



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Connectique et accès aux données du pavillon DEMO MI2

Rapport
PROJET DE SEMESTRE 6

V.1.0

Haute école d'ingénierie et d'architecture de Fribourg
Filière informatique
21 mai 2021

Etudiant
Denis Rosset
Julien Piguet

Superviseurs
Jacques Robadey
Nicolas Schroeter

Versions du document

N° de révision	Date	Description
0.1	16 mars 2021	Première version
0.2	14 mai 2021	Première de l'analyse
0.2.1	15 mai 2021	Ajout de la comparaison des Hats LoRa dans l'analyse
0.3	15 mai 2021	Première de la spécification et de la conception. Ébauche de la réalisation
0.3.1	16 mai 2021	Ajout de la conception du serveur web et finalisation de la partie conception
0.4	17 mai 2021	Ajout de la partie LoRa et Application Web dans la réalisation. Création d'une section mise en place
0.5	18 mai 2021	Complétion de la partie mise en place
0.5.1	18 mai 2021	Écriture d'une première version du protocole de test
0.5.2	18 mai 2021	Correction et complétion de la partie réalisation
0.6	19 mai 2021	Création d'une section de mode d'emploi
0.6.1	19 mai 2021	Complétion des tests
0.7	20 mai 2021	Écriture de l'introduction et ébauche de la conclusion
0.7.1	20 mai 2021	Finalisation des parties spécification, conception et réalisation
0.7.2	20 mai 2021	Finalisation de la partie mise en place et mode d'emploi
0.8	21 mai 2021	Finalisation de la partie conclusion
0.8.1	21 mai 2021	Correction des erreurs et mise en page
1.0	21 mai 2021	Version final

Table des matières

1	Introduction	1
1.1	Acteurs	1
1.2	Contexte et but du projet	1
1.3	Situation avant le projet	2
1.4	Objectifs principaux du projet	2
1.4.1	Récolte de données de différents capteurs par un système embarqué autonome	2
1.4.2	Choix et validation de fonctionnement des capteurs	2
1.4.3	Envoie des données sur le réseau LoRa de la Ville de Fribourg	2
1.4.4	Réception et affichage des données sur une application Web	2
1.5	Objectifs secondaires du projet	3
1.5.1	Localisation du système par GPS	3
1.5.2	Retour visuel du fonctionnement des capteurs sur le système embarqué	3
1.6	Méthode de travail	3
1.6.1	Cahier des charges	3
1.6.2	Planning	4
2	Analyse	5
2.1	Protocoles	5
2.1.1	I2C	5
2.1.2	Transmission Série RS-485	5
2.2	Capteurs	6
2.2.1	SHT31-D	6
2.2.2	MCP9808	6
2.2.3	BMP280	7
2.2.4	SWEMA-03	7
2.2.5	Pyranomètre Kipp&Zonen CM3	8
2.2.6	ADS1115	8
2.3	Raspberry Pi	9
2.3.1	Alimentation	9
2.4	LoRa	10
2.4.1	Technologie LoRa	10
2.4.2	LoRaWAN	10
2.4.3	Architecture LoRa	11
2.4.4	Raspberry Pi Hats	13
3	Spécification	16
3.1	Organisation physique	16
3.2	Connectique du Raspberry Pi	17
3.2.1	Port USB	17
3.2.2	GPIOs	18
3.3	Maquettes Web	19

4 Conception	21
4.1 Application du système embarqué	21
4.1.1 Diagramme d'activité	21
4.1.2 Prise des mesures	21
4.1.3 Sauvegarde des mesures en local	22
4.1.4 Envoi des données via LoRa	22
4.1.5 Accès ssh	22
4.1.6 Fichier de configuration	22
4.1.7 Création de logs (journalisation)	23
4.2 LoRa	23
4.2.1 Format des données	23
4.2.2 Trame	24
4.2.3 Network Server	25
4.3 Application Web	25
4.3.1 Serveur	25
4.3.2 BBData	26
4.3.3 Routes	27
4.3.4 Client	29
5 Réalisation	30
5.1 Environnement de développement et logiciels utilisés	30
5.2 Implémentation de l'application du système embarqué	30
5.2.1 Librairies	30
5.2.2 Composants	31
5.2.3 Gestion des logs (journalisation)	32
5.2.4 Gestion de l'ordonnanceur	32
5.2.5 Mesures des données (méthode measure_data())	32
5.2.6 Sauvegarde des valeurs mesurées en local (méthode save_data())	33
5.2.7 Envoi des données sur LoRa (méthode send_to_lora())	33
5.3 LoRa et Chirpstack server	33
5.3.1 Chirpstack	33
5.3.2 BBData	38
5.4 Application et serveur web	39
5.4.1 Serveur Web	39
5.4.2 Application Web	39
5.4.3 Resultats	41
6 Mise en place	45
6.1 Raspberry pi	45
6.1.1 Activer I2C	45
6.1.2 Installation de l'application	46
6.1.3 Crontab	46
6.2 LoRa - Raspberry Pi	46
6.2.1 IoT LoRa Node pHat	46
6.2.2 Lora/GPS HAT	48
6.3 Installation du serveur	50
6.4 Montage	53

6.4.1	Montage sur Bread Board	53
6.4.2	Montage Final	55
7	Mode d'emploi	57
7.1	Mode d'emploi du fonctionnement de l'application du Raspberry Pi	57
7.1.1	Connexion au système embarqué à distance	57
7.1.2	Configuration du système embarqué	57
7.1.3	Démarrage de l'application du système embarqué	58
7.2	Accès au serveur web	59
8	Tests et validation	60
8.1	Tests fonctionnels	60
9	Améliorations	62
9.1	Amélioration du système embarqué	62
9.1.1	Écran d'affichage de l'état du système embarqué	62
9.1.2	Gestion des erreurs	62
9.1.3	Pyranomètre	62
9.2	Amélioration de l'application web	63
9.3	Amélioration de l'envoi de données LoRa	63
9.3.1	Séparation des mesures	63
9.3.2	Données de test	63
10	Conclusion	64
10.1	Atteinte des objectifs	64
10.1.1	Objectifs principaux	64
10.1.2	Objectifs secondaires	64
10.2	Suivi du planning	64
10.3	Déroulement global du projet	64
10.4	Collaboration entre les étudiants	65
10.5	Conclusions personnelles	65
10.5.1	Conclusion de M. Rosset	65
10.5.2	Conclusion de M. Piguet	65
10.5.3	Conclusion commune	65
11	Déclaration d'honneur	66
12	Annexes	67
13	Glossaire	68
14	Références	69
15	Figures	70

1 Introduction

Ce document décrit et détail le déroulement du projet de semestre 6 intitulé "Connectique et accès aux données du pavillon DEMO MI2". Son but est détaillé les choix qui ont été pris ainsi que les résultats qui ont été obtenus.

1.1 Acteurs

Ce projet est suivi par les personnes suivantes :

- Jacques Robadey, superviseur
- Nicolas Schroeter, superviseur
- Raphael Compagnon, mandant
- Denis Rosset, étudiant
- Julien Piguet, étudiant

1.2 Contexte et but du projet

Le Développement durable de la Ville de Fribourg et la HEIA-FR ont lancé le projet d'un pavillon de démonstration du rafraîchissement naturel d'un espace public extérieur pour les périodes de canicule. Au mois de juin 2021, le pavillon, nommé DEMO-MI2, sera monté pour la première fois au bout du Boulevard de Pérrolles. Ce pavillon mobile innovant sera un îlot de fraîcheur pour toute la saison estivale et se déplacera dans plusieurs zones publiques de la ville.

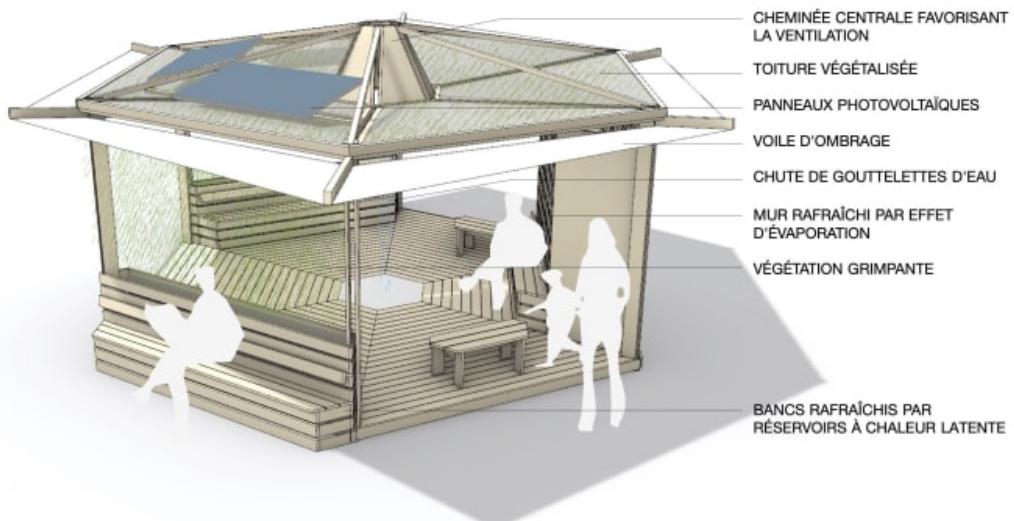


FIGURE 1 – Esquisse du pavillon DEMO-MI2

Le but du projet est de créer un système de mesures de données météo à différents points du pavillon DEMO-MI2 pour pouvoir étudier les différences météorologiques entre l'intérieur et l'extérieur du pavillon. Le système doit être déplaçable facilement et les données doivent être lisibles depuis une application web ainsi que stockées sur le système embarqué. Deux

systèmes semblables tels que décrits précédemment doivent être développés pour qu'ils soient utilisés en parallèle afin de pouvoir comparer les mesures entre les deux systèmes.

1.3 Situation avant le projet

Le pavillon est conçu en parallèle du projet et le système embarqué

1.4 Objectifs principaux du projet

Les objectifs principaux du projet sont ceux qu'il est nécessaire de terminer avant que le projet n'arrive à terme.

1.4.1 Récolte de données de différents capteurs par un système embarqué autonome

Un système embarqué doit être développé et mis en place afin qu'il permette de récolter et traiter des données provenant de différents capteurs. Les capteurs sont spécifiques à la récolte d'informations météo. Le système embarqué autonome doit pouvoir fonctionner une demi-journée au moyen de batteries et panneaux photovoltaïques. Le code source doit être en Python. La période d'échantillonnage doit être comprise entre 1 et 5 secondes. Un interrupteur permet de définir le début et la fin de la récolte des mesures. Modification possible des paramètres du système une fois déployé.

1.4.2 Choix et validation de fonctionnement des capteurs

L'interfaçage entre le système embarqué et chaque capteur doit être définie et implémentée. Les capteurs proposés doivent être validés et, le cas échéant, de nouveaux capteurs doivent être choisis et utilisés. Les capteurs doivent être en mesure d'effectuer les mesures suivantes :

- La température de l'air
- La température radiante
- L'humidité de l'air
- La vitesse de l'air
- La puissance du rayonnement solaire

Les mesures des capteurs doivent être précises afin d'avoir des résultats significatifs.

1.4.3 Envoie des données sur le réseau LoRa de la Ville de Fribourg

Le système embarqué doit permettre l'envoi des données sur le réseau LoRa de la Ville de Fribourg de façon périodique. La quantité et la fréquence sont déterminées par la capacité du réseau ainsi que par les besoins du projet.

1.4.4 Réception et affichage des données sur une application Web

Les données provenant du système embarqué sont réceptionnées au travers du réseau LoRa. Elles sont ensuite traitées et affichées sur l'application Web. La température ressentie est calculée et affichée par l'application web.

1.5 Objectifs secondaires du projet

Les objectifs secondaires du projet sont réalisables si le projet n'est pas terminé, mais que les objectifs principaux ont déjà été réalisés.

1.5.1 Localisation du système par GPS

La position GPS du système est localisée grâce à une carte GPS et est envoyée sur le réseau LoRa en même temps que les données des autres capteurs.

1.5.2 Retour visuel du fonctionnement des capteurs sur le système embarqué

Une information visuelle doit permettre d'indiquer si chaque capteur récolte ses données et les transmet correctement.

1.6 Méthode de travail

Afin de réaliser ce projet, un suivi constant est fait par les superviseurs et le mandant grâce à des séances de projet qui ont lieu chaque semaine et auxquelles les superviseurs, le mandant et les étudiants participent. Un procès-verbal a été rédigé pour chaque séance¹.

1.6.1 Cahier des charges

Un cahier des charges a été écrit au début du projet pour clarifier les besoins et objectifs relatifs au projet. Ce cahier a permis de connaître chaque but et contrainte nécessaires à la réalisation de ce projet. Une comparaison entre les objectifs du cahier des charges et les objectifs atteints dans le projet est disponible dans le chapitre 10.1.1.

1. <https://gitlab.forge.hefr.ch/denis.rosset/ps6-pavillon-demo-mi2/-/tree/master/docs/PVs>

1.6.2 Planning

Une planification du projet a été réalisée en même temps que le cahier des charges (2).

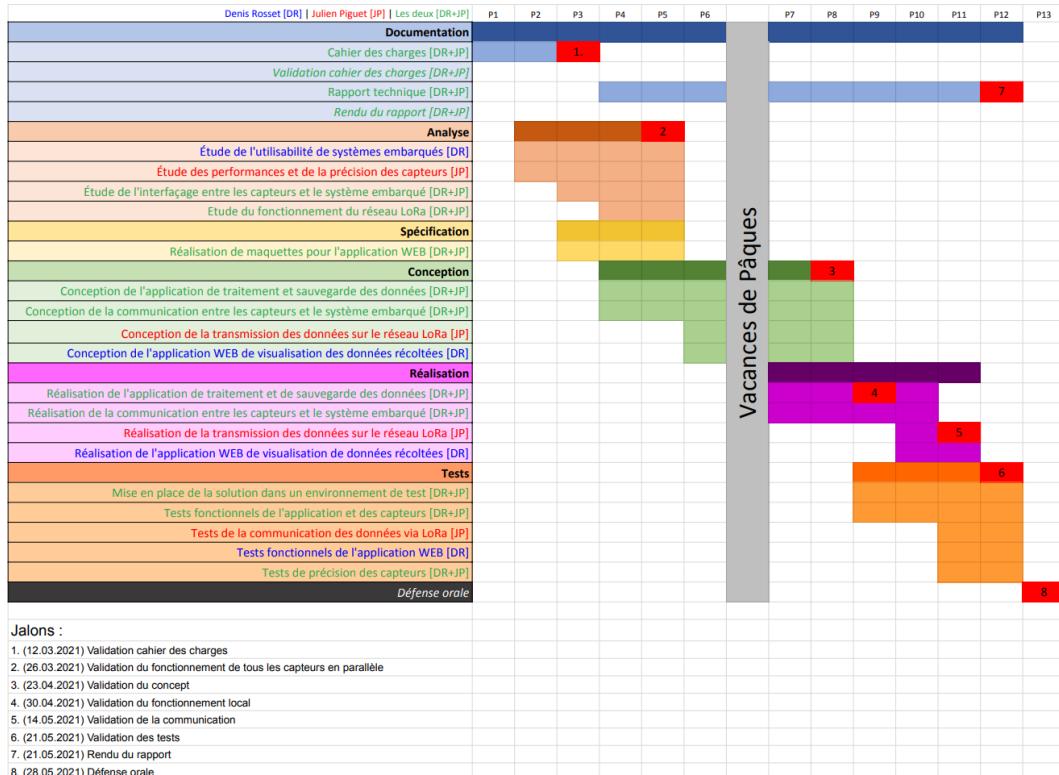


FIGURE 2 – Planning du projet

Une rétrospective sur le suivi du planning est disponible dans le chapitre 8

2 Analyse

Dans cette section, une analyse des besoins du projet a été réalisée. Les composants électroniques de mesures de données météorologiques ont été choisi par le mandant du projet pour des raisons de précision des mesures. L'analyse de ces composant a permis de prouver qu'ils peuvent communiquer avec le système embarqué. Pour prouver ce point, une analyse des protocoles utilisés par les capteurs a aussi été réalisée. Pour finir une analyse de systèmes embarqués permettant la communication grâce aux protocoles des capteurs a été réalisée. Une analyse de la communication LoRa a aussi été réalisée.

2.1 Protocoles

2.1.1 I2C

Le protocole I2C (qui signifie Inter-Integrated Circuit) est un bus informatique qui permet à plusieurs systèmes de communiquer entre eux. La connexion se fait grâce à deux lignes physiques :

SDA Serial Data Line : ligne de données bidirectionnelle

SCL Serial Clock Line : ligne d'horloge de synchronisation bidirectionnelle La liaison se fait grâce à un principe maître-esclave qui peut s'inverser. Ce système permet d'envoyer des commandes en mode master puis de se mettre en mode slave pour recevoir la réponse à ces commandes. Ce système permet un adressage sur 7 bit ce qui permet d'avoir jusqu'à 128 microprocesseurs connectés sur le même bus.

Ce système nous permet de communiquer entre le système embarqué et tous les capteurs utilisant ce protocole sur le même bus afin de récupérer les valeurs des différents capteurs. La distance maximale des communications via le bus dépend de la capacitance du câble. Avec un câble blindé, une communication jusqu'à 10 mètres est envisageable. Dans le projet, le bus I2C est utilisé pour communiquer avec les composants suivants :

- **SHT31-D** Capteur de température et d'humidité
- **MCP9808** Capteur de température
- **BMP280** Capteur de pression
- **ADS1115** Convertisseur analogique vers numérique

2.1.2 Transmission Série RS-485

La transmission de données en série est une modalité de transmission où les données se suivent les unes après les autres sur une seule voie. La norme RS-485 définit les caractéristiques de la couche physique de l'interface. Elle permet l'envoi de données au travers d'une paire torsadée grâce à des variations de tension en mode différentiel. Elle est utilisée dans le projet pour communiquer avec le capteur SWEMA-03.

2.2 Capteurs

Différents capteurs sont utilisés afin de pouvoir fournir les informations sur la température de l'air, la température globe, la vitesse de l'air, l'humidité relative de l'air et le rayonnement solaire horizontal.

2.2.1 SHT31-D

Le capteur SHT31-D est un capteur de température et d'humidité relative créée par Adafruit. Il a une précision de $\pm 2\%$ pour l'humidité et de $\pm 0.3^\circ\text{C}$ pour la température. Il fonctionne avec le bus I2C. Le capteur doit alimenter en 3V. Le fournisseur propose une librairie python afin de pouvoir utiliser le capteur en I2C.

Les valeurs retournées par le capteur sont la température de l'air en degré Celsius et l'humidité relative de l'air en pour cent. Le capteur possède également une fonction de chauffe afin d'enlever le surplus d'humidité sur le capteur d'avoir une valeur plus précise.

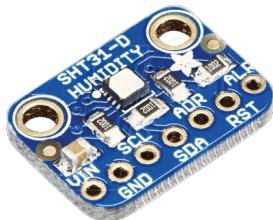


FIGURE 3 – SHT31-D (source : <https://www.adafruit.com/product/2857>)

2.2.2 MCP9808

Le capteur MCP9808 est un capteur de température créée par Adafruit. Il a une précision typique de $\pm 0,25^\circ\text{C}$ sur la plage du capteur de -40°C à $+125^\circ\text{C}$ et une précision de $\pm 0,0625^\circ\text{C}$. Il fonctionne avec le bus I2C. Le capteur fonctionne en 3V et 5V.

