



HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

PROJET IOT

Mesure de la qualité de l'air dans l'école

Classe IOT 2018

Supervisée par
Hervé DEDIEU et Benoît LOVIS

Haute École d'Ingénierie et de Gestion du Canton de Vaud

Yverdon-les-Bains, VD Suisse

12 juin 2018

Table des matières

1 Analyse sécuritaire	6
1.1 Introduction	6
1.2 Description du système	7
1.3 Sources de menaces	16
1.4 Scénarios d'attaque	17
1.5 Contre-mesures	27
1.6 Conclusion sécuritaire	36
2 Partie firmware	38
2.1 Technologies utilisées	38
2.2 Spécificités	41
2.3 Déploiement	43
2.4 Tests effectués	43
2.5 Conclusion	44
2.6 Sources	44
3 Partie infrastructure	45
3.1 Introduction	45
3.2 Technologies utilisées	47
3.3 Spécificités	51
3.4 Déploiement	52
3.5 Conclusion	57
3.6 Documentation supplémentaire	58
4 Partie back-end	59
4.1 Introduction	59
4.2 Technologies utilisées	59
4.3 Spécificités	59
4.4 Déploiement	63
4.5 Conclusion	63
4.6 Documentation supplémentaire	63
5 Partie front-end	64

5.1	Introduction	64
5.2	Technologies utilisées	64
5.3	Spécificités	65
5.4	Déploiement	69
5.5	Architecture	70
5.6	Problèmes rencontrés et difficultés à prendre en compte	70
5.7	Conclusion	70
5.8	Liens utiles	70

Table des figures

1.1	Schéma de l'architecture du projet	7
1.2	NUCLEO-F401RE	8
1.3	Arduino Uno Click Shield	9
1.4	Ambient 2 Click BME680	9
1.5	LoRa Click	10
1.6	Montage complet du capteur	10
1.7	Raspberry Pi Model B	11
1.8	iC880A-SPI - LoRaWAN Concentrator 868 MHz	12
1.9	Diagramme des flux	15
1.10	Principes essentiels de la sécurité informatique	21
1.11	Protection anti-modification par maillage	32
2.1	Détails du capteur BME680	38
2.2	Arduino uno click shield	39
2.3	Espruino web IDE 1	40
2.4	Espruino web IDE 2	41
2.5	Entêtes arduino	44
3.1	Architecture LoRaWAN simplifiée	45
3.2	Architecture détaillée	46
3.3	Architecture du network-server	47
3.4	Application ajoutée	49
3.5	REST API pour le app server	50
3.6	Branchemet de la Raspberry Pi en série	53
3.7	Trafic entrant dans la gateway	57
3.8	Communication entre device et LoRa server	57
4.1	Infrastructure du back-end	59
5.1	Aperçu de la page de connexion	66
5.2	Aperçu de la page carte	67
5.3	Aperçu de la page d'un capteur avec les jauge	68
5.4	Aperçu de la page d'un capteur avec les graphiques classiques	69

Chapitre 1 : Analyse sécuritaire

1.1 Introduction

Ce chapitre sur la sécurité a pour but de décrire d'un point de vue sécuritaire le projet élaboré dans le cadre du cours IoT dispensé à la HEIG-VD. Ce dernier est une plateforme de collecte de données environnementales issues de plusieurs capteurs. Dans ce document, les exigences du projet en général seront décrites ainsi que les biens nécessitant une protection contre tout type de menaces ou tout scénarios d'attaque ainsi que les contre-mesures associées. De plus, la liste de tous les composants hardware utilisés pour ce projet est donnée. Toutes les technologies utilisées pour ce projet ont été prises en compte.

1.1.1 Equipes

Lors de ce projet, nous avons créé 5 groupes distincts afin de répartir les différentes compétences et ainsi répartir la charge de travail pour chacun de ces derniers. Chacun des groupes avaient un répondant pour les autres groupes (représentés en gras ci-dessous).

Groupe	Etudiant
Frontend	Aurélie Lévy Tony Clavien Mathias Gilson
Backend	Ludovic Delafontaine Guillaume Milani Sathiya Kirushnpillai Mathieu Monteverde Nicolas Rod
Sécurité	Lara Chauffoureaux Matthieu Chatelan Thibaud Besseau Emmanuel Schmid
Firmware	David Truan Théo Gallandat Gaëtan Othenin-Girard Marie Lemdjo Ludovic Richard
Infrastructure	Julien Brêchet Yosra Harbaoui Guillaume Semeels Adrien Marco Ali Miladi Dany Tchente

1.2 Description du système

1.2.1 Objectifs du système

Le but de ce système est la collecte de données depuis plusieurs capteurs répartis sur le site de Cheaseaux. Toutes les informations seront consultables sur un frontend accessible par les visiteurs authentifiés sur un compte public.

Les différents capteurs (voir sous-section 1.2.3) se chargent de récolter des informations relatifs à leur environnement et les transmettent à une gateway dont le but est de collecter ces informations issues des différentes sources. Un bridge assure le lien entre le réseau LoRa et le réseau internet standard. Toutes les informations récoltées sont finalement transmises à un serveur d'applications sur lequel tourne le backend ainsi que le frontend.

