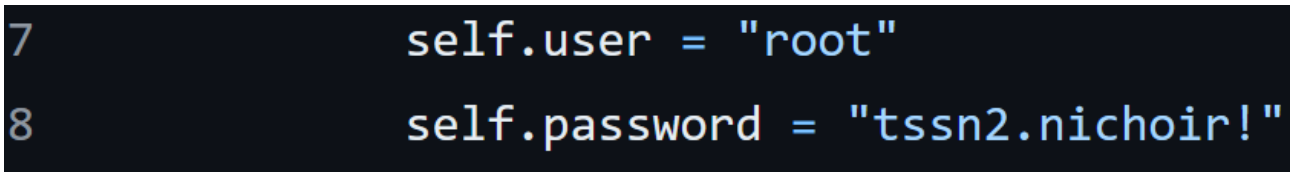


Figure 14: Modification identifiants serveur, fichier database.py

Compilation

Une fois toutes les modifications enregistrées, veuillez compiler le programme en C++ pour prendre en compte les changements dans le fichier `make`.

Étape 1. Rendez vous dans le dossier `/projet_nichoir/cpp` avec la commande `cd /projet_nichoir/cpp`.

Étape 2. Lancez la compilation avec la commande `g++ -Wall *.cpp -o make -lwiringPi -lmysqlclient -lhx711 -llgpio -lsqlite3.` La console devrait signaler un warning, ce qui est normal, la compilation s'est quand même déroulée sans souci.

Automatisation

Afin que les programmes soient exécutés lorsque le système s'active, il est nécessaire de mettre en place un crontab.

Étape 1. Lancez l'exécution du crontab avec la commande *`crontab -u nichoir -e`*. Choisissez nano comme éditeur de texte.

Étape 2. Coller le texte suivant à l'intérieur du fichier crontab :

```
@reboot sleep 60;/home/nichoir/projet_nichoir/cpp/./make
@reboot sleep 60;/usr/bin/python3 /home/nichoir/projet_nichoir/python/main.py
@reboot sleep 60;sudo timedatectl set-ntp true
```

Site web

Étape 1. Après avoir récupéré les fichiers du site web sur le dépôt GitHub cité sur la première page du guide et les avoir déposés dans le dossier www du serveur UwAmp, éditez le fichier *`config.php`* du dossier *`secrets`* avec un éditeur de texte.

Étape 2. Remplacez l'adresse IP de la première ligne par celle de votre serveur de base de données et les identifiants en fonction de ceux que vous souhaitez utiliser pour vous connecter à la base de données et enregistrez le fichier.

```php
<?php
define('SERVEUR', '172.21.28.52');
define('UTILISATEUR', 'root');
define('MOT_DE_PASSE', 'tssn2.nichoir!');
define('BASE_DE_DONNEES', 'nichoir');
```

Figure 15: Modification identifiants serveur site web, fichier config.php

Application de consultation

Vous pouvez choisir le poste sur lequel installer l’application de consultation. Cependant, il faut nécessairement que le poste ait une installation Windows.

Étape 1. Récupérez le dossier de l’application à l’aide du lien GitHub suivant : https://github.com/suziksiya/src_nichoir

Étape 2. Déposez le dossier où vous le souhaitez sur votre machine et lancez l'exécutable de l'application.

Étape 3. Dans la fenêtre qui s’ouvre, vous avez la possibilité de modifier les paramètres du serveur de base de données pour renseigner les vôtres.