Les valeurs retournées par le capteur sont la température en degré Celsius. Il est utilisé afin d'obtenir la température. Cette température sera mesurée à l'intérieur d'un globe afin de pouvoir avoir une température qui est moins sensible aux faibles variations.



FIGURE 4 – MCP9808 (source : <https://www.adafruit.com/product/1782>)

2.2.3 BMP280

Le capteur BMP280 est un capteur de pression barométrique, et de température créée par Adafruit. Il a une précision absolue de ± 1 hPa sur la plage du capteur de 300 hPa à 1100 hPa. Il fonctionne avec le bus I2C. Le capteur fonctionne en 3V et 5V.

La valeur nécessaire à ce projet est celle de la pression et elle est retournée par le capteur en hectopascal. La valeur est utilisée pour calibrer l'anémomètre afin que les valeurs de la vitesse du vent soient précises au maximum.



FIGURE 5 – BMP280 (source : <https://www.adafruit.com/product/2651>)

2.2.4 SWEMA-03

Le capteur SWEMA-03 est un anémomètre omnidirectionnel conçu spécialement pour les faibles vitesses d'air. Il mesure des valeurs de vitesse d'air entre 0.05m/s et 3m/s avec une précision de trois chiffres significatifs. Il est créé par SWEMA AB et est connecté avec un câble USB via lequel est utilisé le protocole RS485.

Les valeurs retournées par le capteur sont en mètres par seconde et le capteur est utilisé pour mesurer les différences de déplacement de l'air entre l'intérieur et l'extérieur du pavillon.



FIGURE 6 – BMP280 (source : <https://thermolab.ch/produit/swema-03-anemometre-omnidirectionnel-vitesse-air/>)

2.2.5 Pyranomètre Kipp&Zonen CM3

Le CM3 est un pyranomètre robuste fabriqué par Kipp & Zonen. Il mesure le rayonnement solaire grâce à une thermopile noircie protégée par un dôme. La thermopile noircie fournit un voltage entre 0 et 50 mV qui correspond aux valeurs entre 0 et 2000 W/m^2 .

La valeur du voltage doit être mesurée grâce à la différence entre le voltage du câble de contrôle et celui de mesure. Cette valeur doit ensuite être divisée par une valeur propre à chaque pyranomètre afin d'obtenir la valeur du rayonnement solaire en W/m^2 .



FIGURE 7 – BMP280 (source : <https://www.campbellsci.com/cm3>)

2.2.6 ADS1115

Le capteur ADS1115 est une carte électronique de conversion analogique vers numérique. Elle possède une précision de 16 bits et permet de mesurer quatre canaux différents. Elle est branchée aux câbles du pyranomètre afin de mesurer les valeurs analogiques qui correspondent aux valeurs de rayonnement solaire et de les convertir en valeurs numériques. Les valeurs renvoyées par le capteur sont les voltages des différents canaux. Il faut ensuite effectuer une différence entre les deux mesures de voltage pour obtenir la valeur mesurée par le pyranomètre.

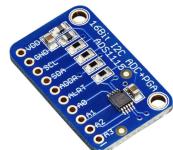


FIGURE 8 – MCP9808 (source : <https://www.adafruit.com/product/1085>)

2.3 Raspberry Pi

Les Raspberry-Pi sont des *nano single-board computer* (nano-ordinateur monocarte) qui font la taille d'une carte de crédit. Ils possèdent une interface I2C ainsi que série pour pouvoir communiquer avec les capteurs décrits dans le point 2.2. Grâce à un système d'exploitation Linux, ils permettent de développer rapidement des applications complexes qui nécessitent d'être exécutés sur système embarqué.

Dans le cadre du projet, deux Rapsberry Pi différents ont été proposés. Il s'agit du Raspberry Pi 3B+ ainsi que du Raspberry Pi 4. Le but du projet étant de pouvoir faire fonctionner les systèmes embarqués sur batterie le choix s'est porté sur le Raspberry Pi 3B+ car il consomme moins d'énergie que le Raspberry pi 4 [6] mais reste suffisamment puissant pour effectuer toutes les opérations nécessaires à la réalisation du projet.



FIGURE 9 – Raspberry Pi 3b+ (source : <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>)

2.3.1 Alimentation

Les batteries Baseus Power Bank 65 watt ont une charge de 30'000 miliampères-heures. Grâce au calculateur Raspberry Pi Battery Runtime Calculator [5], Nous pouvons affirmer que la batterie sera suffisante pour les besoins en énergie du projet. La consommation des capteurs se calcule en nanoampère. Elle est donc négligeable par rapport à celle du Raspberry pi.



FIGURE 10 – Baseus Bank (source : <http://www.baseus.com/product-683.html>)

2.4 LoRa

LoRa est l'acronyme de Long Rang (longue portée). C'est une technologie de modulation par étalement de spectre. Le protocole utilisé pour exploiter LoRa est le LoRaWAN. Cette technologie permet de transmettre des données sur de longues distances. Il est principalement utilisé dans les villes intelligentes, le monitoring industriel ou l'agriculture. La ville de Fribourg possède un réseau LoRa intitulé FriLoRaNet.

2.4.1 Technologie LoRa

LoRa est une technologie de modulation à étalement de spectre de type Chirp spread spectrum (CSS). Le CSS est une technique d'étalement du spectre qui utilise des impulsions chirp, signal sinusoïdal avec une fréquence qui augmente ou diminue au fil du temps, à large bande pour coder les informations. Les bandes de fréquences sont 868 MHz en Europe et 915 MHz en Amérique du Nord. Cette modulation permet d'avoir en moyenne une distance pour la communication entre une passerelle et un équipement jusqu'à 5 km en zone urbaine et 15 km en zone rurale.

2.4.2 LoRaWAN

LoRaWAN est un protocole de communication qui est basé sur la technologie LoRa. LoRaWAN est l'acronyme de "Long Rang Wide Area Network". Le protocole a pour but d'être simple et peu coûteux en énergie grâce à un débit peu élevé et à la technologie LoRa.

Authentification

Deux modes d'authentification sont disponibles, le OTAA et le ABP.

- **ABP** (Activation By Personalization) : Les clés de chiffrement sont générées par le network server et stocké à la main dans le Node.
- **OTAA** (Over The Air) : Les clés de chiffrement sont obtenues automatiquement à la connexion. Seul l'AppKey doit être renseigné.

Dans les deux cas, ce sont des EUI (Extended Unique Identifier) qui servent à identifier les servers/nodes. Différentes clés sont également utilisées pour l'authentification.

- **DevEUI** (Device EUI) : Identité du Device (du Node). Elle doit être configurée.
- **AppEUI** (Application EUI) : Identité de l'application. Elle doit être configurée.
- **NwkSKey** (Network Key) : Clé utilisée par le serveur et le device pour calculer et vérifier le champ MIC (intégrité du paquet). Elle est générée en OTAA, mais doit être configurée en ABP.
- **AppSKey** (Application Secure Key) : Clé utilisée par le serveur et le device pour chiffrer/déchiffrer les données. Elle est générée en OTAA, mais doit être configurée en ABP.
- **AppKey** (Application Key) : Clé utilisée lors de la connexion OTAA. Inexistante en ABP.

Classes

Il existe trois classes d'équipement :

- **Class A** : Elle a consommation d'énergie la plus faible. Il n'y a pas de contrôle à l'envoi de données. Après l'envoi il ouvre 2 fenêtres d'écoute consécutives pour d'éventuels messages en provenance du serveur.
- **Class B** : C'est un compromis entre la consommation d'énergie et la communication bi directionnelle. Il ouvrent des fenêtres d'écoute à des intervalles programmés par des messages envoyés par le serveur
- **Class C** : C'est la plus forte en consommation d'énergie. Elle permet une communication bi directionnelle avec une fenêtre d'écoute permanente.

2.4.3 Architecture LoRa

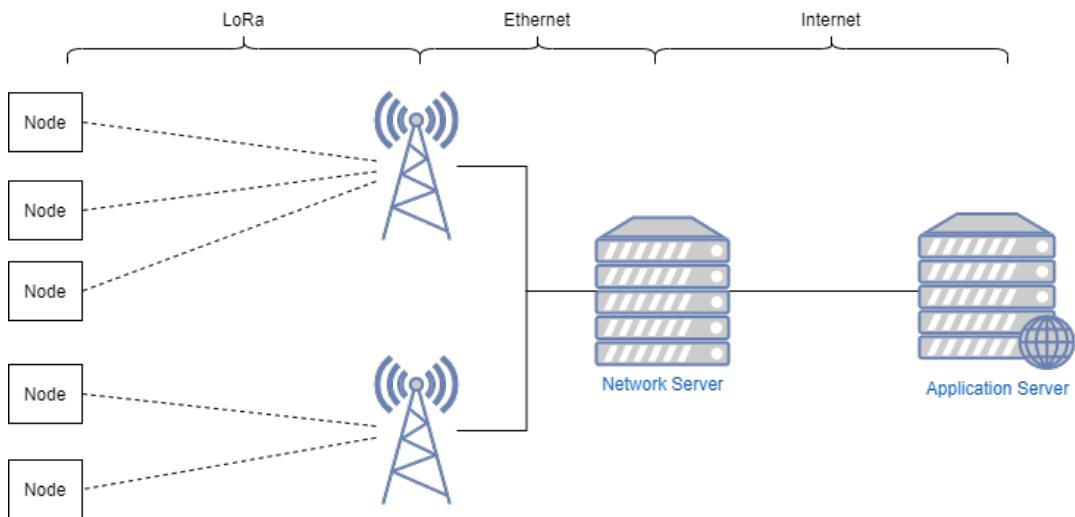


FIGURE 11 – Schéma LoRa Architecture

Il y a quatre composants principaux d'une architecture LoRa :

Node

Le node (end-node, end-device ou encore device) sont les appareils se trouvant en bout de chaîne et communiquant directement en LoRa, à gauche sur le schéma d'architecture 11. Ce sont tous les système embarqué et autres device de l'Internet of things (internet des objets) utilisant le réseau LoRa. Leurs utilisations sont assez variées suivant le domaine.

Gatway

La gateway (ou passerelle) est un émetteur/récepteur permettant de communiquer via LoRa avec tous les Nodes présent dans sa portée. Son but de permettre au node de pouvoir communiquer avec le network server souhaité. La communication avec le network server se fait via Ethernet. La gateway ne gère pas l'authentification, elle se contente de transmettre les packets aux différents network server. Tous les nodes peuvent donc se connecter sur toutes les gateways qui ont pour but de transmettre la connexion avec le network server ciblé.

Network Server

Le network server (serveur de réseau) est le cœur du réseau LoRa. Il a pour buts de gérer les connections entrantes provenant des nodes et transmettent par les gateways. Il gère l'authentification des nodes mais aussi le recensement et l'enregistrement des gateways. Il permet également d'enregistrer plusieurs nodes au sein d'une application et de gérer l'intégration avec un serveur d'application tierce.

Application Server

Le serveur d'application est un serveur récupérant les données provenant du network serveur et les utilisant pour diverses tâches. Cela peut être pour les traiter, pour les stocker ou encore pour les afficher.

2.4.4 Raspberry Pi Hats

Le Raspberry Pi LoRa Hat est un module pour Raspberry, se branchant sur les pins GPIO de celui-ci et permettant au Raspberry d'utiliser le réseau LoRa. Le Raspberry peut utiliser LoRa de deux manières, soit en tant que Node, soit en tant que Gateway. Dans le cadre de ce projet, le Raspberry doit être utilisé comme un Node. Les Gateway utilité sont celles de la ville de Fribourg.

Le projet a initialement été démarré avec un seul Hat. Hat qui nous a été fourni par l'école. Mais ce Hat ayant principalement été conçue pour être une Gateway, il a été décidé de le comparer avec d'autres Hat disponibles sur le marché. Le Hat fourni est le "Lora/GPS HAT" de Dragino et le Hat alternatif trouvé est le "IoT LoRa Node pHAT".

Lora/GPS HAT

Le Dragino Lora/GPS_HAT est un module d'extension LoRaWAN et GPS pour Raspberry PI. Il est compatible pour toutes les versions de Raspberry. Il est disponible à l'adresse <https://www.pi-shop.ch/lora-gps-hat>. La particularité de ce Hat est, en plus de l'utilisation de LoRa, il permet de connaître la localisation GPS (pas utilisé dans le projet). Pour cela il utilise le L80 GPS (Basé sur MTK MT3339).

Il est basé sur les émetteurs-récepteurs SX1276/SX1278.

La documentation officielle décrit comment utiliser ce Hat en tant que Gateway (https://wiki.dragino.com/index.php?title=Lora/GPS_HAT).



FIGURE 12 – Lora/GPS HAT

Caractéristiques :

- **Prix** : 32.90 CHF
- **Bandes de Fréquences** : 868 MHZ
- Prend en charge les connexions LoRaWAN et les modes LoRaP2P

IoT LoRa Node pHAT

Le IoT LoRa Node pHAT est un node LoRa. Il est compatible avec le TTN et autres network server similaire.

Il est basé sur l'émetteur-récepteur SX1276. Le module RAKWireless RAK811 est fourni afin d'utiliser celui-ci.

Une librairie python RAK811 (version 3) est mise à disposition par le fabricant afin d'utiliser le Hat (<https://github.com/AmedeeBulle/pyrak811>).

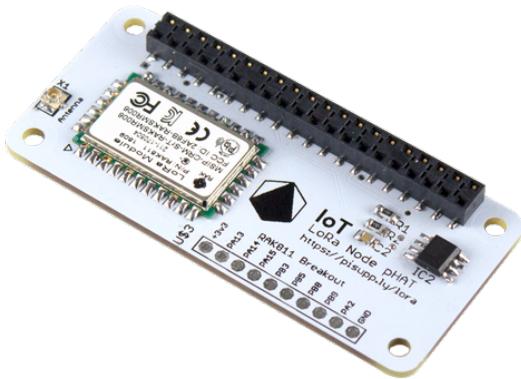


FIGURE 13 – IoT LoRa Node pHAT

Caractéristiques :

- **Prix** : 39.90 CHF
- **Bandes de Fréquences** : 868 MHZ et 915 MHz
- Communique avec le Raspberry Pi par UART.
- Prend en charge les connexions LoRaWAN et les modes LoRaP2P

Comparaison

Les deux Hats proposent le nécessaire pour la communication Lora. Il est possible de les utiliser les deux sur le LoRaWAN ou en Peer-to-peer. Il fonctionne également les deux en 868 MHz qui est la norme européenne.

Lora/GPS HAT :

Avantages :

- Ne coûte pas cher
- Contient un module GPS

Inconvénients

- Documentation officielle faite pour le configurer en tant que gateway.
- Les librairies Raspberry Pi pour l'utilisation de la carte sont des librairies faites par la communauté.
- Les librairies pour Raspberry sont uniquement en C.
- La documentation n'est pas très complète.

IoT LoRa Node pHAT :

Avantages :

- Bien documenté.
- Librairie Python fournie par le constructeur pour l'utilisation en tant que node sur Raspberry.
- Fait pour être utilisé comme node.

Inconvénients

- Coûte plus cher
- Pas de possibilité pour l'utiliser en tant que gateway.

3 Spécification

3.1 Organisation physique

Le système est composés de plusieurs éléments physiques :

- **Hygromètre et thermomètre air ambiant (SHT31-T)** : capteur mesurant le taux d'humidité et la température ambiante
- **Thermomètre globe (MCP9808)** : capteur mesurant la température à l'intérieur d'un globe
- **Pyranomètre (Kipp&Zonen CM3)** : capteur mesurant la quantité d'énergie solaire
- **Interface ADC (ADS1115)** : interface pour convertir les données mesurées par le pyranomètre en données numériques pour le système embarqué
- **Raspberry Pi 3B+** : système embarqué qui connecte tous les autres capteurs ainsi que la carte LoRa
- **Anémomètre (Swema03)** : capteur mesurant la vitesse de l'air
- **Baromètre (BMP280)** : capteur mesurant la pression de l'air
- **Carte LoRa** : Carte d'extension du Raspberry pi qui permet l'envoi via LoRa
- **Lora Gateway** : Gateway à laquelle la carte d'extension doit envoyer les données
- **Serveur d'application** : Serveur qui fait le lien entre les données reçues sur LoRa et la base de données BBData
- **BBData** : Base de données sur laquelle sont stockés les données envoyées via LoRa
- **WebApp** : Application de visualisation des données enregistrées dans BBData
- **Capteur de pollution** : Capteur externe développé à part et envoyant automatiquement ses données sur LoRa grâce à une carte LoRa

Le schéma suivant (Figure 14) démontre l'agencement physique des composants.

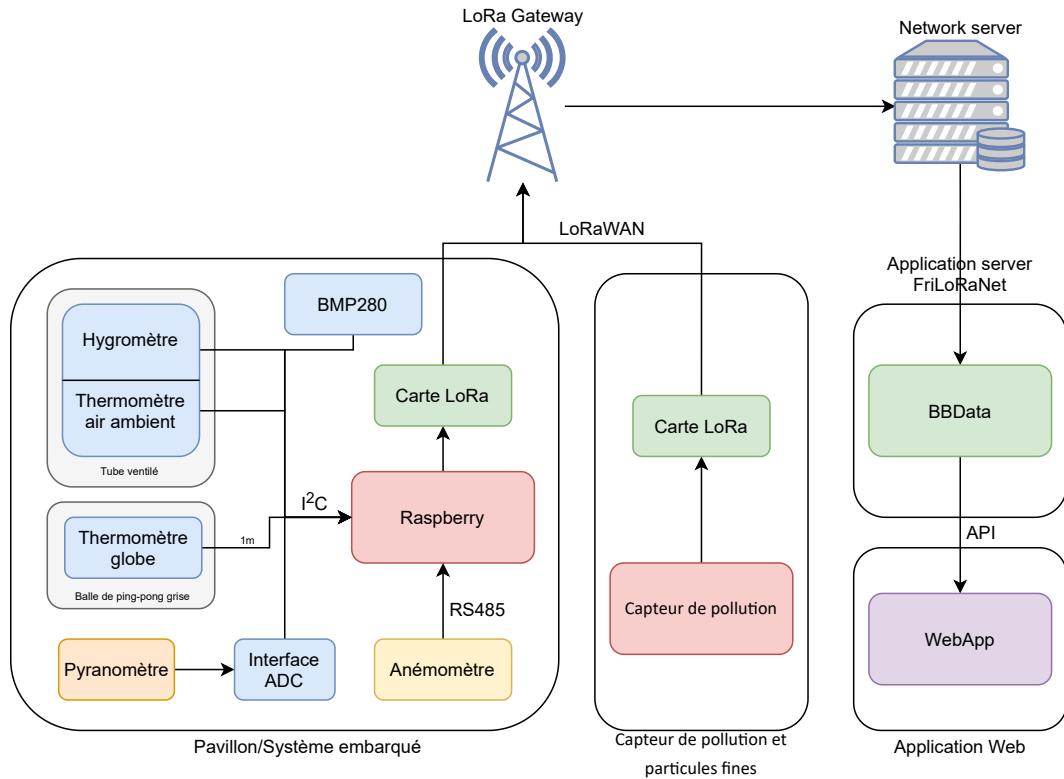


FIGURE 14 – Schéma global du système

3.2 Connectique du Raspberry Pi

Le Raspberry Pi 3B+ possède 4 ports USB ainsi que 40 GPIO. Ces ports permettent de communiquer avec les capteurs et autres composants électroniques nécessaires au fonctionnement du système.