Le schéma ci-dessous illustre cette architecture :

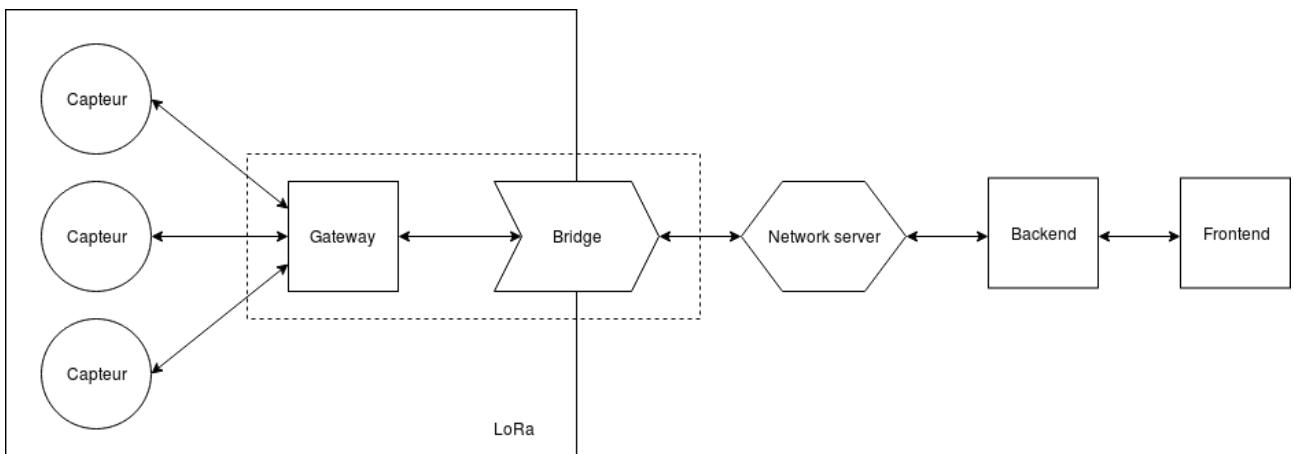


FIGURE 1.1 – Schéma de l'architecture du projet

1.2.2 Exigences de l'application

Pour la sécurité, nous avons interprété les exigences suivantes :

- L'accès au backend ainsi que l'accès au frontend ne doit être possible que pour les personnes autorisées et authentifiées à l'aide d'un compte que ce soit un compte utilisateur ou administrateur.
- Un utilisateur classique ne doit pas pouvoir accéder aux fonctionnalités réservées aux administrateurs.
- Les données transmises par les capteurs ne doivent pas être lisibles sur le réseau.

1.2.3 Éléments du système

Afin de rendre ce système fonctionnel, plusieurs composants hardware ainsi que software doivent être utilisés. Les sous-sections suivantes représentent les différents modules hardware utilisés ainsi que les parties software développées pour ce projet. Pour chacun des modules hardware, une brève description est donnée (fournie par les fabricants).

Capteurs

La carte STM32 Nucleo (Figure 1.2) offre aux utilisateurs un moyen abordable et flexible d'essayer de nouveaux concepts et de construire des prototypes avec le microcontrôleur STM32, en choisissant parmi les différentes combinaisons de performances, de consommation d'énergie et de fonctionnalités. Pour les cartes compatibles, le SMPS réduit considérablement la consommation d'énergie en mode Run.

Cette carte sera utilisée comme base pour tous les capteurs déployés dans le terrain.

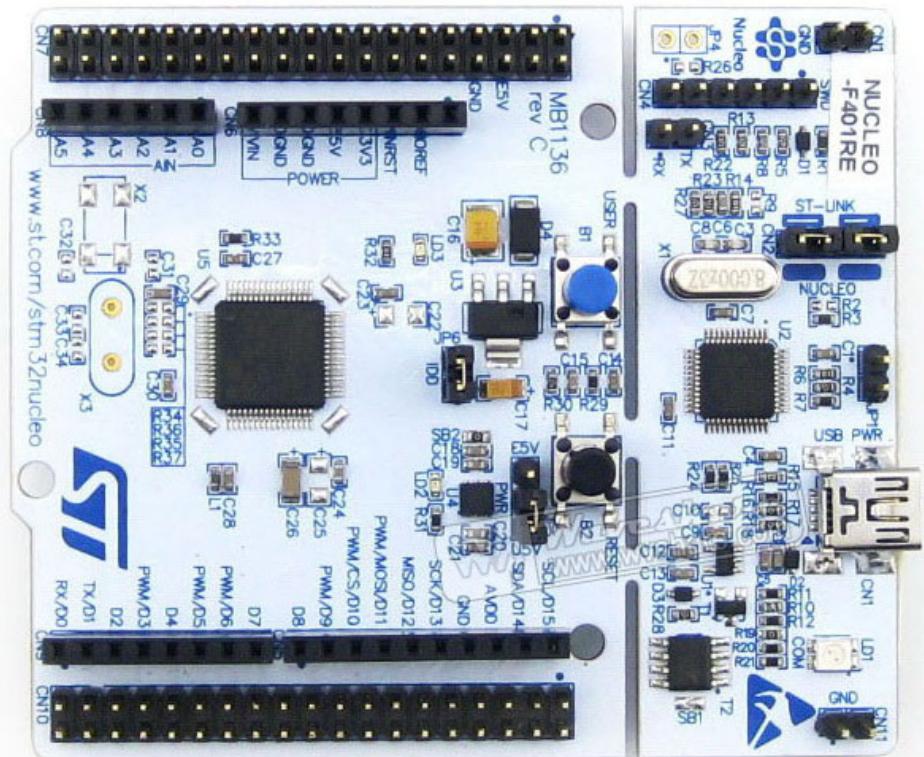


FIGURE 1.2 – NUCLEO-F401RE

Le shield utilisé pour ce projet (Figure 1.3) rend un Arduino compatible avec plus de 75 capteurs de type clic. C'est un shield simple avec deux prises hôte mikroBUS™ d'un côté et un connecteur Arduino sur le côté opposé.

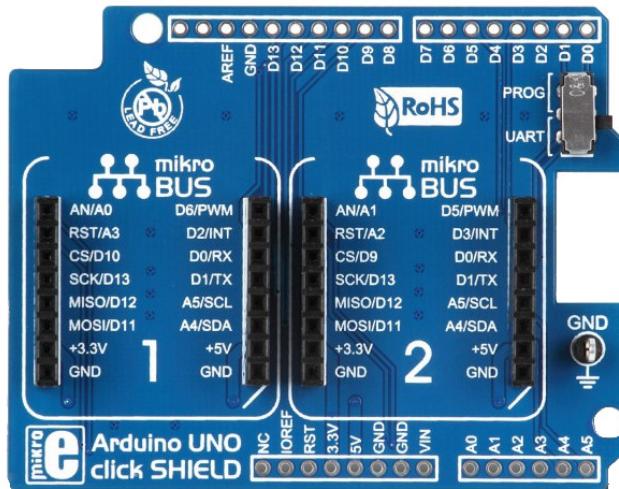


FIGURE 1.3 – Arduino Uno Click Shield

Le module *Environment click* (Figure 1.4) mesure la température, l'humidité relative, la pression et les COV (composés organiques volatils gazeux). Le clic intègre le capteur environnemental BME680 de Bosch. L'Environnement Click est conçu pour fonctionner sur une alimentation de 3,3 V. Il communique avec le microcontrôleur cible via l'interface SPI ou I2C.