3.2.1 Port USB

Le capteur SWEMA-03 utilise le protocole RS-485 pour recevoir les commandes et envoyer les données mesurées. Il est possible d'utiliser ce protocole via USB. C'est pourquoi le capteur est directement connecté à l'un des ports USB du Raspberry Pi.

3.2.2 GPIOs

Plusieurs composants nécessitent l'utilisation des GPIOs pour transmettre et recevoir des informations. Les composants suivants nécessitent d'être connectés sur le bus I2C :

- Hygromètre et thermomètre air ambiant (SHT31-T)
- Thermomètre globe (MCP9808)
- Interface ADC (ADS1115)
- Baromètre (BMP280)

La carte d'extension LoRa-GPS-hat utilise le protocole SPI pour pouvoir communiquer avec le Raspberry Pi. Une LED ainsi qu'un bouton ON/OFF sont aussi connectés sur les GPIOs :

- Bouton ON/OFF permettant de démarrer et d'arrêter les
- LED clignotant à chaque mesure et clignotant différemment lorsqu'un évènement spécifique se produit

Le branchement complet des GPIO est décrit dans le schéma suivant Figure 15 :

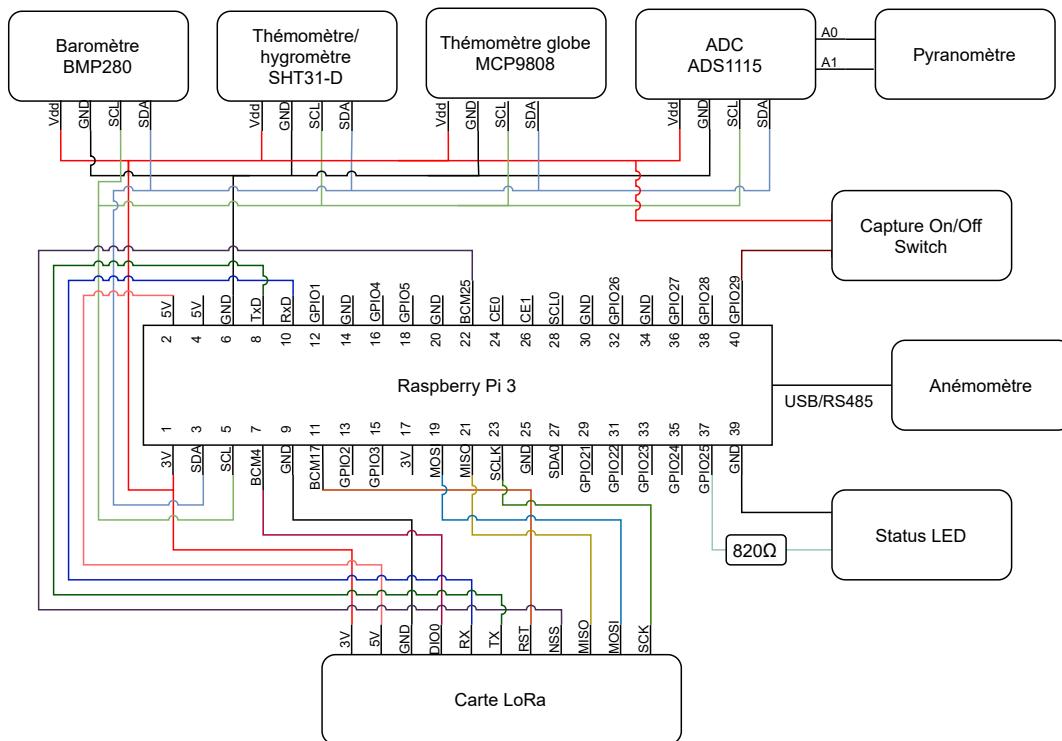


FIGURE 15 – Raspberry connectique

3.3 Maquettes Web

L'application web est responsive et est donc s'adapte aux différents formats d'écrans comme les plus connus : window (Figure 16) et mobile (Figure 17). Le site est composé d'une page unique.

Les informations présente sur la page sont, dans le header, la localisation du pavillon, la date des données affichée et le statut spécifiant si les capteurs sont en live. Dans le corps de la page il y a les 5 les informations des 5 capteurs plus celui du capteur capteur de polution. Pour chaque capteur il y a les dernières valeurs mesurées à l'intérieur et à l'extérieur. En cliquant sur le capteur, le graphique de toutes les mesures de la journée apparaît. Si le système n'est pas en live, ce sont les mesures de la dernière journée de capture qui apparaissent. Dans le footer il y a les différents liens en rapport avec le projet (site officiel du projet et lien FriLoRaNet).

Version pour écran moyen/large :

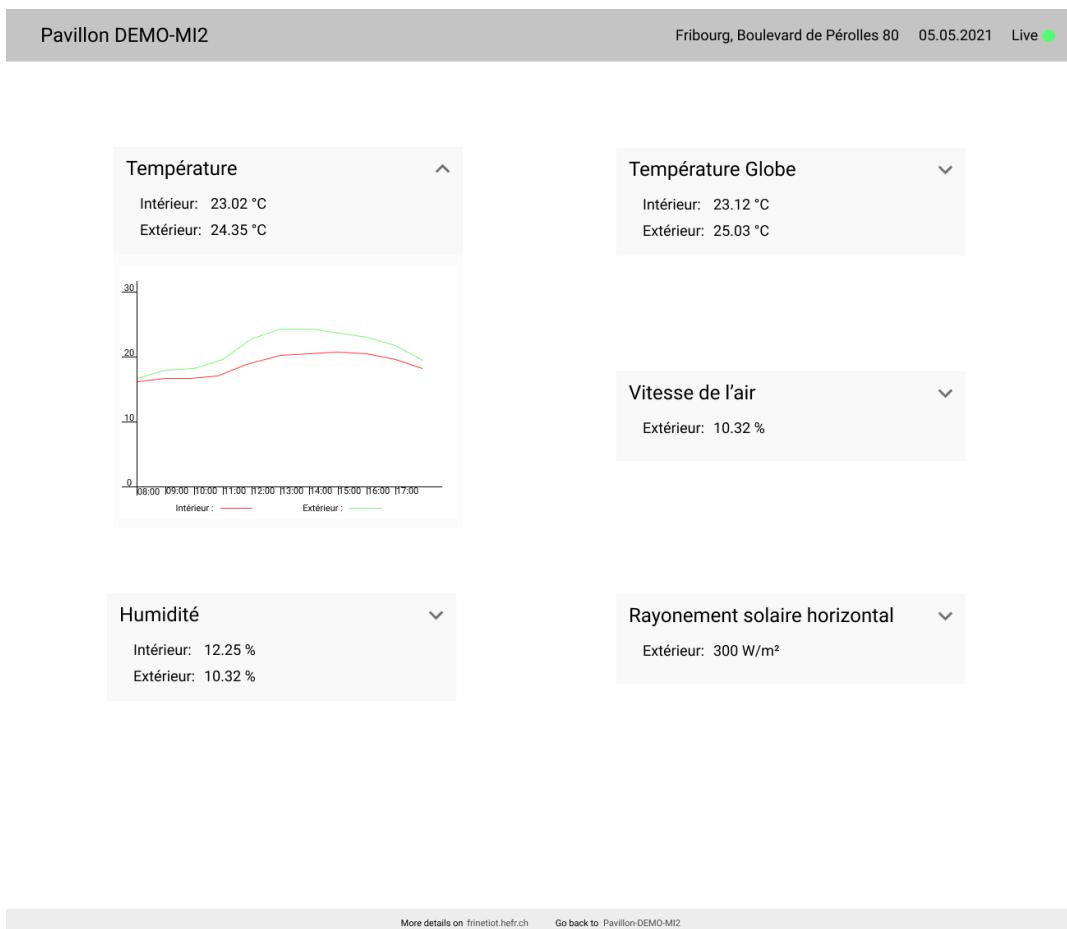


FIGURE 16 – Maquette window

Version pour mobile :

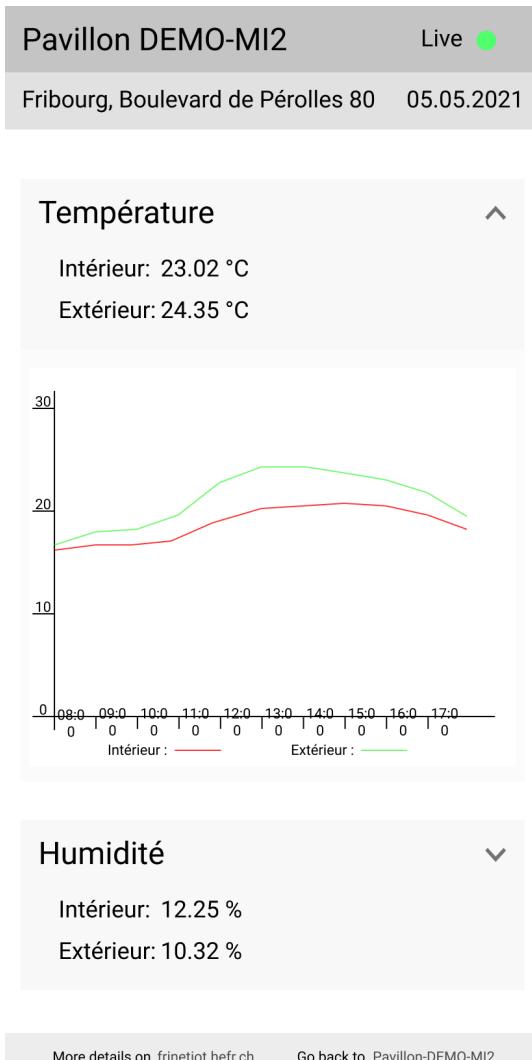


FIGURE 17 – Maquette mobile

Les deux versions sont les mêmes, les éléments se placent à la verticale quand la largeur est trop petite.

4 Conception

Ce chapitre décrit comment le système dans sa globalité a été conçu et par quelles étapes il passe pour effectuer les mesures, les stocker, les envoyer sur la base de données et les afficher sur une application web.

4.1 Application du système embarqué

La conception de l'application de mesure, sauvegarde et d'envoi des données du système embarqué est décrite dans ce chapitre.

4.1.1 Diagramme d'activité

Le diagramme d'activité suivant (Figure 18) décrit comment se comporte l'application du système embarqué lors de son exécution. Les deux processus (celui de sauvegarde des données en local et celui d'envoi des données sur LoRa) sont appelés automatiquement par l'ordonnanceur (scheduler) aux horaires transmis.

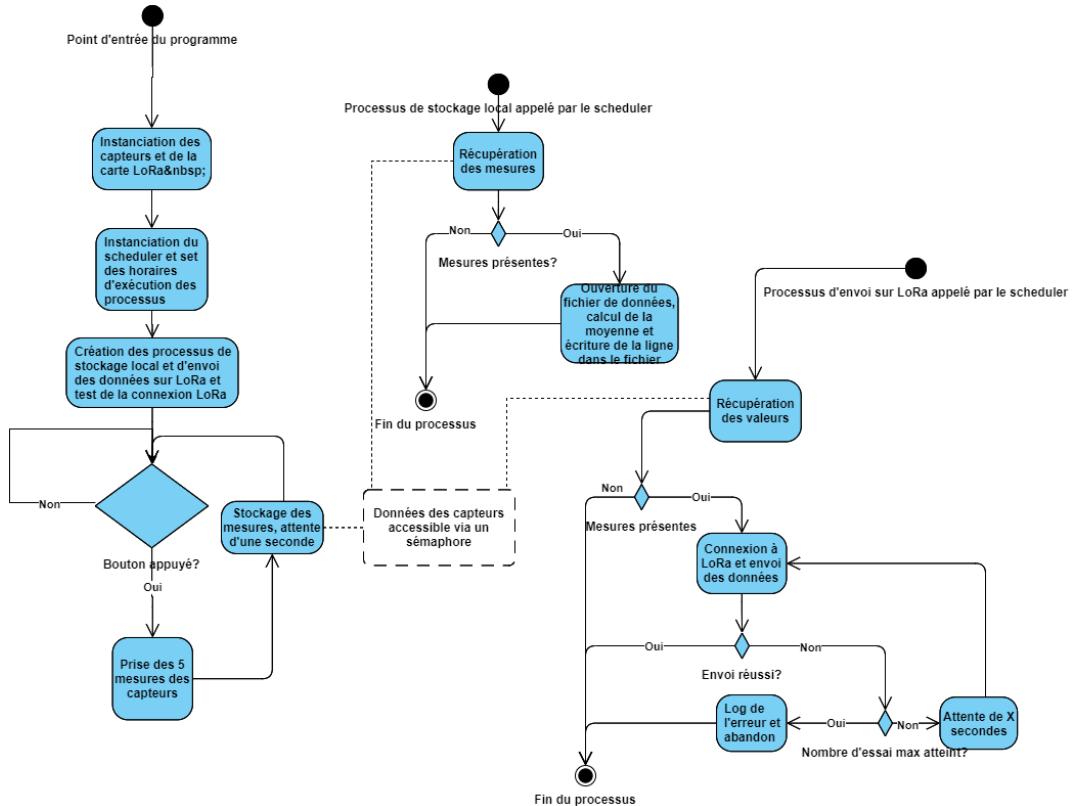


FIGURE 18 – Diagramme d'activité de l'application fonctionnant sur le Raspberry Pi

4.1.2 Prise des mesures

Les mesures sont prises par le système embarqué via le protocole correspondant à la mesure à prendre. Elles sont ajoutées à un dictionnaire avec comme clé le nom de la mesure et comme valeur l'information renvoyée par le capteur correspondant. Une fois que toutes les

mesures ont été prises, elles sont ajoutées à une collection globale qui est accessible via un sémaphore afin d'éviter les problèmes de concurrence lorsque les autres processus lisent la valeur.

4.1.3 Sauvegarde des mesures en local

Le processus de sauvegarde des mesures en local est appelé par l'ordonnanceur à chaque intervalle défini dans le fichier de configuration. Ce processus récupère les données en utilisant le même sémaphore que celui décrit au point 4.1.2. Un fichier est ensuite ouvert et les données y sont inscrites. Le processus est ensuite terminé et l'ordonnanceur attend la fin de l'intervalle pour en créer un nouveau.

Génération du CSV

Le fichier ouvert dans lequel sont écrites les données est un fichier au format CSV. Chaque colonne correspond à la mesure d'un des capteurs ainsi que la date à laquelle la mesure a été prise. Chaque ligne correspond à une nouvelle mesure prise. Les intervalles sont ceux définis dans le fichier de configuration. Le préfixe du fichier est défini dans le fichier de configuration et son suffixe est la date de lancement du programme.

4.1.4 Envoi des données via LoRa

L'ordonnanceur crée un processus à chaque intervalle défini dans le fichier de configuration. Ce processus récupère les données des capteurs de la même manière qu'au point 4.1.3. Le processus met ensuite les données dans une trame qu'il envoie ensuite à la gateway LoRa via sa carte d'extension. Si l'envoi ne fonctionne pas, il attend pendant le temps défini dans le fichier de configuration et réessaie. Il réessaie un nombre de fois égal au nombre défini dans le fichier de configurations. Si aucun essai n'est parvenu jusqu'à la gateway LoRa, l'envoi est abandonné et le processus est détruit.

4.1.5 Accès ssh

Plusieurs possibilités permettent d'accéder au Raspberry Pi à distance afin de pouvoir le configurer. S'il est dans le même sous réseau, utiliser un client ssh pour s'y connecter permet d'avoir un invite de commande semblable à celui directement disponible sur le système d'exploitation installé. Pour pouvoir être dans le même sous réseau, une solution est de faire fonctionner le Raspberry pi comme étant aussi un point d'accès Wi-Fi et ainsi pouvoir s'y connecter avec un client Wi-Fi possédant ssh. Cela permet de configurer le Raspberry Pi de manière simple même lorsqu'il n'est pas connecté à internet.

4.1.6 Fichier de configuration

Pour pouvoir configurer le système embarqué afin qu'il sache avec quel identifiant envoyer les données, quels capteurs utiliser, quel intervalle l'ordonnanceur doit attendre entre deux enregistrements ou deux envois et combien d'essais il effectue avant d'abandonner l'envoi des trames LoRa, un fichier de configuration est utilisé.

4.1.7 Crédit de logs (journalisation)

Un système de logs de création de logs permet d'enregistrer dans un fichier les moments notables par lequel passe l'application. Le préfixe du fichier est défini dans le fichier de configuration et son suffixe est la date de lancement du programme.

Commutateur de mesures des données

Un commutateur on/off permet au système embarqué de savoir lors qu'il doit prendre des mesures et lorsqu'il doit arrêter. Lorsque le commutateur est sur la position off, l'application du système embarqué ne mesure pas de données. Lorsque le commutateur est sur la position on, les mesures sont prises.

Clignotement de la LED

Plusieurs vitesses de clignotement de la LED ont été mises en place pour avoir un retour sur l'état dans lequel le système embarqué est. Ils sont décrits ci-dessous :

- **Pas de clignotement** L'application vient d'être démarrée et attend que le bouton soit appuyée pour commencer les mesures
- **Clignotement une fois par trois secondes** Le bouton de capture des mesures à été mis sur position OFF alors que l'application fonctionnait. Les processus continuent d'être exécutés, mais aucune nouvelle mesure n'est prise.
- **Clignotement une fois par seconde** L'application prend des mesures chaque seconde et les sauvegarde dans le fichier de données CSV chaque minute.
- **Clignotement dix fois par seconde** L'application est entrain d'envoyer ses données sur le réseau LoRa. Les mesures et les sauvegardes dans le fichier de données CSV continuent d'être faites durant ce temps.

4.2 LoRa

La connexion Lora permet de transmettre les données provenant des capteurs vers la base de données BBData. Pour cela les données doivent être formatées, envoyées sur LoRa, réceptionnées et décodées par le network server et enfin enregistrer dans la base de données.

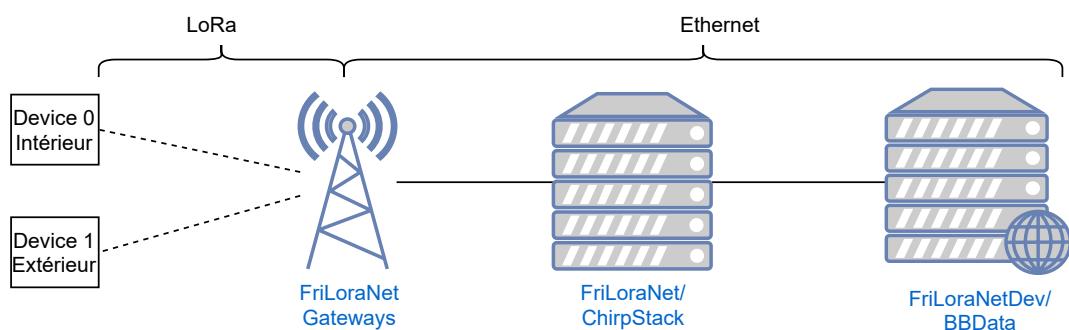


FIGURE 19 – Schéma LoRa DEMO-MI2

4.2.1 Format des données

Afin de minimiser la taille des trames et d'utiliser le moins possible le réseau LoRa les données sont reformatées avant d'être envoyé.

Toutes les données des capteurs sont envoyées sous forme d'entier de 2 bytes (16 bits). Ce qui permet d'envoyer des données de -32'768 à 32'767. La position de la virgule est affectée de manière fixe pour chaque valeur. Les données sont converties en entier avant l'envoi puis retransformées en nombre à virgule à la réception. Exemple :

Température : 22.34°C -> code : 22.34*100 vers entier -> envoyer de 2234 (sur 2 bytes) ->
 decode : 2234/100 vers float -> 22.34°C

Cela permet d'envoyer des nombres à virgule avec un minimum de données. Les besoins du projet ne requièrent pas une précision de plus de deux décimale, sauf pour la vitesse du vent qui sera sur trois décimales.

Les plages de valeurs acceptées pour les capteurs sont les suivantes :

- Température de l'air : -327,68 à 327,67
- Humidité : -327,68 à 327,67
- Température Globe : -327,68 à 327,67
- Vitesse du Vent : -32,768 à 32,767
- Rayonnement Solaire : -327,68 à 327,67

4.2.2 Trame

La trame est composée de 15 bytes. Elle possède 1 byte de contrôle, 4 bytes de timestamp et 2 bytes par capteur (5 capteurs).

Bits	1	2-5	6-7	8-9	10-11	12-13	14-15
Data	Control	Timestamp	Pyrano	Thermo	ThermoG	Humi	Vent

Le byte de contrôle est composé du flags de test, de l'id du device (identifiant de quel raspberry pi envoie la trame) et de la version de la trame.

Bytes	1	2-4	5-8
Data	Test flag	Device Id	Version

Les champs du byte de contrôle ont pour utilité :

- **Test flag** : Déterminé si la trame est une trame de test, ce qui signifie que les valeurs ne sont pas à prendre en compte. 1 si c'est une trame de test sinon 0.
- **Device Id** : Id du device (du Rapsberry) qui envoie la trame. L'id 0 est pour le device intérieur et 1 est pour le device extérieur. Les autres id ne sont pas utilisés, mais sont disponibles au besoin.
- **Version** : Numéro de la version de la trame. La trame est actuellement à la version 1. La version doit être incrémentée si la trame est modifiée dans le futur.

Le Timestamp est un entier de 32 bits qui correspond à la date et l'heure (en seconde) de la mesure. Cette valeur est celle qui est enregistrée comme horodatage de la mesure dans BBData.

Les autres bytes sont les valeurs des capteurs. L'ordre des valeurs est fixe et est comme suit :

- Rayonnement Solaire
- Température de l'air
- Température Globe
- Humidité
- Vitesse du Vent

Les données sont formatées sur deux bytes comme décrits dans la section 4.2.1.

Grâce à la compression des données comme décrite précédemment, la limite d'utilisation du réseau LoRa de 1% n'est pas dépassée. Cela pour l'envoi d'une trame toutes les 5 minutes.

4.2.3 Network Server

Le network server est un serveur ChirpStack, site officiel : <https://www.chirpstack.io/>. Il réceptionne les données provenant des device, les décode et les transmet au serveur FriLoRaNetDev (serveur déjà existant). Le serveur ChirpStack est également déjà existant, son adresse est <http://friloranet.tic.heia-fr.ch:8080/>

Les éléments mis en place sur le serveur ChirpStack sont :

- **Application** : Création d'une nouvelle application dans le serveur afin de pouvoir y enregistrer les devices et l'intégration.
- **Device-profile** : Création d'un profile pour nos devices afin de leurs enregistrer les paramètres de connexion, la classe, l'authentification (OTAA dans notre cas) et une fonction de décode. La fonction de décode, en JavaScript, sera appelé pour chaque trame entrante.
- **Device** : Chaque device doit être enregistré dans l'application. Un device correspond à un node LoRa, à un Raspberry dans notre cas. Le DevEUI, provenant du node, doit être fourni pour l'enregistrer. Et l'AppKey peut être récupéré pour que le device puisse établir une connexion.
- **Integration** : L'intégration permet de transmettre les données reçues vers un serveur tiers. Dans notre cas elle seront transmises en HTTP au serveur FriLoRaNetDev.

4.3 Application Web

L'application web a pour but d'afficher les données collectées durant la journée ou les données de la dernière session de capture. Elle permet également de voir en direct le statut de l'installation (si l'installation est en fonctionnement). Cela pour tous les capteurs mis en place, intérieur et extérieur, ainsi que pour le capteur de pollution associé. L'application Web doit être disponible en tout temps. Un code QR sera également associé à l'adresse du site afin d'y accéder plus rapidement sur place.

4.3.1 Serveur

L'application web est une application Flask en Python 3. Son but est de fournir une page de présentation des données récoltées ainsi formater et fournir les données elles-mêmes. Il

communique directement avec l'API BBData afin d'obtenir les données.

Gunicorn est également utilisé afin de fournir une interface disponible pour UNIX et utilisable par un proxy tel que NGINX. Il utilise le Web Server Gateway Interface (WSGI) qui est une interface entre serveur et application disponible pour python. Il permet donc de gérer le serveur Flask et d'offrir une interface vers celui-ci.

La partie accessible depuis l'extérieur de l'application est un serveur NGINX. C'est un serveur Web ainsi qu'un reverse proxy. Dans notre cas c'est la fonction de reverse proxy qui est utilisé afin d'accéder à l'interface fournie par Gunicorn (sous forme de socket UNIX).

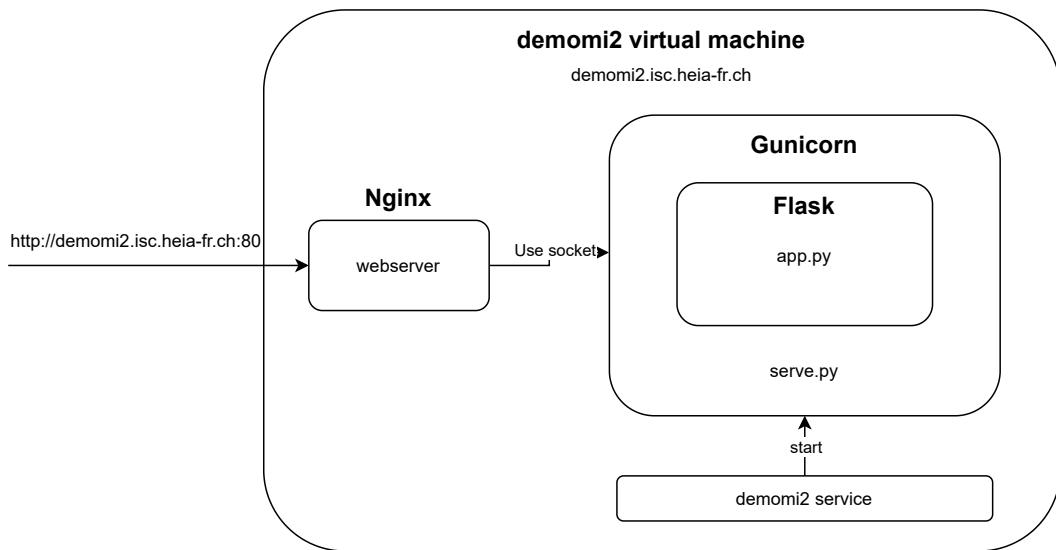


FIGURE 20 – Schéma Serveur Web

Les requêtes entrantes, sur le port 80, sont donc réceptionnées par NGINX et transmises à Gunicorn qui contient le serveur Flask. NGINX et Gunicorn sont configurés en tant que services et sont lancés au démarrage du serveur.

4.3.2 BBData

Le stockage et la mise à disposition des données dans BBData est faite par l'institut RO-SAS. Un fois les objets créés dans BBData, ils correspondent à un id qui peut être utilisé afin de récupérer les données.

Afin de récupérer des données sur BBData, il faut s'authentifier :

Route : /login

Body :

```
{
    "username": "string",
    "password": "string"
}
```

Réponse :

```
{
  "id": 0,
  "secret": "string",
  "expirationDate": "2021-05-20T14:09:47.353Z",
  "userId": 0,
  "readOnly": true,
  "description": "string"
}
```

Cela permet de récupérer le bbuser (UserId) ainsi que le bbtoken (secret). Ces deux informations permettent de requêter l'API afin d'obtenir des valeurs.

Il est possible de récupérer des valeurs, correspondant à un objet (via son id) dans une certaine plage de temps, comme suit :

Route : /objects/objectId/values ?from=from_date&to=to_date
 Header :

```
{
  "bbuser": "string",
  "bbtoken": "string"
}
```

Réponse :

```
[
  {
    "timestamp": "2021-05-20T14:00:25.781Z",
    "value": "string",
    "comment": "string"
  }
]
```

Les données retournées sont un tableau JSON de toutes les valeurs enregistrées dans ce laps de temps pour cet objet.

4.3.3 Routes

Le serveur web met 3 routes à disposition.

/

La racine permet de récupérer la page HTML index. Cette page est celle affichant les graphiques.

/is-live

La route "is live" permet de récupérer des informations sur le statut actuel de la prise de données. Les données retournées sont formatées comme suit :

```
{
  "islive": bool,
```

```

    "date_last_measure": "string",
    "location": "string"
}

```

Les données reçues sont :

- **islive** : Le statut des capteurs, si au moins un capteur à retourné des valeurs dans les 15 dernières minutes.
- **date_last_measure** : La date de la dernière mesure prise parmi tous les capteurs.
- **location** : La localisation renseignée dans le fichier de configuration.

/get-data ?type=type_name

La route "get data" permet de récupérer les données pour un type de données (ex. température, vitesse du vent,...). Les types de données requérables sont les suivants :

- **temp** : Température de l'air (°C)
- **humi** : Humidité (%)
- **glob** : Température Globe (°C)
- **wind** : Vitesse du Vent (m/s)
- **suni** : Rayonnement Solaire (W/m²)
- **polu** : Particules fines PM2.5 (g/m³)

Le type doit être spécifié comme paramètre de la requête.

Le données retournées sont formaté comme suit :

```

{
  "data": "CSV: timestamp,Intérieur,Extérieur",
  "date": "string",
  "type": "string",
  "lastint": "string",
  "lastext": "string"
}

```

Les données reçues sont :

- **data** : Les données des capteurs de la dernière journée de capture. Elles sont sous forme CSV avec comme champs le timestamp, la valeur Intérieur et la valeur Extérieur. Les capteurs de rayonnement solaire et de pollution ne contiennent qu'une valeur extérieur.
- **date** : La date de la dernière mesure prise.
- **type** : Le type de données demandé.
- **lastint** : La dernière valeur intérieure reçue, Offline si pas de valeur dans les 15 dernières minutes.
- **lastext** : La dernière valeur extérieure reçue, Offline si pas de valeur dans les 15 dernières minutes.

4.3.4 Client

L’application cliente est une page statique en HTML. L’application possède une seule page qui a pour but d’afficher les données sous forme de graphiques. Les données sont récupérées par JavaScript par le browser sur le serveur web. Le design de l’interface est présenté dans la section 3.3.

La librairie utilisée pour générer les graphiques est dygraphs, <https://dygraphs.com/>. C’est une librairie JavaScript permettant de générer des graphiques à partir de données CSV.

5 Réalisation

Ce chapitre décrit comment les différents éléments du projet sont réalisés et quels outils ont été utilisés.

5.1 Environnement de développement et logiciels utilisés

Toute l'implémentation de l'application du système embarqué ainsi que celle de l'application web ont été réalisées avec Visual Studio Code², un éditeur de code extensible.

Le code a été transmit aux Raspberry Pi grâce à³, un logiciel libre de synchronisation de fichiers.

Le traitement de la trame LoRa après sa récupération est effectuée directement sur le network-server chirpstack⁴.

La collaboration entre les deux étudiants du projet à été réalisée grâce à gitlab⁵, un outil de gestion de versions du code efficace.

5.2 Implémentation de l'application du système embarqué

L'application du système embarqué à été intégralement développée pour le Raspberry Pi 3B+. Elle a été presque totalement écrite en python 3. Elle gère la prise des mesures, leur stockage, leur envoi et la gestion des erreurs et de la journalisation.

5.2.1 Librairies

Les librairies python 3 utilisées sont les suivantes :

logging : Librairie permettant d'avoir un système de journalisation efficace qui peut être à la fois écrit dans un fichier et dans l'invite de commande qui a lancé l'application

time et datetime : Librairies qui permet d'accéder à l'heure du système et faire des conversions d'heure en timestamp et inversement. Cette librairie gère aussi les fuseaux horaires et permet d'avoir des mesures prises aux bonnes heures à partir du moment où l'heure du Raspberry Pi est réglée correctement.

threading : Librairie permettant de gérer la concurrence dans l'application. Elle permet de créer des processus et de gérer les systèmes de blocage à l'aide de sémaphores.

statistics : Librairie utilisée pour effectuer des statistiques sur les données mesurées comme la moyenne lors de l'envoi sur LoRa

GPIO : Librairie permettant d'accéder aux GPIOs du Raspberry Pi et de les régler dans le mode désiré afin de pouvoir les utiliser.

2. <https://code.visualstudio.com/>

3. rsync.ubuntu-fr.org/rsync

4. <http://friloranet.tic.heia-fr.ch:8080//organizations/1>

5. <https://about.gitlab.com/fr-fr/>

apscheduler : Librairie permettant de créer et utiliser un ordonnanceur efficace à qui on donne une méthode et des intervalles réguliers. Un processus est ensuite créé aux horaires décrits pour qu'il exécute la méthode donnée.

struct : Librairie qui permet de mettre des données dans une structure précise afin de les ordonner avant de les envoyer via LoRa.

serial : Librairie permettant de communiquer aux appareils connectés en série au Raspberry Pi.

subprocess : Librairie permettant d'appeler des commandes système et de gérer le retour de ces commandes. Elle est utilisée pour appeler le code C qui permet l'envoi sur LoRa avec le LoRa-GPS-Hat.

board : Librairie offrant des fonctionnalités génériques pour différentes cartes. Elle est nécessaire pour utiliser les voies du bus I2C SDA et SCL décrites au point 2.1.1.

busio : Librairie permettant de formater du texte. Elle est utilisée pour formater la réception des données via le bus I2C.

adafruit : **bmp280**, **mcp9808**, **ads1x15**, **sht31d** : Librairies utilisées pour communiquer avec les différents capteurs connectés au bus I2C.

rak811 : Librairie permettant de communiquer avec la carte LoRa afin d'envoyer des trames LoRa avec le IoT LoRa Node pHAT.

5.2.2 Composants

Les composants utilisés dans le Raspberry Pi sont dans le répertoire /home/pi/src. Ils sont décrits dans la Figure 21.

```

src
  Configuration.py
  demo_mi2.py
  device
    anemometer.py
    bmp280.py
    mcp9808.py
    pyrano.py
    sht31_d.py
  lora
    Makefile
    README.MD
    send_lora
    send_lora.cpp
    send_lora.py

```

FIGURE 21 – Organisation des différents composants du Raspberry Pi

Le composant *Configuration.py* est le fichier contenant toutes les options de configuration de l'application. Une explication détaillée des options du fichier est disponible au point 7.1.2

Le composant principal est *demo_mi2.py*. C'est le point de départ de l'application et c'est dans ce module que l'ordonnanceur est créé et que les processus sont exécutés.

Le composant *send_lory.py* permet d'envoyer les données sur le réseau LoRa. Les composants présents dans le répertoire *device* sont les interfaces avec les différents capteurs nécessaires au fonctionnement de l'application.

Les composants présents dans le répertoire *lora* sont les fichiers nécessaires à la compilation du code C++ permettant de gérer la carte d'extension LoRa-GPS-Hat.

5.2.3 Gestion des logs (journalisation)

Pour pouvoir générer les logs et les écrire dans un fichier, la librairie logging est utilisée. Elle permet de définir facilement le formatage que prendront les logs et quels niveaux seront écrits. Dans l'application du Raspberry Pi, 3 niveaux différents sont utilisés :

- info : Toutes les informations utiles pour savoir ce qu'est en train de faire l'application
- warning : Permet d'afficher les informations qui sont importantes à lire (par exemple si le bouton de démarrage des captures n'est pas sur la position ON)
- error : Tous les problèmes que rencontre l'application sont écrits avec ce niveau de log. Les erreurs de prises des mesures, les erreurs d'envoi des données via LoRa, etc...

Les logs permettent de vite comprendre qu'est-ce que l'application est en train d'effectuer et lorsqu'il y a des problèmes ils permettent de comprendre quels sont les problèmes même après l'exécution de l'application.

5.2.4 Gestion de l'ordonnanceur

L'ordonnanceur permet d'appeler des méthodes à des heures précises. Dans le cadre du projet, il est nécessaire d'enregistrer des mesures en local toutes les minutes et d'envoyer les données sur le réseau LoRa toutes les 5 minutes. Il faut donc définir quelles méthodes l'ordonnanceur va appeler et lui fournir les détails des moments auxquels les processus vont être créés. Il va ensuite gérer lui-même la création des processus aux bonnes heures, leurs exécutions et leurs destructions.

5.2.5 Mesures des données (méthode `measure_data()`)

Le processus principal effectue une mesure des données à intervalle régulière chaque seconde puis les stocke dans un dictionnaire contenant des tableaux. Chaque entrée du dictionnaire correspond à un type de mesure et chaque valeur correspond à un tableau des valeurs qui ont été mesurées. Cette structure de données est ensuite enregistrée dans la mémoire vive du système embarqué à deux exemplaires. Un pour les sauvegardes en local et un pour les envois sur LoRa. Les mesures sont faites via les composants correspondants aux capteurs. Toutes les heures, la pression est récupérée du baromètre pour ensuite la donner à l'anémomètre afin qu'il soit calibré à nouveau.

5.2.6 Sauvegarde des valeurs mesurées en local (méthode save_data())

Chaque minute, le processus de sauvegarde des données est exécuté par l'ordonnanceur. Il récupère la structure de données qui lui est destinée et qui contient les mesures réalisées puis la copie. Une fois cette opération réalisée, la structure de données est réinitialisée afin que les mesures qui y sont ajoutées soient nouvelles. Grâce à la copie des mesures, une moyenne est effectuée sur chaque entrée du dictionnaire, pour ne garder ainsi qu'une seule valeur par tableau. Ces valeurs sont ensuite formatées puis ajoutées à un fichier csv afin qu'elle soit utilisable sous forme de tableau.

5.2.7 Envoi des données sur LoRa (méthode send_to_lora())

Afin de tester la connexion avec LoRa au début de l'application et envoyer une trame au serveur pour qu'il puisse mettre à jour l'interface et afficher les capteurs comme étant en direct, une trame de test est envoyée. Elle se comporte comme les autres trames, mais à un bit de contrôle à 1.