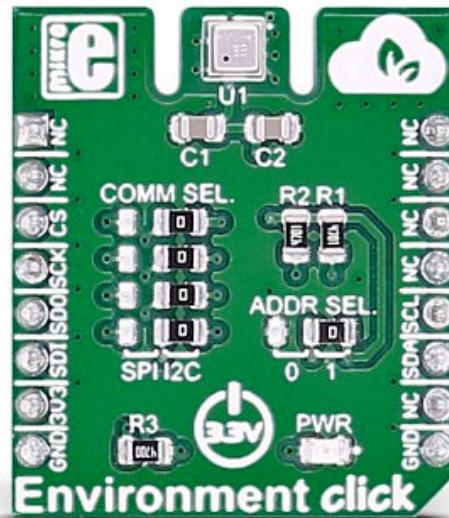


FIGURE 1.4 – Ambient 2 Click BME680

LoRaWAN™ ou Low Area Wide Network est une technologie sans fil développée pour permettre des communications à bas débit sur de longues distances, principalement pour les applications IoT et les capteurs.

Le module émetteur-récepteur LoRa longue portée RN2483 de Microchip (Figure 1.5) est une solution facile à utiliser et à faible consommation d'énergie pour la transmission de données sans fil à longue portée.

Le module RN2483 a une portée spécifiée > 15 km dans les zones rurales et suburbaines, et > 5 km dans les zones urbaines.

Une pile de protocole LoRaWAN™ classe A est intégrée (périphériques finaux bidirectionnels), ainsi qu'une interface de commande ASCII accessible via UART. La sensibilité élevée du récepteur peut descendre à -148 dBm.



FIGURE 1.5 – LoRa Click

Ci-dessous, une photo du montage complet du capteur avec tous les modules attachés :



FIGURE 1.6 – Montage complet du capteur

Gateway

Le Raspberry Pi 2 Model B est le Raspberry Pi de deuxième génération. Il a remplacé l'original Raspberry Pi 1 Model B+ en février 2015. Ce dernier est utilisé comme base pour la gateway.

Le Raspberry Pi 2 comporte les caractéristiques suivantes :

- 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- 100 Base Ethernet
- 4 Ports USB
- 40 pins GPIO
- Port HDMI
- Jack audio 3.5mm et sortie vidéo composite
- Interface Caméra (CSI)
- Interface d'affichage (DSI)
- Slot pour Micro SD
- VideoCore IV 3D coeur graphique

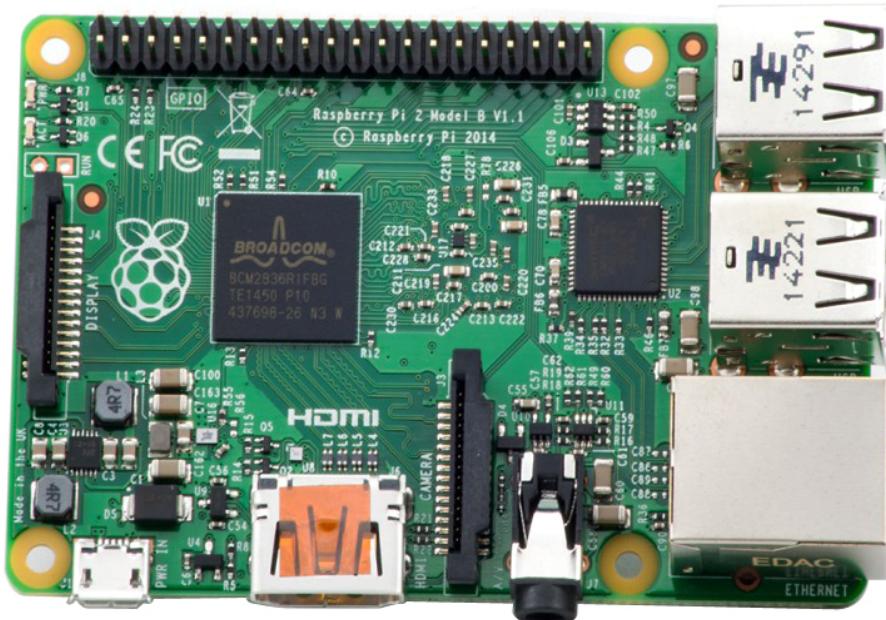


FIGURE 1.7 – Raspberry Pi Model B

Le module iC880A-SPI est capable de recevoir des paquets de différents appareils envoyés avec différents facteurs d'étalement sur jusqu'à 8 canaux en parallèle. En raison du fait que la combinaison des facteurs d'étalement et des largeurs de bande du signal donne des débits de données différents, l'utilisation de "Dynamic Data-Rate Adaption" devient possible. Cela signifie que les nœuds LoRa® à grande distance du concentrateur doivent utiliser des facteurs d'étalement plus élevés et donc avoir un débit de données plus faible.

Les nœuds LoRa qui sont plus proches du concentrateur peuvent utiliser des facteurs d'étalement plus faibles et peuvent donc augmenter leur débit de données. Cela permet de construire des réseaux étoiles ou multi-étoiles faciles à gérer sans besoin de routeurs ou de répéteurs.