Chaque cinq minutes, le processus d'envoi des données est exécuté par l'ordonnanceur. Il récupère la structure de données qui lui est destinée et qui contient les mesures réalisées puis la copie. Une fois cette opération réalisée, la structure de données est réinitialisée afin que les mesures qui y sont ajoutées soient nouvelles. Grâce à la copie des mesures, une moyenne est effectuée sur les cinq dernières minutes et les moyennes des données mesurées sont ajoutées dans une structure afin d'être envoyées via LoRa. L'appel au composant LoRa est fait avec la structure. Si l'appel réussi, le processus se termine. S'il lance une exception, car il n'a pas réussi à joindre le serveur, une nouvelle tentative est effectuée après avoir attendu un temps défini dans le fichier de configuration. Si trop de tentatives échouent, le processus abandonne l'envoi et termine.

Afin d'avoir un maximum de réussite d'envoi des données via le réseau LoRa, il est préférable de faire envoyer les deux systèmes embarqués à des horaires décalés. Une option du fichier de configuration permet de décaler l'envoi sur l'un des deux systèmes embarqués. Les données envoyées restent inchangées.

5.3 LoRa et Chirpstack server

5.3.1 Chirpstack

Le serveur ChirpStack est le "Network Server" de l'infrastructure LoRa FriLoRaNet. Il permet de récupérer les données émises sur LoRa et de les envoyer sur un autre serveur (en l'occurrence FriLoRaNet). Il est disponible à l'adresse <http://friloranet.tic.heia-fr.ch:8080/>

Device profile

Le "Device-profile" permet de créer un profile pour équipements similaires. Un profile "PavillonDEMOMI2" a été créé. Il permet d'utiliser les mêmes paramètres LoRa pour tous nos systèmes embarqués. La configuration est la suivante :

Device-profiles / PavillonDEMOMI2 DELETE

GENERAL	JOIN (OTAA / ABP)	CLASS-B	CLASS-C	CODEC	TAGS
Device-profile name *	PavillonDEMOMI2	A name to identify the device-profile.			
LoRaWAN MAC version *	1.0.2	The LoRaWAN MAC version supported by the device.			
LoRaWAN Regional Parameters revision *	B	Revision of the Regional Parameters specification supported by the device.			
ADR algorithm *	Default ADR algorithm	The ADR algorithm that will be used for controlling the device data-rate.			
Max EIRP *	14	Maximum EIRP supported by the device.			
Uplink interval (seconds) *	86400	The expected interval in seconds in which the device sends uplink messages. This is used to determine if a device is active or inactive.			

FIGURE 22 – ChirpStack Device-Profile

L’authentification OTAA est également activée pour nos appareils :

Device-profiles / PavillonDEMOMI2 DELETE

GENERAL	JOIN (OTAA / ABP)	CLASS-B	CLASS-C	CODEC	TAGS
<input checked="" type="checkbox"/> Device supports OTAA					
UPDATE DEVICE-PROFILE					

FIGURE 23 – ChirpStack Device-Profile OTAA

JSON parse

Afin de récupérer les données plus facilement pour les enregistrer dans BBData, il est possible de parser les données reçues (sous forme de bytes) directement en JSON. Pour cela il faut, dans le Device-Profile, sous l’onglet "CODEC" activer "Custom JavaScript codec functions".

Device-profiles / PavillonDEMOMI2

GENERAL	JOIN (OTAA / ABP)	CLASS-B	CLASS-C	CODEC	TAGS
Payload codec <input checked="" type="checkbox"/> Custom JavaScript codec functions					

FIGURE 24 – ChirpStack Codec

Et implémenter la fonction Decode :

```
function Decode(fPort, bytes, variables) {
    return {
        IsTest: ((bytes[0] & 0x80)!=0),
        DeviceId: ((bytes[0] & 0x70)>>4),
        Version: (bytes[0] & 0x0F),
        Timestamp: (((bytes[1]<<24) | (bytes[2]<<16) | (bytes[3]<<8) | (bytes[4])) ,
        SunIntensity: (((((bytes[5]<<8 | bytes[6]) << 16) >> 16) /100).toFixed(2),
        Temperature: (((((bytes[7]<<8 | bytes[8]) << 16) >> 16) /100).toFixed(2),
        TemperatureGlobe: (((((bytes[9]<<8 | bytes[10]) << 16) >> 16) /100).toFixed(2),
        Humidity: (((((bytes[11]<<8 | bytes[12])<< 16) >> 16)/100).toFixed(2),
        Windspeed: (((((bytes[13]<<8 | bytes[14]) << 16) >> 16)/1000).toFixed(2)
    };
}
```

Cela permet de récupérer les données déjà décodées directement dans le JSON des données reçues. Un nouvel objet "objectJSON" apparaît :

```
"objectJSON": "{ \"DeviceId\":0, \"Humidity\":\"2.10\", \"IsTest\":false,
\"SunIntensity\":\"56.45\", \"Temperature\":\"23.12\", \"TemperatureGlobe\":\"24.01\",
\"Timestamp\":1620146783, \"Version\":1, \"Windspeed\":\"3.20\"}";
```

Application

Il est nécessaire de créer une application dans ChirpStack. Notre application est nommée "PavillonDEMOMI2".

12 PavillonDEMOMI2 HEIA-FR PS6 - Connectique et accès aux données du pavillon demo MI-02 - Julien Piguet & Denis Rosset

FIGURE 25 – ChirpStack Application

Dans l'application il est possible de voir la liste de devices enregistrée.

Applications / PavillonDEMOMI2					
DEVICES		APPLICATION CONFIGURATION		INTEGRATIONS	
Last seen	Device name	Device EUI	Device profile	Link margin	Battery
a few seconds ago	Device0-Interieur	60c5a0fffe798627	PavillonDEMOMI2	n/a	n/a
3 minutes ago	Device1-Exterieur	60c5a0fffe7985dc	PavillonDEMOMI2	n/a	n/a

Rows per page: 10 < 1-2 of 2 >

FIGURE 26 – ChirpStack Application Devices Liste

Il est possible de changer le nom et la description de l'application.



Application name *
PavillonDEMOMI2
The name may only contain words, numbers and dashes.

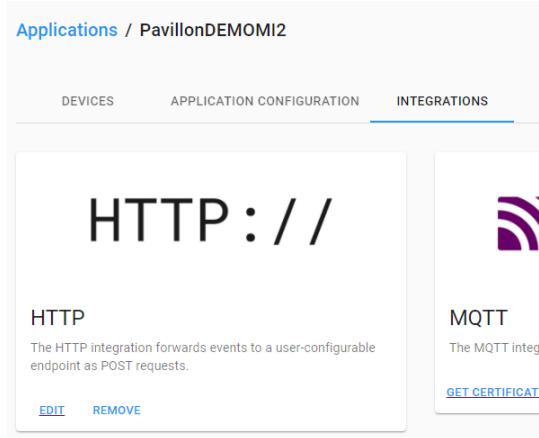
Application description *
PS6 - Connectique et accès aux données du pavillon demo MI-02 - Julien Piguet & Denis Rosset

Note: The payload codec fields have moved to the device-profile.

FIGURE 27 – ChirpStack Application Configuration

Intégration

Il est possible dans l’application de rajouter une intégration afin que les données y soient automatiquement envoyées. Ici elles sont envoyées en HTTP sur le serveur friloranet sous forme JSON. Il est possible de la modifier dans l’onglet intégration.



HTTP	MQTT
The HTTP integration forwards events to a user-configurable endpoint as POST requests. EDIT REMOVE	The MQTT integ GET CERTIFICATE

FIGURE 28 – ChirpStack Intégration

Avec "EDIT" il est possible de modifier l'intégration :

FIGURE 29 – ChirpStack Intégration HTTP

Device

Il faut créer un device dans l'application pour chaque device physique. Les informations à renseigner lors de la création sont le nom du device, la description, le device EUI et le device-profile. Le device profile est celui créé précédemment. Le device EUI doit être récupérée sur le device lui-même.

FIGURE 30 – ChirpStack Device Create

Il est ensuite possible de l'Application Key pour l'authentification OTAA dans l'onglet "Keys (OTAA)"

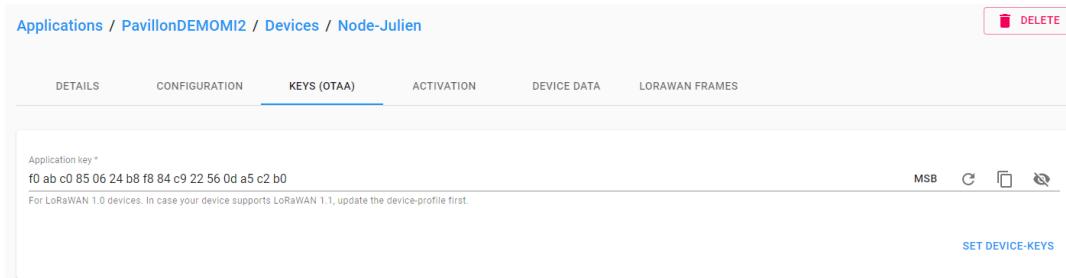


FIGURE 31 – ChirpStack Device OTAA

Il est ensuite possible de voir les connexions et les données échangées avec ce device dans l'onglet "DEVICE DATA".

5.3.2 BBData

Dans la base de données BBData, les données sont sous forme de tuple avec un timestamp, une valeur et un commentaire. Nos deux systèmes de mesures météorologiques ont les noms :

- #128 Demo MI2 - Intérieur
- #129 Demo MI2 - Extérieur

Nom des devices présentes sur BBData

Chaque système présent sur BBData possède 5 capteurs qui leur sont attribués :

- device 0, Humidity (13560)
- device 0, SunIntensity (13561)
- device 0, Temperature (13562)
- device 0, TemperatureGlobe (13563)
- device 0, Windspeed (13564)
- device 1, Humidity (13565)
- device 1, SunIntensity (13566)
- device 1, TemperatureGlobe (13568)
- device 1, Windspeed (13569)
- DecentLab01-AirQuality-PM2.5 (13545)

En plus de ces identifiants, nous récupérons aussi la valeur du capteur de pollution qui a l'identifiant 13545 afin de l'afficher sur l'application web. Les valeurs entre parenthèses décrites dans la liste ci-dessus sont les identifiants des tuples et permettent de récupérer les données présentes dans les tuples lorsqu'ils sont demandés à l'application.

Trame de test

Lors de la réception d'une trame de test, le commentaire "test" est ajouté à la trame pour que lors de la récupération des données depuis BBData, le serveur puisse savoir que cette trame est une trame de test et ainsi l'ignorer et ne pas l'afficher dans les graphs. Grâce à cette information, le serveur peut mettre à jour son statut et afficher "live" lorsqu'une trame de test a bien été reçue. Cela permet de vérifier au démarrage du programme que la connexion LoRa est fonctionnelle.

5.4 Application et serveur web

5.4.1 Serveur Web

Le serveur qui héberge l'application est une machine virtuelle fournis par la HEIA-FR. Son adresse est "demomi2.isc.heia-fr.ch".

C'est une machine virtuelle Ubuntu 20.04.2 LTS. Les ports 80 et 443 sont ouverts vers l'extérieur. Il est possible d'y accéder en ssh avec son adresse depuis le réseau de l'école. Il est également possible de la gérer depuis l'interface VMWare vSphere Client, <https://vmware.tic.eia-fr.ch/>.

Le serveur Web possède un reverse proxy NGINX afin de rediriger les requêtes à destination du port 80 vers le serveur Flask. Le serveur Flask est encapsulé par Gunicorn qui est configuré en tant que service.

5.4.2 Application Web

L'application web est une application Flask mono page.

Structure

Les fichiers qui constituent l'application sont les suivants :

```
webapp/
    Configuration.py
    app.py
    serve.py
    templates
        index.html
```

Description des fichiers :

- **app.py** : Serveur Web Flask. Répond au trois routes "/", "/is-live" et "/get-data".
- **serve.py** : Point d'entrée de Gunicorn. Permet de lancer l'application Flask.
- **Configuration.py** : Fichier de configuration. Contiens la localisation du pavillon.
- **index.html** : Page web de l'application. Retourné par la racine de l'application.

Application Flask

Fonctions de l'application Flask :

root(name)

Fonction appelé lors d'une requête sur la route "/". Elle permet d'établir la connexion avec BBData et renvoi la page index grâce à la fonction "render template" de Flask. Si la connexion avec BBData ne fonctionne pas ou s'il n'y a pas de données utilisables (erreur dans l'obtention du statut du serveur), une erreur serveur est renvoyé.

is_live()

Fonction appelé lors d'une requête sur la route "/is-live". Elle récupère toutes des dates des dernières mesures pour tous les capteurs, sauf celui de pollution, et détermine la date de la dernière mesure reçue. S'il n'y a aucune mesure dans la journée ou si elle date de plus

de 15 minutes, le statut est considéré comme n'étant pas en ligne. La fonction retourne un JSON comme suit :

```
{
  "islive": bool,
  "date_last_measure": "string",
  "location": "string"
}
```

"islive" est le statut de l'application, "date_last_measure" est la dernière date, celle sur laquelle il c'est basé pour déterminer le statut, et "location" est la localisation du pavillon renseigné dans le fichier "Configuration.py". La date la plus récente est enregistrée comme variable globale afin d'être utilisable par les autres fonctions.

get_measures_from_last_day_available(type)

Fonction appelé lors d'une requête sur la route "/get-data". Elle prend en paramètre le nom du type (string). Elle récupère toutes les valeurs de la dernière journée de mesure, pour le rayonnement solaire et la pollution seulement les valeurs extérieures et pour la température, la température globe, l'humidité et la vitesse du Vent les valeurs intérieure et extérieure. Toutes les valeurs contenant le commentaire test ne sont pas prises en compte. Les valeurs rayonnement solaires sont arrondies pour supprimer les décimales. Les mesures avec des données intérieure et extérieure sont arrondies à la minute et fusionné ensemble afin d'obtenir un tableau avec trois colonnes : timestamp, valeur intérieure et valeurs extérieures. Si un timestamp ne contient qu'une des deux valeurs, le champ est laissé vide. Dans ce cas, la valeur sera affiché en undefined sur l'interface. Les dernières valeurs intérieure et extérieure sont également récupérées. Pour les valeurs qui sont seulement présentes dans les mesures extérieures, la fonction retourne un JSON comme suit :

```
{
  "data": "CSV: timestamp,Extérieur",
  "date": "string",
  "type": "string",
  "last": "string"
}
```

et pour les valeurs intérieure et extérieure :

```
{
  "data": "CSV: timestamp,Intérieur,Extérieur",
  "date": "string",
  "type": "string",
  "lastint": "string",
  "lastext": "string"
}
```

"data" est le CSV contenant toutes les données, "date" est la date de données retournée, "type" est le type qui a été requêté, "lastint" est la valeur de la dernière mesuré en intérieur, "lastext" est la valeur de la dernière mesurée en extérieur et "last" est la dernière valeur mesurée dans le premier cas. Si la dernière valeur mesurée date de plus de 15 minutes, elle

est remplacée par "Offline".

get_data(start, end, type, interior=True, stats=False)

Cette fonction prend en paramètre une date de début, une date de fin, un type, une boolean si pour les données intérieures/extérieures et un boolean pour ne récupérer que les statistiques. Elle permet de requêter BBData pour récupérer pour la plage de temps donnée soit les statistiques, soit toutes les mesures des objets intérieur ou extérieur. Elle retourne les données sous forme de tableau de données.

connect()

Cette fonction permet d'établir la connexion avec BBData. Elle contient les informations de connexion et récupère les token de connexion nécessaire pour les autres requêtes.

setHeaders(headers)

Permet de stocker les entêtes contenant les informations de connexion (token) dans une variable globale. Prend en paramètre les entêtes à stocker.

getHeaders()

Permet de récupérer les entêtes précédemment stockés en variable globale et en renvoyer une copie.

Librairies

Les librairies python 3 utilisées sont les suivantes :

flask : Micorframework de serveur web très léger [4].

datetime : Types de base pour la date et l'heure, proviennent de la librairie standard.

pandas : Outil d'analyse et de manipulation de données rapide, puissant, flexible et facile à utiliser. [2]

numpy : Librairie de manipulation des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques [1].

requests : Permet d'envoyer des requêtes HTTP.

json : Permet d'encoder et de décoder des données en JSON, provient de la librairie standard.

dateutil : Librairie fournissant de puissantes extensions au module datetime standard.

5.4.3 Resultats

Le site est accessible à l'adresse : <http://demomi2.isc.heia-fr.ch/>.