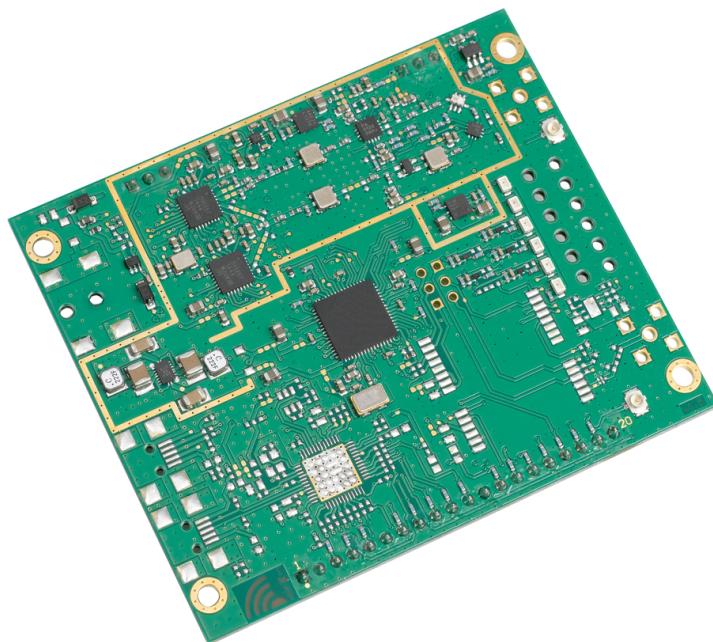


FIGURE 1.8 – iC880A-SPI - LoRaWAN Concentrator 868 MHz

Afin de connecter ce module au Raspberry, un shield tel que le *RPi to iC880a interface* est nécessaire.

Bridge

Dans notre cas, le bridge fait partie intégrante de la gateway. Sur le schéma dans la sous-section 1.2.1, ce dernier est séparé de la gateway afin de bien représenter le passage de l'utilisation du protocole *LoRa* vers le protocole *MQTT*. Plus d'informations sont disponibles sur le github suivant : <https://github.com/heig-vd-iot2018/infrastructure>

Network Server

Ce dernier est divisé en trois parties différentes : Routeur, Broker et Network server. Plus d'informations sont disponibles sur le github suivant : <https://github.com/heig-vd-iot2018/infrastructure>

Backend

Le backend est développé en Node.js avec le framework Express. Ce dernier reçoit des informations depuis le network server et traite ces informations avant de pouvoir fournir ces dernières au frontend. Plus d'informations sont disponibles sur le github suivant : <https://github.com/heig-vd-iot2018/back-end>

Frontend

Le frontend consiste en la mise en place d'une interface utilisateur permettant de voir les différentes valeurs fournies par les capteurs. Ce dernier a été développé à l'aide des technologies suivantes : **React**, **D3** et **Mapbox.js**. Plus d'informations sont disponibles sur le github suivant : <https://github.com/heig-vd-iot2018/front-end>

1.2.4 Biens nécessitants une protections

Les biens principaux à sécuriser sont toutes les données qui peuvent contenir des informations métier ou clients. Par exemple, les données transmises par les capteurs ne doivent pas être lisibles par des personnes non-autorisées au cours de leur trajet sur les différents réseaux empruntés.

Les biens concernés par cette protection sont les suivants :

- Les données transmises par les capteurs
 - Identité
 - Géo-localisation
 - Données environnementales
- Les données utilisateurs
 - Adresses e-mails¹
 - Rôles des utilisateurs²
 - Mots de passe³
- Le fonctionnement de l'application
- Les appareils et éléments physiques qui font tourner l'architecture

Ces biens doivent être protégés à tous les niveaux et à tous les endroits où elles sont susceptibles d'apparaître (e.g. des capteurs à la gateway mais aussi de la gateway au serveur d'application).

1.2.5 Périmètre de sécurisation

Dans ce projet, la sécurité doit être analysée sur chacun des éléments qui composent l'architecture mais aussi sur les liens entre eux. Il n'y a aucune zone considérée comme "sûre de base", il faut donc penser à tous les éléments suivants :

- Sécurisation des connexions aux éléments physiques.
- Sécurisation des trames à l'intérieur du protocole LoRa.
- Sécurisation des accès au back-end et au front-end.
- Sécurisation de toutes les communications MQTT.
- Sécurisation des éléments hébergeant les différents éléments de l'infrastructure.
- Sécurisation et gestion des versions pour les technologies, langages et bibliothèques utilisées.

1. Afin d'éviter toute réutilisation pour du phishing ou du spamming de masse.

2. Pour éviter les vols de session ciblés.

3. Les raisons sont évidentes et de plus ces mots de passe pourraient être ajoutés à des listes de brute-force.

1.2.6 Diagramme des flux

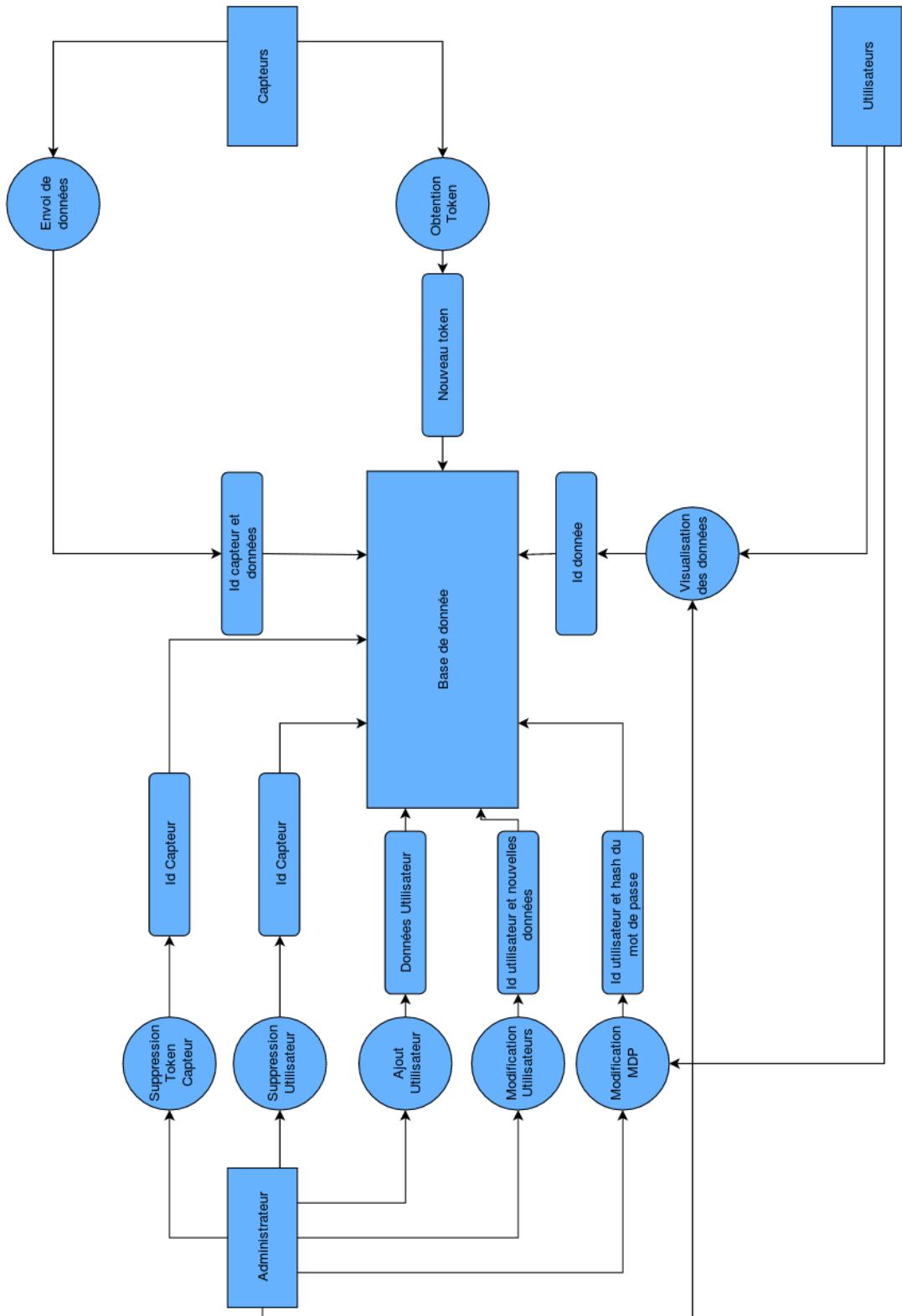


FIGURE 1.9 – Diagramme des flux

1.3 Sources de menaces

Différentes sources de menaces existent autour des applications web mais aussi autour du monde de l'*Internet des objets*. Dans le cadre de notre projet nous retrouvons les catégories de menaces décrites ci-dessous :

- **Étudiants kleptomanes**

Motivation : Gagner un capteur, une gateway, une Raspberry Pi gratuite
Cible : Le matériel
Probabilité : Haute

- **Hacker, script-kiddies**

Motivation : S'amuser, s'entraîner, le faire pour la reconnaissance
Cible : Tout ce qui peut être visé et qui entre dans ses compétences
Probabilité : Moyenne

- **Éventuels concurrents**

Motivation : Récupération des particularités de l'application (espionnage industriel)
Cible : Le fonctionnement de l'application
Probabilité : Moyenne

- **Cybercriminels**

Motivation : Récupérer des adresses mails (spamming) et mots de passe, se servir de l'application web comme passerelle vers son site malveillant ou pour répandre un virus
Cible : Les données des utilisateurs, l'accès à la partie web
Probabilité : Faible

- **Organisation étatique**

Motivation : Récolter des données, espionner
Cible : Toute l'application
Probabilité : Presque nulle

1.4 Scénarios d'attaque

Dans les sections ci-dessous, différents scénarios d'attaque sont listés et analysé en détails. Pour chacun d'eux, les failles qui existaient dans le projet ont été listées et analysées. Toutes les contre-mesures applicables pour les corriger sont détaillées dans la section 1.5. Dans la conclusion, vous retrouverez toutes les listes des failles existantes, absente, corrigées ou exploitable.

Un scénario d'attaque correspond à une marche à suivre qu'un attaquant souhaitant attaquer le système suivrait pour arriver à ses fins. Celui-ci est généralement élaboré par un ingénieur sécurité s'étant mis à la place d'un attaquant. Cet exercice permet de mieux comprendre les vecteurs d'attaque, l'impact, les motivations et les potentielles cibles d'un individu malveillant. Ces scénarios ne représentent pas les failles sécuritaires à proprement parler, mais les motivations générales qu'un attaquant peut avoir pour attaquer le système.

Chacun des chapitres ci-dessous reprend donc un scénario d'attaque ainsi que les failles qui lui sont associées. Avec chaque scénario, se trouve un résumé des enjeux et la liste des failles ainsi que la manière dont elles peuvent être exploitées.

1.4.1 Vol d'informations dans la base de données

Ce scénario concerne principalement les groupes front-end et back-end.

Comme dans la plupart des applications web et des projets de cette envergure, le groupe *back-end* a mis en place une base de données. Celle-ci est évidemment une cible privilégiée pour les attaquants car elle contient toutes les informations métiers et celles nécessaires à la plateforme web.

Scénario : Un hacker décidé à récupérer des données pour son nouveau dictionnaire de mots de passe suisses romands, connaît l'existence d'un jeune projet à la HEIG-VD. En accédant au site web, il découvre une vulnérabilité lui permettant de récupérer les données présentes dans la base de données. Il peut donc compléter son dictionnaire qu'il utilisera à des fins malveillantes.