On live

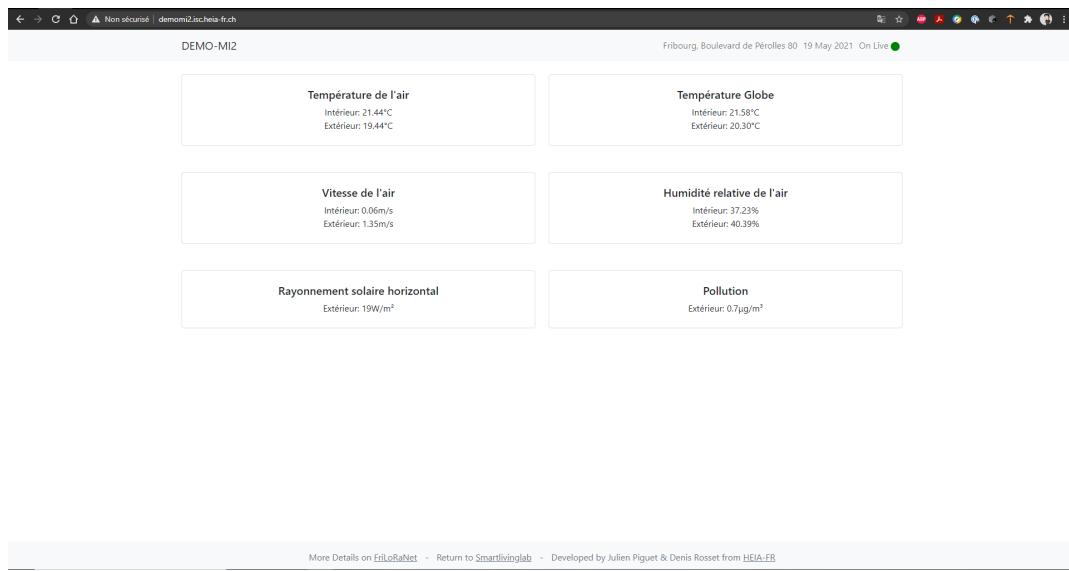


FIGURE 32 – WebSite Desktop On Live



FIGURE 33 – WebSite Desktop On Live Graphs



FIGURE 34 – WebSite Mobile On Live

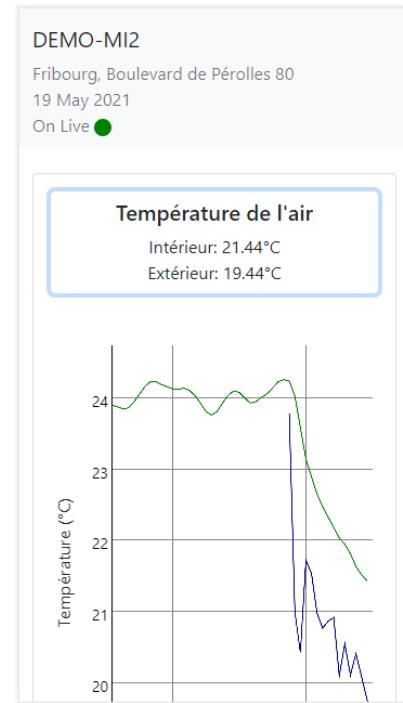


FIGURE 35 – WebSite Mobile On Live Graphs

Offline

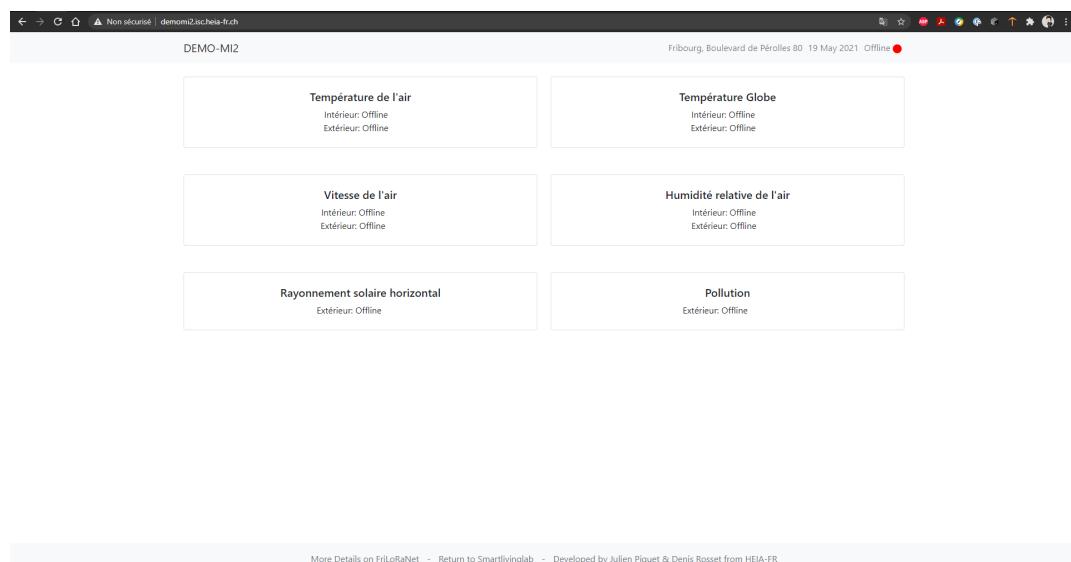


FIGURE 36 – WebSite Desktop Offline



FIGURE 37 – WebSite Desktop Offline Graphs

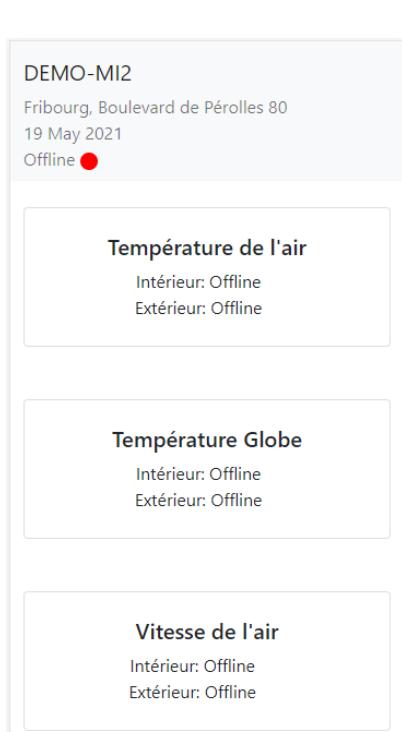


FIGURE 38 – WebSite Mobile Offline

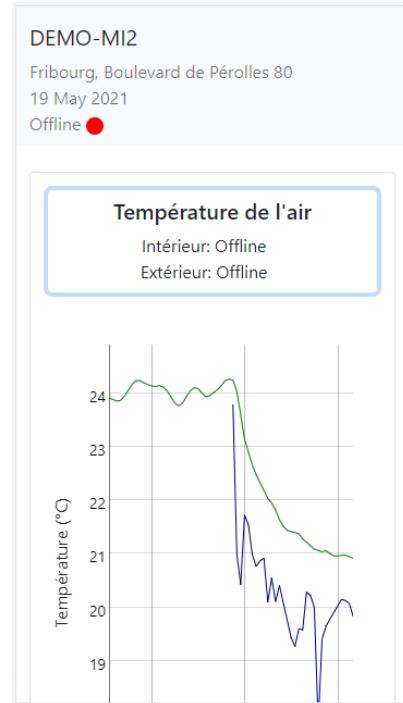


FIGURE 39 – WebSite Mobile Offline Graphs

6 Mise en place

6.1 Raspberry pi

6.1.1 Activer I2C

I2C doit être activé, pour celui il faut aller changer la configuration du Raspberry.

```
sudo raspi-config
```

Il faut aller dans "3 Interface Options".

Puis dans "P5 I2C"

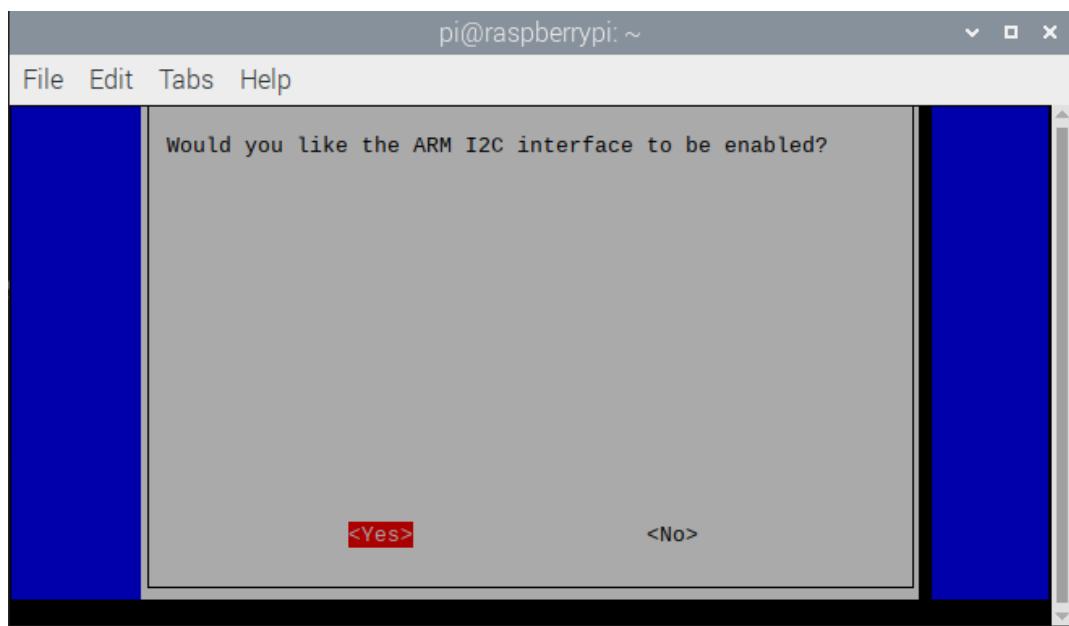


FIGURE 40 – Raspberry Enable I2C

Il faut sélectionner "<Yes>". Un message devrait apparaître comme quoi I2C est bien activé.

Il faut redémarrer :

```
sudo reboot now
```

Il doit être maintenant possible de voir les capteurs connectés, ainsi que leurs adresses avec la commande suivante :

```
sudo i2cdetect -y 1
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- - - - - - - - - - - - - - - - - - - - -
10: - - - - - - - - - - 18 - - - - - - - - - - - - - -
20: - - - - - - - - - - - - - - - - - - - - - - - - - -
30: - - - - - - - - - - - - - - - - - - - - - - - - - -
40: - - - - - 44 - - - - - - - - - - - - - - - - - - -
50: - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
60: -- - - - - - - - - - - - - - - - - - - -  
70: -- - - - - - - - - - - - - - - - - - - 77
```

Toutes les adresses des capteurs doivent apparaître ici. Une par capteur.

6.1.2 Installation de l'application

L'application Python se trouve dans le répertoire src du git (<https://gitlab.forge.hefr.ch/denis.rosset/ps6-pavillon-demo-mi2>).

Il faut premièrement copier les fichiers de l'application sur le Raspberry. Seul le répertoire "src" a besoin d'être copié. Exemple avec rsync :

```
rsync -avP src/. pi@192.168.137.165:src/.
```

Puis il faut installer toutes les dépendances de l'application

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

6.1.3 Crontab

Le lancement de l'application au démarrage du Raspberry se fait via Crontab.

Pour cela il faut ouvrir crontab :

```
crontab -e
```

Puis il faut y ajouter la ligne suivante à la fin. Le chemin et le fichier spécifié doivent bien correspondre au répertoire où l'application a été déployée.

```
@reboot cd /home/pi/src/ && /usr/bin/python3 /home/pi/src/demo_mi2.py
```

Si le programme a bien démarré, il est possible de le voir dans la liste de processus. Ainsi que dans les logs.

6.2 LoRa - Raspberry Pi

6.2.1 IoT LoRa Node pHat

Installation de la librairie nécessaire au fonctionnement de la carte IoT LoRa Node pHat.

Activation du port serial :

```
sudo raspi-config
```

Il faut aller dans "3 Interface Options".

Puis dans "P6 Serial Port"

Il faut ensuite sélectionner "<No>" :

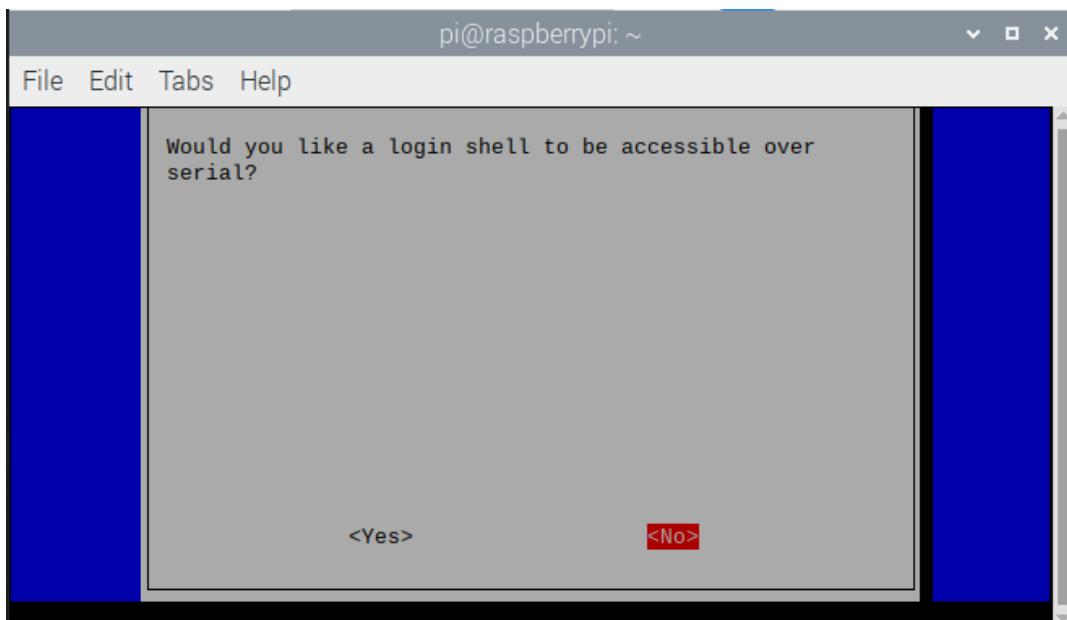


FIGURE 41 – Raspberry Enable Serial Shell

Puis "<Yes>" :

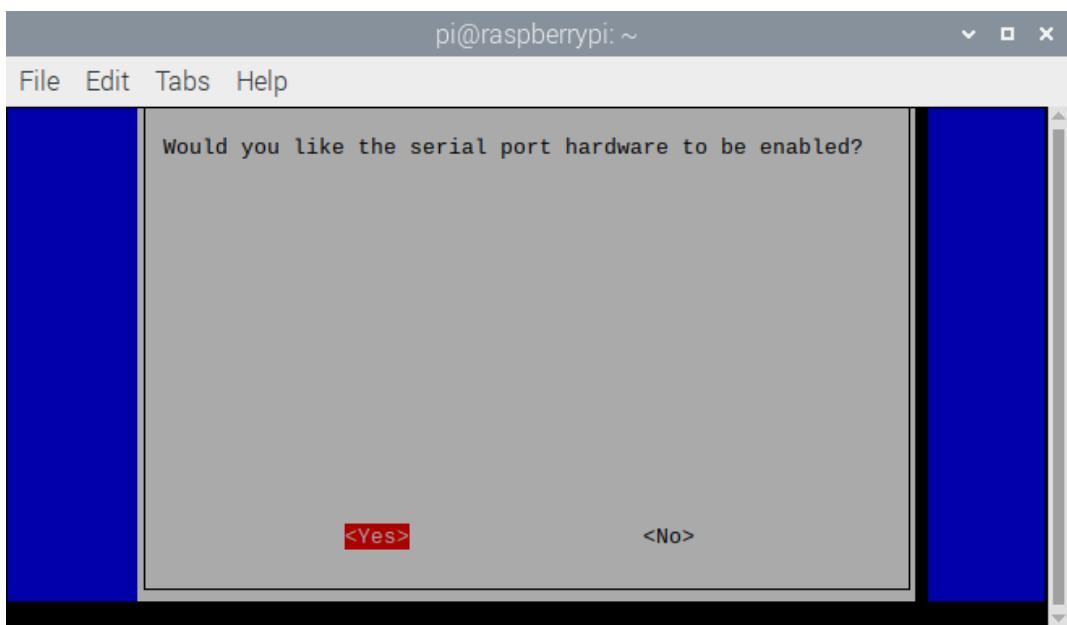


FIGURE 42 – Raspberry Enable Serial Port

Un message devrait apparaître avec le texte suivant :

```
The serial login shell is disabled  
The serial interface is enabled
```

Ouvrir le fichier boot config :

```
sudo nano /boot/config.txt
```

Et ajouter la ligne suivante à la fin :

```
dtoverlay=pi3-miniuart-bt
```

Puis redémarrer :

```
sudo reboot now
```

La librairie python suivante doit être installé :

```
sudo pip3 install rak811
```

La version de la librairie rak811 doit la version 3. Il est possible de la vérifier avec la commande suivante. Cela permet aussi de vérifier qu'elle est bien installée.

```
rak811 hard-reset
rak811 version
V3.0.0.14.H
```

Il est ensuite possible de récupérer le DevEui de la carte avec les commandes suivantes :

```
rak811v3 set-config lora:join_mode:0
rak811v3 get-config lora:status | grep DevEui
DevEui: 60C5A8FFFE7985DC
```

Après import de la librairie dans un programme python, il est maintenant possible de l'utiliser.

6.2.2 Lora/GPS HAT

Le Lora/GPS HAT utilise une librairie C afin d'être utilisé. Il faut donc installer ses composants et compiler le fichier exécutable utilisable par l'application Python. La librairie raspi-lmic est utilisée.

Installation

Installation des dépendances si manquante :

```
apt-get install build-essential
apt-get install git-core wget
```

Installation de la librairie bcm2835 :

```
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.69.tar.gz
tar zxvf bcm2835-1.69.tar.gz
cd bcm2835-1.69
./configure
make
sudo make check
sudo make install
sudo reboot now
```

Clone de raspi-lmic :

```
git clone https://github.com/pmanzoni/raspi-lmic.git
```

Il faut ensuite, dans le Makefile de notre application ("src/lora/Makefile"), spécifier le répertoire source de la librairie "/raspi-lmic/src". Exemple :

```
LMICBASE = /home/pi/src/lora/raspi-lmic/src
```

Il est ensuite possible de compiler "send_lora.cpp" :

```
cd src/lora/  
make
```

Le fichier exécutable résultant est celui qui sera utilisé par l'application pour envoyer des données via le Lora/GPS HAT. L'emplacement de cet exécutable doit être modifier dans le fichier de configuration "src/Configuration.py" sous "SEND_LORA_EXE_PATH".

Configuration

Pour chaque changement dans la configuration du fichier "send_lora.cpp" il devra être recompilé. Les principaux champs à modifier sont les adresses DEVEUI et APPKEY.

Exemple dans de DEVEUI et APPKEY :

```
// This should be in little endian format  
static const u1_t PROGMEM DEVEUI[8] =  
{ 0xb4, 0xef, 0x44, 0xf7, 0x57, 0xc4, 0xf9, 0xee };  
  
// This key should be in big endian format  
static const u1_t PROGMEM APPKEY[16] =  
{ 0x68, 0x24, 0xc9, 0xe7, 0x31, 0x1a, 0x2a, 0xc8, 0xd, 0xb1, 0x9b,  
0x06, 0x5f, 0x06, 0x18, 0x55 };
```

Il est important de contrôler que le DEVEUI est en little endian et que le APPKEY est en big endian.

Le DEVEUI doit changé sur chaque nouveau node. Il est possible le récupérer avec la librairie raspi-lmic :

```
cd raspi-lmic/examples/get_deveui/  
make  
../get_deveui
```

Le APPKEY est récupéré dans le ChirpStack server. Il doit être changé avant d'utiliser le node.

6.3 Installation du serveur

Le serveur web est installer sur une machine virtuelle Ubuntu 20

Il faut premièrement mettre à jour APT et installer Python 3 si ce n'est pas déjà fait.

```
sudo apt update
sudo apt install python3-pip python3-dev build-essential libssl-dev
libffi-dev python3-setuptools
```

L'environnement virtuel de python est également nécessaire.

```
sudo apt install python3-venv
```

Il faut ensuite cloner l'application stockée sur gitlab sur la machine virtuelle. Pour cela nous créons un nouveau répertoire à la racine nommé webapp. Une fois cloné dedans, l'application est dans le répertoire "/webapp/ps6-pavillon-demo-mi2/webapp/".

```
mkdir /webapp
cd /webapp
git clone https://gitlab.forge.hefr.ch/denis.rosset/ps6-pavillon-demo-mi2.git
cd ps6-pavillon-demo-mi2/webapp
```

Création d'un environnement virtuel.

```
python3 -m venv webappenv
```

Activation de l'environnement virtuel.

```
source webappenv/bin/activate
```

Installation de wheel.

```
(webappenv) # pip install wheel
```

Installation de Flask et de gunicorn.

```
(webappenv) # pip install gunicorn flask
```

Il est ensuite possible de tester si Flask fonctionne correctement, en démarrant "app.py". Cela a pour effet de démarrer le serveur sur le port 5000. Il peut être arrêté avec Ctrl+C.

```
(webappenv) # python app.py
```

Puis il est possible de tester Gunicorn. Cela a également pour effet de démarrer le serveur web sur le port 5000. Il peut également être arrêté avec Ctrl+C.

```
(webappenv) # gunicorn --bind 0.0.0.0:5000 serve:app
```

Flask et Gunicorn doivent démarrer correctement pour valider que le serveur web cloné est bien fonctionnel.