Impact : Moyen

Source de menace : Concurrents, script kiddies, hackers et cybercriminels

Motivation :
La gloire (script kiddies, hackers)
L'argent (cybercriminels)
L'espionnage industriel (concurrents)

Cible : Données utilisateurs et données métier

Contrôles :
Caché le contenu de la base de données
Sécuriser son accès

Injections NoSQL

Quand on parle de failles touchant les bases de données, on pense forcément au numéro 1 du top 10 de l'OWASP : les injections SQL. Dans notre application, la base de données déployée n'est pas une base de données SQL mais MongoDB. Néanmoins, après quelques recherches sur Google, on apprend vite que cela ne change rien et que de telles failles existent aussi en NoSQL. L'existence d'une telle faille dans le système mènerait au scénario décrit ci-dessus du vol de données. Voici le lien officiel de l'OWASP expliquant en gros le fonctionnement et la détections de telles injections dans un base de données MongoDB :

https://www.owasp.org/index.php/Testing_for_NoSQL_injection

Une courte explication des contre-mesures envisageables pour une telle faille est donnée dans la sous-section 1.5.1

Lisibilité du contenu sensible

Ceci ne reprend pas vraiment une faille à proprement parler mais plutôt une précaution qu'il faut prendre lors du développement d'une application web. Imaginons que nous ayons corrigés les failles permettant les injections NoSQL décrites ci-dessus, mais qu'une faille zero-day soit découverte sur les librairies utilisées par notre application. Malgré nos protections mises en place, un attaquant pourra donc accéder au contenu de la base de données et récupérer les mots de passe et autres données confidentielles.

Prévenir vaut mieux que guérir et donc chiffrer et saler les mots de passe dans la base de données s'avère nécessaire. Cela permet d'éviter que les sessions utilisateurs soient compromises sans aucune attaque par dictionnaire ou par brute-force.

Dans les contremesures, une explications plus poussée du salage et du hachage sont données. Voici néanmoins les conseils de l'OWASP à ce propos : [https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet](https://www.owasp.org/index.php>Password_Storage_Cheat_Sheet)

Configuration de la base de données

La configuration d'une base de données est très importante car de nombreux points peuvent vite s'avérer cruciaux en cas d'attaque :

- Si une base de données n'est pas maintenue à jour, elle peut être la cible d'attaques diverses et connues sur les versions antérieures. [Une liste longue comme le bras de vulnérabilités](#) existe sur les anciennes versions de MongoDB. Il est donc essentiel de garder la base de données à jour.
- Une mauvaise gestion de droits des utilisateurs de la base de données peut être dangereux si un utilisateur accède directement à la base données.
- Une notion importante dans une base de données est celle de *built-in function*. En cas de réussite d'une attaque NoSQL, si celle-ci sont laissées activées par défaut, le pivot et la compromission complète de la base de données devient facile pour un attaquant même inexpérimenté.

Les contre-mesure de la sous-section 1.5.3 expliquent quels modèles utiliser pour la configuration d'une telle base de données.

1.4.2 Contournement d'authentification

Ce scénario concerne principalement les groupes front-end et back-end.

Comme précisé dans notre diagramme des flux (sous-section 1.2.6), nous voyons qu'il existe plusieurs rôles communiquant avec le back-end.

1. Les administrateurs
2. Les utilisateurs normaux
3. Les capteurs

Ceux-ci disposent bien évidemment de possibilités différentes dans le cadre du projet. Évidemment cette séparation des pouvoirs est importante et les cloisonnement des rôles est primordial afin de limiter au maximum les abus. Un utilisateur normal ayant la possibilité d'abuser le système pour devenir administrateur, peut représenter un grand risque pour la plateforme. Tous les autres cas de changement illicite de rôle sont tout aussi risqués et il est nécessaire de mettre en place des protections contre ce genre de motivations.

Scénario : Jean-Kevin, un script kiddie entreprenant, souhaite montré à ses copains ses progrès en informatique. Pour cela, il prend pour cible un projet sur lequel travaille activement un de ces collègues étudiant à la HEIG-VD. Il veut montrer qu'il peut se connecter en tant qu'administrateur sans connaître le mot de passe. Il arrive à ses fins en se connectant au compte d'un administrateur qui n'était pas protégé par un mot de passe fort et peut disposer à sa guise de l'application.

Impact : Haut

Source de menace : Script kiddies, hackers

Motivation : La gloire (script kiddies)
Nuire à l'application et aux utilisateurs (hackers)

Cible : Données et fonctionnement de l'application

Contrôles : Protéger la session ouverte contre des vols de sessions externes
Empêcher l'utilisateur de faire malgré lui des actions dans sa session
Protéger correctement les sessions avec des mots de passe forts

Cross-site scripting

Une faille XSS exploitée correctement par un attaquant permet de voler un cookie de session valide ou de faire en sorte que du code soit exécuté involontairement par un utilisateur. Cette vulnérabilité fait aussi partie du top 10 de l'OWASP.

Voici le lien officiel décrivant donc le fonctionnement de telles attaques :

[https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_(XSS))

Brute-force du système de login

Une autre possibilité pour réaliser ce scénario est la brute force pur et simple du système de login. Sans contrôle particulier, un utilisateur possédant un bon dictionnaire de nom d'utilisateurs et de mots de passe peut tenter de forcer le login de l'application et donc d'accéder directement à une session qui n'est pas la sienne.

Cross-site request forgery

Les attaques CSRF⁴ sont souvent ignorée dans les applications web et pourtant, elles peuvent avoir des conséquences importantes au niveau de la sécurité. Une faille CSRF est difficile à mettre en évidence et à exploiter, mais si aucune protection n'a explicitement été mise en place, on peut supposer que l'application en question sera vulnérable aux CSRF.

Contrôles d'accès

Le contrôle d'accès est un point **crucial** de la sécurité des applications web. Si un utilisateur lambda peut accéder à une page ou à une fonction administrateur, l'aboutissement du scénario est complet. Si le contrôle d'accès n'est pas bien fait, une élévation verticale ou horizontale des priviléges devient possible.

Rappelons ici que notre application possède quatre vues en tout :

- Les simples visiteurs non-connectés
- Les utilisateurs standards
- Les administrateurs
- Les capteurs

Le contrôle d'accès doit bien évidemment **cloisonner** chacun des rôles à sa propre vue même si les implications ne semblent pas importantes. Pour cela, la sous-section 1.5.7 de contremesures décrit un système qui peut être implémenté pour un contrôle d'accès correct avec une structure comme la notre.

Durée de vie de la session

Si l'application ne vérifie pas la durée de vie d'une session. Un utilisateur laissant sa session connectée laisserait n'importe quelle personne qui, prenant la main sur l'ordinateur, utiliser le compte de l'utilisateur. Prenons l'exemple de l'utilisateur allant chercher un café. Si une personne avait accès à l'ordinateur de l'utilisateur il pourrait faire toutes les opérations qu'il voudrait avec ce compte pendant une durée illimitée. De plus, si celui-ci est un peu doué en informatique, il peut copier le cookie de session pour le réutiliser à volonté depuis son ordinateur.

En cas de ban d'un utilisateur, le manque de durée de vie est aussi problématique. Imaginons qu'un administrateur décide de bannir un utilisateur en supprimant son compte. Si cet utilisateur est connecté au moment du bannissement, il n'aura aucune répercussion à part s'il perd son cookie, ce qui est assez rare. Tandis qu'avec une durée de vie de session, ce dernier, après un certain temps, ne pourra plus effectuer d'action sur le site et sera déconnecté.

4. Petit rappel sur le CSRF : https://en.wikipedia.org/wiki/Cross-site_request_forgery

1.4.3 Récupération passive d'information

Ce scénario concerne principalement le groupe infrastructure.

Souvent, on imagine un hacker pénétrant le système et réalisant des **actions** sur ce dernier. Ne considérer que ce pan de la sécurité est une grave méprise. Rappelons le triangle suivant :



FIGURE 1.10 – Principes essentiels de la sécurité informatique

La confidentialité peut en effet être mise à mal sans même une action directe sur les données. L'interception de données peut permettre la lecture de ces dernières et si l'information n'est pas obfuscée, on peut récupérer des données sans le consentement des parties et donc mettre à mal la sécurité de l'information. Il est donc primordial de protéger les données dans leur transmission surtout sur des médiums partagés tels que les réseaux sans fil.

Scénario :	Les concurrents directs de notre application cherchent à obtenir un maximum de données afin de proposer un service plus précis et exhaustif que le nôtre. Pour arriver à leurs fins ils décident d'écouter le trafic généré par nos capteurs à l'intérieur de l'école. Si le trafic n'est pas chiffré, les concurrents peuvent l'utiliser de manière illicite dans leur alternative.
Impact :	Haut
Source de menace :	Hackers, concurrents et cybercriminels
Motivation :	Tremplin vers une plus grosse attaque (hackers) L'argent (cybercriminels) Nuire à notre image, voler des données (concurrents)
Cible :	Les données utilisateurs et métier
Contrôles :	Éviter que l'application ne fournit des données sur elle-même ou sur les utilisateurs Protéger les communications entre les éléments du système

Communications en clair

Le wifi est une chose fantastique, mais malheureusement les données passent dans l'air et donc tout le monde peut les "voir". Cela implique évidemment que toutes les données passent en clair sur le réseau mots de passe et données comprises. C'est évidemment pas sécurisé et la confidentialité des données est fortement impactée. Le compromis idéal est la mise en place d'un tunnel TLS pour toutes les communications HTTP de notre projet. Plus de détails sur la mise en place d'un tel système sont donnés dans la sous-section 1.5.9.

Le protocole HTTP classique n'est pas le seul utilisé dans notre projet. Nous utilisons aussi le protocole LoRa qu'il faut donc aussi rendre illisible pour toute personne effectuant une capture sur le réseau.

Messages d'erreur révélant des informations

Les messages d'erreur sont un moyen de communiquer comme un autre avec l'utilisateur. Il ne doivent en aucun cas révéler des informations qui pourraient être exploitées par quelqu'un de malveillant.

L'exemple le plus courant est celui du message d'erreur au login indiquant si oui ou non un utilisateur existe. Cela donne déjà beaucoup d'information à un hacker qui n'aura plus qu'à deviner le mot de passe pour usurper la session.

1.4.4 Utilisation frauduleuse du matériel

Ce scénario concerne principalement le groupe firmware.

La nature d'un projet en Internet des Objets implique la présence de capteurs et ou autres éléments physiques dans un bâtiment. Évidemment ces éléments sont exposés "au public" et peuvent être pris pour cibles par des personnes mal-intentionnées. Cette partie de la sécurité est important mais aussi relativement difficile à mettre en place.

Scénario :	Un étudiant un peu curieux et kleptomane sur les bords repère dans l'école une jolie borne composée d'une Raspberry Pi. Celui-ci décide un jour que la borne ne sert pas à grand chose et il l'embarque pour son usage personnel.
Impact :	Haut
Source de menace :	Étudiants kleptomanes, script-kiddies
Motivation :	Profit personnel (étudiants kleptomanes) La gloire (script-kiddies)
Cible :	Le matériel et le fonctionnement de l'application
Contrôles :	Ne pas laisse le matériel à des endroits accessibles à tous Mettre en place des "fuse protections" sur les éléments physiques

Accès non-contrôlé au hardware

Empêcher le vol du matériel est quelque chose de compliqué parce que c'est une mesure qui pourra toujours être contournée avec des moyens. Si le concepteur met le dispositif dans un boîte, la boîte peut être volée etc...

Pourtant c'est un point qu'il est important de ne pas négliger dans un projet pareil car le matériel coûte cher.

Composants réinscriptibles

Au cas où le matériel aurait été subtilisé malgré les protections mises en place au point précédent, il faut éviter au maximum que le profiteur ait envie de recommencer. Si celui-ci vole du matériel et peut être en mesure de l'utiliser, ça peut vivement l'encourager à recommencer. Alors que si des protections sont mises en place, le risque est un peu mitigé. Dans les contre-mesures de la sous-section 1.5.12, une méthode très connue des développeurs hardware est présentée pour protéger les matériaux contre un écrasement de leur contenu actuel.

1.4.5 Altération des données

Ce scénario concerne principalement les groupes infrastructure et firmware.

D'ordinaire, on se dit qu'un individu malveillant cherche forcément à avoir le contrôle complet de sa cible. Néanmoins, le plus souvent un contrôle ou une altération partielle du système ou des données peut avoir de grosses conséquences sur le système. Il est donc très important de protéger dans les trois axes présentés précédemment dans la sous-section 1.4.3 : confidentialité, intégrité, disponibilité.