Nous pouvons ensuite désactiver l'environnement virtuel.

```
(webappenv) # deactivate
```

Un nouveau service doit être créé. Son nom est demomi2, ce qui correspond au nom du fichier ".service".

```
sudo nano /etc/systemd/system/demomi2.service
```

La configuration suivante doit être ajoutée dans le fichier. Il est important de contrôler que les chemins vers l'application web soient bien correct. Ainsi que le nom du socket, ici "webapp.sock". L'environnement utilisé pour l'application est l'environnement virtuel précédemment créé.

```
[Unit]
Description=Gunicorn instance to serve DEMO-MI2
After=network.target

[Service]
User=julien.piguet
Group=TIC
WorkingDirectory=/webapp/ps6-pavillon-demo-mi2/webapp
Environment="PATH=/webapp/ps6-pavillon-demo-mi2/webapp/webappenv/bin"
ExecStart=/webapp/ps6-pavillon-demo-mi2/webapp/webappenv/bin/gunicorn --workers 3 --bind un

[Install]
WantedBy=multi-user.target
```

Il faut ensuite activer le service (enable) afin qu'il se lance au démarrage. Puis il est possible de le démarrer directement avec le "start".

```
sudo systemctl enable demomi2
sudo systemctl start demomi2
```

Le statut du service est affichable avec la commande suivante. Si le service fonctionne correctement, son statut doit être "active".

```
sudo systemctl status demomi2
. demomi2.service - Gunicorn instance to serve DEMO-MI2
   Loaded: loaded (/etc/systemd/system/demomi2.service; enabled; vendor preset>
   Active: active (running) since Mon 2021-05-17 16:18:34 UTC; 1 day 22h ago
     Main PID: 814607 (gunicorn)
        Tasks: 7 (limit: 2280)
      Memory: 160.4M
        CGroup: /system.slice/demomi2.service
                  814607 /webapp/ps6-pavillon-demo-mi2/webapp/webapp/bin/python3 /weba>
                  814627 /webapp/ps6-pavillon-demo-mi2/webapp/webapp/bin/python3 /weba>
                  865949 /webapp/ps6-pavillon-demo-mi2/webapp/webapp/bin/python3 /weba>
                  865950 /webapp/ps6-pavillon-demo-mi2/webapp/webapp/bin/python3 /weba>
```

Il faut ensuite configurer nginx. La création d'un nouveau site se fait avec la commande suivante.

```
sudo nano /etc/nginx/sites-available/demomi2
```

La configuration du site est la suivante. Le port utilisé est le 80 et le nom du serveur correspond à l'URL et au nom de la machine virtuelle dans le DNS. Le site est à la racine (location '/') et le socket créé précédemment doit lui être spécifié dans le proxy_pass.

```
server {  
    listen 80;  
    server_name demomi2.isc.heia-fr.ch www.demomi2.isc.heia-fr.ch;  
  
    location / {  
        include proxy_params;  
        proxy_pass http://unix:/webapp/ps6-pavillon-demo-mi2/webapp/webapp.sock;  
    }  
}
```

Un lien du fichier doit être créé afin qu'il se trouve aussi dans les sites activés.

```
sudo ln -s /etc/nginx/sites-available/demomi2 /etc/nginx/sites-enabled
```

Il est possible de contrôler s'il n'y a pas d'erreur avec la commande suivante :

```
sudo nginx -t
```

Il faut ensuite redémarrer le service nginx.

```
sudo systemctl restart nginx
```

En cas de modification des fichiers sur le serveur web, pour l'actualiser il faut simplement redémarrer le service :

```
sudo service demomi2 restart
```

Le site est accessible à l'adresse <http://demomi2.isc.heia-fr.ch/> Le service démarre automatiquement lors du redémarrage de la machine virtuelle. Il devrait donc être accessible en tout temps. Si le site n'est pas accessible il faut vérifier que les deux services fonctionnent normalement (demomi2 et nginx), si ce n'est pas le cas les services retournent un message d'erreur.

6.4 Montage

6.4.1 Montage sur Bread Board

Le montage des deux capteurs ont d'abord été fait sur une bread board avec les capteurs directement branchés sur celle-ci. Le montage a été fait pour les deux hats LoRa.

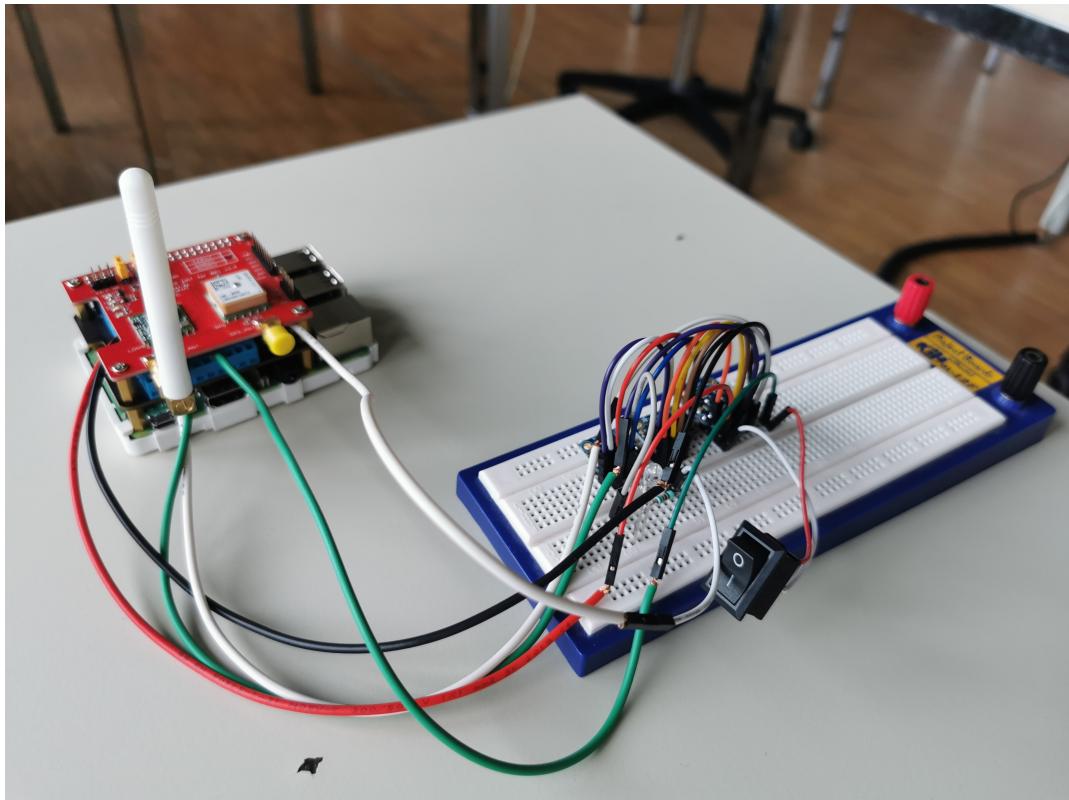


FIGURE 43 – Montage Lora/GPS HAT

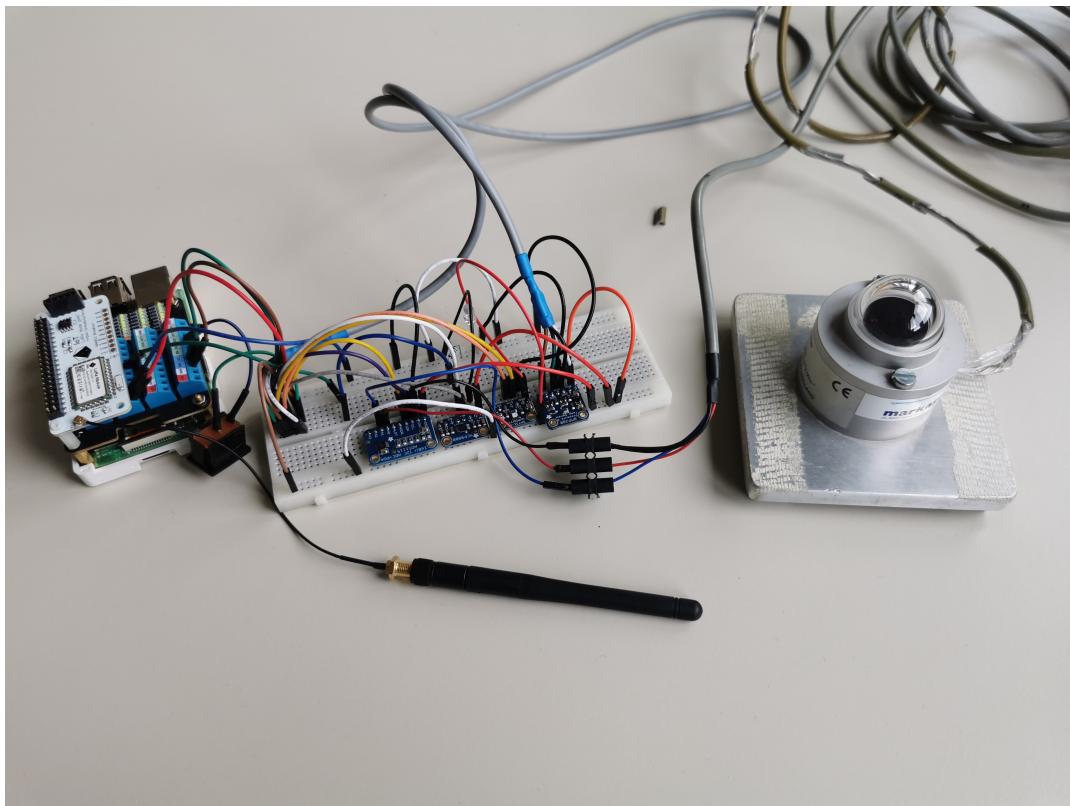


FIGURE 44 – Montage IoT LoRa Node pHAT

6.4.2 Montage Final

Le montage final est celui sur trépied qui sera déployé à l'intérieur et à l'extérieur du pavillon. Le montage du Raspberry Pi dans une boîte et les soudures des câbles reste encore à faire, mais les pièces doivent encore être manufacturées à ce stade du projet.



FIGURE 45 – Montage

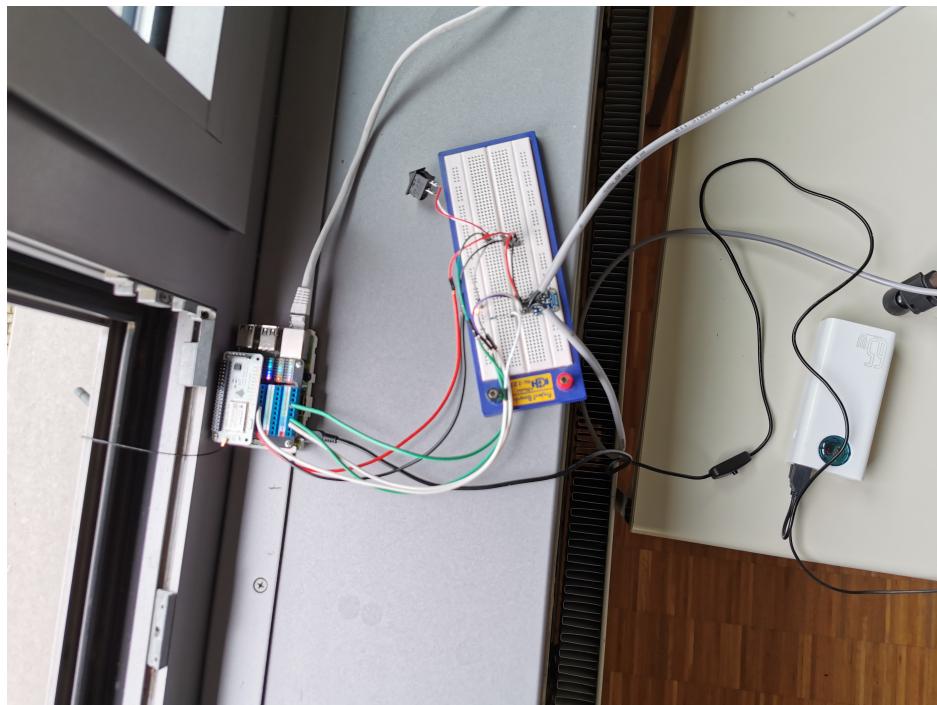


FIGURE 46 – Montage système embarqué

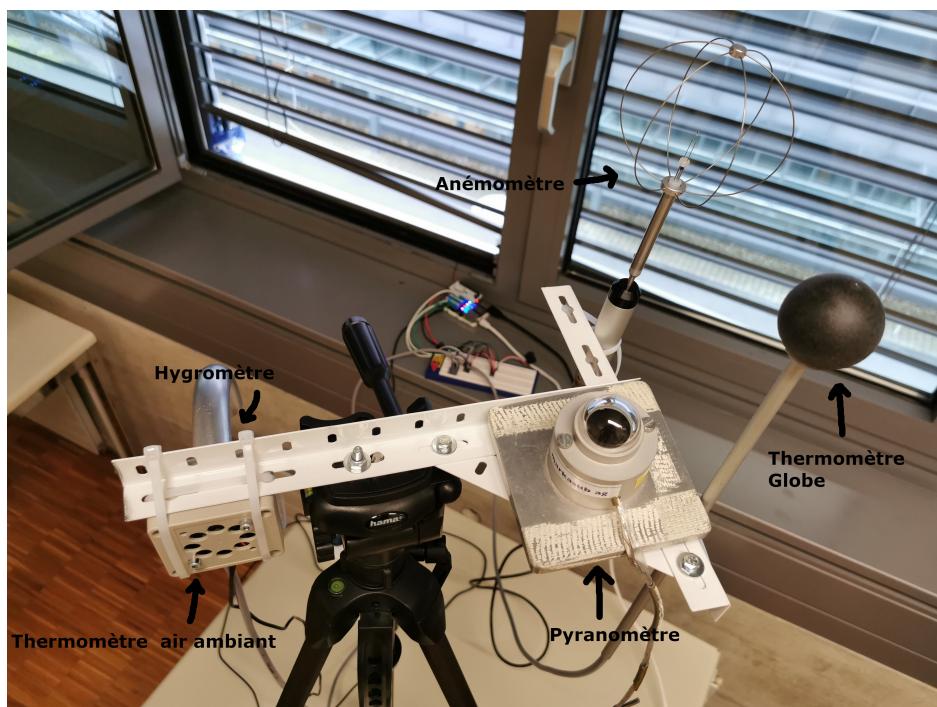


FIGURE 47 – Montage capteurs

7 Mode d'emploi

Cette section permet de décrire les étapes nécessaires au fonctionnement du système de mesures qui a été développé durant le projet.

7.1 Mode d'emploi du fonctionnement de l'application du Raspberry Pi

Cette sous-section permet d'expliquer comment utiliser l'application développée pour le Raspberry Pi 3

7.1.1 Connexion au système embarqué à distance

Pour pouvoir se connecter au Raspberry Pi à distance, il est nécessaire d'être dans le même sous-réseau que ce dernier. Pour faciliter sa présence dans le même sous réseau, le Raspberry pi a été configuré en mode point d'accès Wi-Fi. En se connectant réseau Wi-Fi, le Raspberry pi devient accessible en ssh à l'adresse **192.168.4.1**. Grâce à un client ssh, la connexion avec l'utilisateur **pi** et au mot de passe **raspberry** est possible.

7.1.2 Configuration du système embarqué

Le fichier de configuration du Raspberry Pi se trouve dans /src/Configuration.py il permet de régler plusieurs paramètres décrits ci-dessous :

- PYRANOMETER_DIVIDE_NUMBER : La valeur par laquelle l'intensité renvoyée par le pyranomètre doit être divisée pour trouver la valeur en watt par mètre carré
- ANEMOMETER_SERIAL_PORT : Le port série sur lequel l'anémomètre est branché.
- RASPBERRY_PI_ID : L'identifiant du Raspberry Pi. L'id 0 est le système de capture se trouvant à l'intérieur du pavillon DEMO-MI2 et l'id 1 est le système de capture du Raspberry pi se trouvant à l'extérieur du pavillon DEMO-MI2.
- RASPBERRY_PI_IS_PYTHON_HAT : Permet de définir quelle carte d'extension LoRa le raspberry pi utilise. Si la carte d'extension LoRa est **IoT LoRa Node pHAT**, la valeur doit être **True** si c'est la carte **LoRa-GPS-Hat**, la valeur doit être **False**
- DATA_FILE_BEGINNING : Le début du nom du fichier CSV dans lequel seront enregistrés les données mesurées.
- DATA_FILE_FIRST_LINE : La première ligne du fichier CSV dans lequel seront enregistrés les données mesurées. Cette ligne permet de définir quels seront les noms des colonnes.
- LOG_FILE_NAME : Le début du nom du fichier Logs dans lequel seront enregistrés les informations de journalisation de l'application.
- SECONDS_BETWEEN_MEASURES : Le temps entre chaque mesure des capteurs
- SECONDS_TO_DATA_LOG : Les valeurs de seconds lors desquelles l'enregistrement sera fait.
- MINUTES_TO_DATA_SEND : Les valeurs de minutes lors desquelles l'envoi sur le réseau LoRa sera fait.
- LED_INTERVAL_MEASURE : L'intervalle entre chaque clignotement de LED lors de la prise de mesures.
- LED_INTERVAL_SAVE : L'intervalle entre chaque clignotement de LED lors de la sauvegarde des mesures (la sauvegarde des mesures étant relativement rapide sur le Raspberry Pi, cette valeur n'est pas vraiment prise en compte, car le retour aux mesures est fait quasi-instantanément).

- **LED_INTERVAL_SEND** : L'intervalle entre chaque clignotement de LED lors de l'envoi sur le Réseau LoRa.
- **LED_INTERVAL_STOPPED** : L'intervalle entre chaque clignotement de LED lors de l'arrêt des mesures via le commutateur on/off.
- **RASPBERRY_PI_SWITCH_GPIO** : La GPIO sur laquelle le commutateur on/off est branché.
- **RASPBERRY_PI_LED_GPIO** : La GPIO sur laquelle est branchée la LED.
- **SEND_LORA_EXE_PATH** : Le chemin d'accès vers le fichier exécutable d'envoi sur LoRa de la carte d'extension **LoRa-GPS-Hat**.
- **LORA_MAX_RETRIES_WHEN_FAILED** : Le nombre d'essais que le système embarqué va effectuer avant d'abandonner l'envoi sur LoRa
- **LORA_TIME_BETWEEN_RETRY_SEC** : Le temps entre chaque essai d'envoi sur LoRa
- **LORA_TIME_BETWEEN_INTERIOR_AND_EXTERIOR_SEC** : Le delta de temps entre l'envoi sur LoRa des deux systèmes embarqués. Utilisé pour éviter que les systèmes embarqués n'envoient leurs données en même temps.

7.1.3 Démarrage de l'application du système embarqué

Une crontab [3] permet actuellement de démarrer l'application automatiquement au démarrage du Raspberry Pi afin que l'utilisateur n'ait pas besoin de devoir s'y connecter et la lancer à la main.

L'application peut cependant toujours être lancée à la main. Pour réaliser cette opération, il est nécessaire de se rendre dans le répertoire courant de l'utilisateur pi grâce à la commande **cd**.

Il est nécessaire que l'application ne soit pas déjà en train de fonctionner pour éviter que plusieurs mesures soient envoyées en même temps sur le réseau LoRa. Pour vérifier si l'application est déjà en train de fonctionner, la commande suivante permet de voir si un processus python portant le nom de l'application est déjà en cours d'exécution : `ps -aux | grep python`

Pour vérifier si l'application fonctionne déjà, observer si la LED clignote est aussi une solution.

Pour lancer l'application il suffit, depuis le répertoire de l'utilisateur pi (`/home/pi/`), de lancer la commande suivante : `python3 src/demo_mi2.py`

Pour lancer l'application en tâche de fond, la commande suivante est nécessaire : `python3 src/demo_mi2.py &` ;

Récupération des fichiers générés par l'application

Les fichiers de logs et de données créés sont disponibles dans leurs répertoires respectifs et sont enregistrés dès le lancement de l'application du système embarqué.

Fichier de logs

Les logs seront ensuite écrits dans le dossier de logs dans un fichier portant le nom de la date et l'heure à laquelle l'application a été lancée.

Fichier de données

Les données sous format CSV seront présentes dans le dossier data dans un fichier portant le nom de la date et l'heure à laquelle l'application a été lancée.

7.2 Accès au serveur web

Pour accéder au serveur web deux options sont disponibles. L'accès via l'URL <http://demomi2.isc.heia-fr.ch/> et l'accès via le QR-Code ci-dessous (Figure 48) :



FIGURE 48 – QR Code

8 Tests et validation

Ce chapitre permet de définir ce qui à été testé dans l'application ainsi que comment les tests ont été réalisés et quelle était la procédure de tests et de validation.

8.1 Tests fonctionnels

Les tests fonctionnels suivants ont été réalisés pour prouver que l'application réalise les tâches qui lui sont demandées.

Raspberry Pi			
N°	Description du test	Résultat attendu	Résultat obtenu
1	Le Raspberry Pi démarre sur batterie et fonctionne pendant au moins 6h	OK	OK
2	Le Raspberry Pi est accessible via son Wi-Fi à l'adresse 192.168.4.1	OK	OK
3	Les données mesurées par le Raspberry Pi sont cohérentes par rapport à celles de la météo	OK	OK*

* Le pyranomètre n'est pas toujours très fiable, il prend du temps à se mettre en place et à se stabiliser pour donner des valeurs précises.

Système			
N°	Description du test	Résultat attendu	Résultat obtenu
4	Le système est installé sur son support et les composants sont tous connectés entre eux	OK	OK
5	Le système communique avec tous les capteurs i2c	OK	OK
6	Le système se connecte au réseau LoRa	OK	OK
7	Le système envoie des données cohérentes sur le réseau LoRa	OK	OK
8	Lorsque le bouton est sur "off" la LED clignote une fois par 3 secondes	OK	OK
9	La LED clignote 10x par seconde lors des envois sur LoRa et 1x par seconde lors des mesures	OK	OK
10	Le système démarre automatiquement au démarrage du Raspberry Pi	OK	OK

Stockage et envoie des données via LoRa / BBData			
N°	Description du test	Résultat attendu	Résultat obtenu
11	La limite de données imposée par la ville de Fribourg n'est pas dépassée lors de l'envoi des données	OK	OK
12	Les données stockées sur BBData sont les mêmes que celles envoyées sur le réseau LoRa	OK	OK
13	Les données stockées sur le Raspberry Pi sont la moyenne de celles mesurées	OK	OK
14	Les données récupérées sur le serveur de l'application web sont la moyenne de celles mesurées par le Raspberry Pi	OK	OK
15	Le bouton arrête la capture des données tout en continuant des sauvegardes et les envois sur LoRa des données qui n'ont pas encore été sauvegardés/envoyées	OK	OK
16	Le système de Logs sauvegarde les informations importantes et permet de savoir si l'application a bien fonctionné	OK	OK

Application web			
N°	Description du test	Résultat attendu	Résultat obtenu
17	L'application web affiche Live lorsque des données ont été envoyées lors des 15 dernières minutes	OK	OK
18	L'application web est accessible grâce au QR-Code et à l'adresse web	OK	OK
19	L'application web affiche les dernières valeurs mesurées ainsi que les graphiques de la dernière journée de mesure	OK	OK
20	L'application web affiche, dans son entête, la date des mesures présentes dans les graphiques	OK	OK

Les tests ont permis de voir que toutes les éléments constituant le projet fonctionnaient ensemble. Ces tests permettent de valider les chapitres d'analyse, de conception et de réalisation. Une comparaison avec les objectifs du cahier des charges est disponible au point 10.1.1.

9 Améliorations

Ce chapitre décrit quelles sont les améliorations possibles pour des travaux futurs et comment le système complet pourrait être amélioré.

9.1 Amélioration du système embarqué

Le Raspberry Pi ainsi que tous ses capteurs forment un système complexe qui demande beaucoup de connaissances pour être géré. Des idées d'amélioration qui facilitent l'utilisation de ce système sont décrites dans les sous-chapitres suivants.

9.1.1 Écran d'affichage de l'état du système embarqué

Pour pouvoir mieux afficher l'état du système, les erreurs potentielles ainsi que des données en direct, un petit écran pourrait être ajouté. Il permettrait d'avoir accès aux informations citées précédemment directement sur le système embarqué afin de mieux comprendre son état. Des écrans tactiles de petite taille sont disponibles et relativement peu cher écran pour afficher la connexion des capteurs.

9.1.2 Gestion des erreurs

Afin de comprendre pourquoi le système embarqué ne fonctionne pas, les logs sont très fonctionnels, mais nécessitent d'avoir accès au système embarqué pour pouvoir être lus. La LED permet aussi d'avoir un retour visuel, mais son utilité est limitée. L'écran cité dans le chapitre précédent permettrait d'avoir un retour plus complet.

Une solution serait que le système embarqué tente davantage de résoudre les problèmes par lui-même en gérant mieux quels types d'exception sont levés pour pouvoir régler les problèmes. Par exemple si la mauvaise carte d'extension a été sélectionnée dans le fichier de configuration et que le Raspberry pi lance une exception, car il n'arrive pas à y accéder, il pourrait de lui-même tenter d'utiliser l'autre carte d'extension pour envoyer les données sur LoRa.

9.1.3 Pyranomètre

Le pyranomètre utilisé actuellement semble relativement peu stable. En effet il met beaucoup de temps à donner des valeurs cohérentes. Utiliser un pyranomètre plus récent ou de meilleure qualité permettrait d'avoir plus facilement des données claires et de confiance.

Utiliser un pyranomètre qui gère directement la conversion analogique vers numérique permettrait aussi de beaucoup simplifier la mesure de cette donnée et ainsi simplifier l'électronique.

9.2 Amélioration de l'application web

9.3 Amélioration de l'envoi de données LoRa

Les trames LoRa sont fonctionnels et permettent d'afficher les données mesurées. Cependant plusieurs fonctionnalités pourraient être ajoutées ou modifiées afin de rendre la gestion de LoRa plus simple et plus extensible.

9.3.1 Séparation des mesures

Actuellement les mesures sont toujours envoyées toutes en même temps. Si le réseau LoRa le permet, utiliser une trame différente pour chaque mesure permettrait d'envoyer des mesures même si l'un des capteurs est défectueux. Cela permettrait aussi d'éviter d'envoyer des données incohérentes lorsqu'un capteur n'est pas connecté au Raspberry pi.

9.3.2 Données de test

Les données de tests sont envoyées au démarrage de l'application pour vérifier que la connexion LoRa fonctionne. Actuellement des données incohérentes, mais accompagnées du commentaire "test" sont enregistrées dans la base de données BBData. Puis lorsque l'application web lit ces données, elle les enlève des graphiques, mais peut savoir que la connexion LoRa fonctionne.

Avoir un objet supplémentaire dans la base de données BBData permettrait d'avoir une possibilité de stocker l'état de l'application, ainsi que des codes d'erreur afin d'avoir un retour plus explicite de l'état du système embarqué.

10 Conclusion

Ce chapitre conclu le rapport avec des explications sur comment le projet s'est déroulé, l'atteinte des objectifs, le suivi du planning ainsi qu'une conclusion personnelle pour chaque étudiant.

10.1 Atteinte des objectifs

Dans ce chapitre, les résultats obtenus dans le chapitre 8.1 sont confrontés aux objectifs définis dans le cahier des charges (retrouvables dans les chapitres 1.4 et 1.5).

10.1.1 Objectifs principaux

Tous les objectifs principaux du projet ont été réalisés.

Le choix et la validation des capteurs ont permis d'obtenir toutes les mesures nécessaires au bon fonctionnement du système. L'envoi des données sur le réseau LoRa de la ville de Fribourg permet de sauvegarder les données sur la base de données BBData. La quantité de données envoyées n'excède pas la limite imposée par la ville de Fribourg et correspondent totalement aux besoins du projet. La réception des données et l'affichage sur une page web est fait grâce au serveur web mis en place sur une machine virtuelle fournie par l'école.

10.1.2 Objectifs secondaires

L'implémentation de la localisation de l'application grâce à un système GPS n'a pas été faite, car le temps à disposition pour réaliser l'application n'était pas suffisamment long pour permettre de gérer cette fonctionnalité. Cet objectif n'a donc pas été réalisé.

Le retour visuel du fonctionnement des capteurs sur le système embarqué a été en partie réalisé. La LED permet de comprendre si le système embarqué est en train d'effectuer des mesures. Si une erreur survient pendant la prise des mesures, la LED clignotera à une fréquence différente. Ce qui permet de savoir si tous les capteurs récoltent leurs données. Cependant cette information ne permet pas de définir l'état de chaque capteur. Une explication de comment avoir l'état de chaque capteur est donnée au point 9.1.1. Cet objectif a donc été réalisé que partiellement.

10.2 Suivi du planning

Le planning a été défini dans le cahier des charges au début du projet. Il n'a pas été beaucoup utilisé pour se situer sur l'avancement du projet, mais s'est révélé être relativement correct par rapport au temps qu'ont pris les différentes étapes à être réalisées.

10.3 Déroulement global du projet

Le projet dans son ensemble s'est bien déroulé, les composants électroniques ont été reçus relativement rapidement et la collaboration sur l'implémentation du stockage des données sur BBData s'est bien déroulée.

Les séances hebdomadaires de projet ont été utiles aux étudiants et ont permis de poser des questions aux superviseurs sur des sujets que les étudiants ne maîtrisaient pas suffisamment. Elles ont aussi permis de discuter avec le mandant du projet afin de mieux comprendre de quelle manière le projet devait être conçu.

Cependant fait de n'avoir reçu l'anémomètre que deux jours avant la fin du projet à nécessité de devoir implémenter rapidement son interface et à peut être, malgré les tests et la validation, menés à des erreurs d'implémentation qui n'ont pas encore été découvertes.

10.4 Collaboration entre les étudiants

La collaboration entre les étudiants s'est très bien passée. La distribution des tâches aux étudiants a été réalisée en tenant compte des préférences de chacun ce qui a permis de garder une grande motivation tout au long du projet. La mise en commun des réalisations des étudiants a été faite grâce à l'outil git et a permis d'être toujours efficace à chaque étape de fusion du projet.

10.5 Conclusions personnelles

Une conclusion personnelle de chaque étudiant ainsi qu'une conclusion commune ont été écrites dans ces chapitres

10.5.1 Conclusion de M. Rosset

Les systèmes embarqués étant l'une de mes passions, je me réjouissais de réaliser ce projet. Je suis aussi très intéressé par le domaine de la physique, ce qui m'a permis d'être motivé pour réaliser ce projet et d'apprécier apprendre beaucoup de nouveaux concepts techniques et physiques sur la manière dont doivent être mesurées des données météorologiques. Apprendre ce qu'est un pyranomètre ainsi que la manière dont est mesurée la vitesse du vent avec le capteur SWEMA-03 ont été très enrichissante.

10.5.2 Conclusion de M. Piguet

J'ai trouvé ce projet particulièrement intéressant. Principalement dû au fait ce c'est un projet concret, qui sera réellement mis en place dans la ville de Fribourg. Notre projet s'intègre dans un plus grand projet (DEMO-MI2), ce qui est très motivant pour fournir une solution fonctionnelle et utilisable. Mon intérêt pour les systèmes embarqués, système de communication et le développement a aussi grandement contribué à mon appréciation du projet. Il était très intéressant pour moi de découvrir de nouvelles technologies comme LoRa ou encore la récolte de données météorologique.

10.5.3 Conclusion commune

La réalisation globale du projet était très intéressante et à permis d'enrichir nos connaissances dans les domaines des systèmes embarqués, des capteurs de mesures, du protocole d'envoi de données sans fil LoRa, du stockage de données sur une base de données ainsi que de la réalisation d'une application web. Le panel de technologies à apprendre et à utiliser était conséquent dans ce projet, ce qui était très motivant.

Nous aimions remercier les superviseurs du projet, Monsieur Jacques Robadey et Monsieur Nicolas Schroeter pour leurs conseils et leur accompagnement tout au long du projet. Leurs expériences ont pu nous aider à mieux nous diriger durant le projet et leurs connaissances nous ont permis de trouver beaucoup de réponses aux interrogations que nous avons eues durant la réalisation de ce projet.

Nous aimions remercier le mandant du projet Monsieur Raphaël compagnon pour son suivi du projet, sa disponibilité et ses explications. Ses connaissances en informatique nous ont permis de pouvoir discuter de concept concret avec lui afin de comprendre ses besoins et ses attentes. Ses explications nous ont permis de pouvoir gérer la partie de prises de mesures efficacement et son souhait de simplicité nous a permis de nous concentrer sur la création d'une application fonctionnelle.

Nous aimions remercier Monsieur Vincent Robatel pour son aide pour la mise en place du stockage des données dans la base de données BBData, Monsieur Florian Helper pour ses explications et sa collaboration sur le fonctionnement des capteurs de pollutions, Monsieur Frédéric Montet pour ses explications sur le fonctionnement de la base de données BB-Data et Monsieur François Buntschu pour la mise en place de la machine virtuelle faisant fonctionner notre application web.

11 Déclaration d'honneur

Je, soussigné, Julien Piguet, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.



Je, soussigné, Denis Rosset, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.



12 Annexes

en annexe tous les fichiers du projet (code et documentation supplémentaire). Toutes les annexes sont accessibles depuis le projet gitlab : <https://gitlab.forge.hefr.ch/denis.rosset/ps6-pavillon-demo-mi2>

Les éléments disponibles en annexe sont :

- **Application Python Raspberry** : "/src/"
- **PVs de séances** : "/docs/PVs/*"
- **Maquettes Web** : "/docs/Maquettes/*"
- **Schémas** : "/docs/Schémas/*"
- **Planning** : "/docs/Planning/planning.png"
- **Rapport** : "/docs/Rapport.pdf"
- **Cahier des charges** : "/docs/Cahier_des_charges.pdf"
- **Documentation supplémentaire** : "/docs/Install/*"

13 Glossaire

Glossaire

BBData Base de données mise en place à la HEIA-FR pour permettre de stocker différentes valeurs mesurées par différents capteurs..

CSV Les fichiers CSV sont des fichiers de données qui se présentent sous la forme de tableau avec chaque colonne séparée par un caractère spécifique et chaque ligne séparée par un retour à la ligne.

DNS Serveur permettant de traduire une adresse web en adresse IP.

GPIO Pins d'entrée et de sortie du Raspberry Pi permettant de connecter des composants électronique génériques afin de les utiliser depuis une librairie du Raspberry Pi.

GPS Technologie permettant de localiser la position géographique d'un émetteur..

Gunicorn Gunicorn, est un serveur web HTTP WSGI écrit en Python et disponible pour Unix..

LoRa Acronyme pour Long Range. Protocole de télécommunication permettant la communication bas débit, via radio, d'émetteurs à faible consommation électrique..

NGINX NGINX est un logiciel libre de serveur Web.

SPI Acronyme pour Serial Peripheral Interface. SPI est un bus de données série synchrone.

breadboard Planche à trous où chaque trou est lié à d'autres pour pouvoir connecter des composants électroniques facilement sans avoir besoin de réaliser des soudures.

hat Carte d'extension se plaçant au dessus du Papsberry Pi et se connectant grâce aux GPIOs.

socket Structure logicielle à l'intérieur d'un réseau informatique qui sert de point d'extrémité pour envoyer et recevoir des données sur le réseau..

timestamp Horodatage. Enregistrement numérique du moment d'occurrence d'un évènement particulier.

14 Références

Références

- [1] *Librairie de calcul scientifique python.* URL : <https://numpy.org/> (visité le 20/05/2021).
- [2] *Pandas, a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.* URL : <https://pandas.pydata.org/> (visité le 20/05/2021).
- [3] *Programmer des tâches avec CRON.* URL : <https://doc.ubuntu-fr.org/cron> (visité le 20/05/2021).
- [4] *Python Flask's documentation.* URL : <https://flask.palletsprojects.com/en/2.0.x/> (visité le 20/05/2021).
- [5] *Raspberry Pi Battery Runtime Calculator.* URL : <https://brennerm.github.io/pi-power-calc/#/runtime> (visité le 20/05/2021).
- [6] RASPI.TV. *How much power does the Pi4B use ? Power Measurements.* 2019. URL : <https://raspi.tv/2019/how-much-power-does-the-pi4b-use-power-measurements> (visité le 20/05/2021).

15 Figures

Table des figures

1	Esquisse du pavillon DEMO-MI2	1
2	Planning du projet	4
3	SHT31-D (source : https://www.adafruit.com/product/2857)	6
4	MCP9808 (source : https://www.adafruit.com/product/1782)	6
5	BMP280 (source : https://www.adafruit.com/product/2651)	7
6	BMP280 (source : https://thermolab.ch/produit/swema-03-anemometre-omnidirectionnel-vitesse-air/)	7
7	BMP280 (source : https://www.campbellsci.com/cm3)	8
8	MCP9808 (source : https://www.adafruit.com/product/1085)	8
9	Raspberry Pi 3b+ (source : https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/)	9
10	Baseus Bank (source : http://www.baseus.com/product-683.html)	9
11	Schéma LoRa Architecture	11
12	Lora/GPS HAT	13
13	IoT LoRa Node pHAT	14
14	Schémal global du système	17
15	Raspberry connectique	18
16	Maquette window	19
17	Maquette mobile	20
18	Diagramme d'activité de l'application fonctionnant sur le Raspberry Pi	21
19	Schéma LoRa DEMO-MI2	23
20	Schéma Serveur Web	26
21	Organisation des différents composants du Raspberry Pi	31
22	ChirpStack Device-Profile	34
23	ChirpStack Device-Profile OTAA	34
24	ChirpStack Codec	34
25	ChirpStack Application	35
26	ChirpStack Application Devices Liste	35
27	ChirpStack Application Configuration	36
28	ChirpStack Intégration	36
29	ChirpStack Intégration HTTP	37
30	ChirpStack Device Create	37
31	ChirpStack Device OTAA	38
32	WebSite Desktop On Live	42
33	WebSite Desktop On Live Graphs	42
34	WebSite Mobile On Live	43
35	WebSite Mobile On Live Graphs	43
36	WebSite Desktop Offline	43
37	WebSite Desktop Offline Graphs	44
38	WebSite Mobile Offline	44
39	WebSite Mobile Offline Graphs	44
40	Raspberry Enable I2C	45
41	Raspberry Enable Serial Shell	47

42	Raspberry Enable Serial Port	47
43	Montage Lora/GPS HAT	53
44	Montage IoT LoRa Node pHAT	54
45	Montage	55
46	Montage système embarqué	56
47	Montage capteurs	56
48	QR Code	59