Scénario :	Un concurrent direct à notre application désire montrer que leur solution est meilleure que la notre. Pour cela, ils décident d'altérer les données transmises par nos capteurs pour qu'elles soient erronées. Plusieurs endroits peuvent être vulnérables notamment les trames LoRa et les données transmises par le back-end. Il faut être capable d'éviter des attaques de type <i>man in the middle</i> .
Impact :	Moyen
Source de menace :	Concurrents, hackers, organisations étatiques
Motivation :	Nuire à l'image de l'entreprise (concurrents) L'argent (hackers) L'espionnage (organisations étatiques)
Cible :	Les données métiers
Contrôles :	Contrôle d'intégrité sur les données transmises Contrôle de l'association capteurs - serveur

Man in the middle

Les attaques de types *man in the middle* (MITM) sont très connues. Les attaquants font en sorte, lors d'une attaque de ce type, de se retrouver comme intermédiaire entre deux éléments du système. D'un côté le client aura l'impression de converser avec le bon serveur et le serveur avec un client légitime.

Pour limiter ce type d'attaque, on utilise les contrôles d'intégrité mis sur les trames. Ceux-ci reviennent à signer les messages avec une clé privée afin que ceux-ci ne puissent être modifiés par un MITM sans que cela ne se répercute sur la signature. Dans notre cas, il conviendra de protéger les communications HTTP (avec TLS par exemple), et les communications LoRa.

Plus de détails sur ces contre-mesures sont documentées dans les sections 1.5.9 et 1.5.13.

Association des capteurs à la passerelle

Pour un individu malveillant, un moment idéal pour réaliser une attaque *man in the middle* est celui où les composants s'associent. Le moment où un capteur est ajouté au système est crucial car c'est là que le lien de confiance est établi. Des précautions particulières sont à apporter sur ce point, et des protocoles existants du LoRa permettent cette association avec sécurité.

1.4.6 Altération du fonctionnement de l'application

Ce scénario concerne la plupart des groupes.

De nos jours, les données sont la cible privilégiée des attaques. Néanmoins, une altération ou une interruption du service est tout aussi dommageable pour l'entreprise. Une perte de confiance des utilisateurs nuit rapidement à la réputation et à l'image du prestataire de service. Des fois les hackers jouent sur ce tableau afin de se faire de l'argent facilement sur le dos des entrepreneurs. C'est un problème sécuritaire majeur dans les systèmes et tous les pans du projet peuvent être vulnérables. Ce scénarios reprend en vrac toutes les attaques qui peuvent affecter la disponibilité et le fonctionnement de l'application.

Scénario :	Une organisation de cybercriminels cherche une nouvelle cible et tombe par hasard sur notre projet. Pour se faire de l'argent ils décident de compromettre l'application en la ralentissant ou en touchant sa disponibilité tout en faisant des demandes de rançon aux concepteurs.
Impact :	Moyen
Source de menace :	Concurrents, script-kiddies, hackers
Motivation :	Nuire à l'image de l'entreprise (concurrents) L'argent (hackers) La gloire (script-kiddies)
Cible :	La disponibilité, le fonctionnement de l'application
Contrôles :	Redondance des éléments Contrôle des entrées utilisateurs

Attaque par déni de service

Tous les systèmes informatiques quels qu'ils soient sont vulnérables aux dénis de service. Le principe reste très simple : envoyer trop de requêtes / demandes / trames à une entité pour que celle-ci devienne incapable de toutes les traiter et soit ralentie voir arrêtée. Le problème principal du DDoS (*Distributed Denial of Service*), est qu'il n'est pas possible à endiguer entièrement même avec toutes les mesures du monde. Une menace disposant de moyen techniques et financiers illimités sera toujours capable de créer un déni de service sur une cible.

Néanmoins nous présentons dans la sous-section 1.5.15 plusieurs moyen de mitiger un peu les risques. Mais il convient de garder à l'esprit que pour le déni de service la protection ultime n'existe pas.

Brouillage du réseau

Une autre attaque qu'il est difficile voir impossible de contrer est le brouillage réseau. Nous n'allons pas plus en parler ici car c'est un risque qui pèse sur tout le monde des ondes hertziennes et qui ne peut pas être mitigé à cause des législation qui régissent les réseaux sans-fil (bandes ISM, puissance isotrope rayonnée équivalente).

Intrusion dans les systèmes hébergeurs

Malgré toutes les failles multiples et variées que nous avons traitées jusqu'ici, l'essentiel pour les hackers reste l'intrusion généralisée et classique dans un système. Un port ouvert avec un service vulnérable, une version de l'OS dépassée comportant des failles de sécurité et des failles permettant une escalade de privilège vers un compte administrateur de la machine et **tout le système est compromis dans l'arrière boutique.**

Ces attaques sont les plus dangereuses car elles donnent un accès complet à tous les services, à tous les fichiers, à toutes les bases de données, à toutes les clés privées utilisées pour mettre en place la sécurité jusqu'ici. De plus un attaquant disposant d'un tel contrôle sur une machine d'hébergement peut purement et simplement supprimer le service, l'application, le projet ou même formater complètement tout ce qui lui passe sous la main. Évidemment, inutile de parler de l'impact immense qu'une telle attaque peut avoir sur un système.

Ralentissement volontaire de la base de données

Il ne faut jamais faire confiance à un utilisateur, jamais. Toutes les entrées doivent être contrôlées lors du développement d'un tel projet. Que ce soit pour éviter les attaques de type injection NoSQL ou XSS mais aussi pour éviter une surcharge de données pouvant mener à un ralentissement d'une base de données.

Par exemple un champ beaucoup trop grand 1'000'000 de caractères représente environ 1 MB de données à stocker dans une base de données. Si l'abus est répété, on arrive vite à un volume de données important qui peut ralentir considérablement une base de données (même NoSQL !).

1.5 Contre-mesures

Comme nous avons dans ce projet beaucoup de scénarios, de failles et donc de contre-mesures, nous n'allons donc pas les développer en détails ici. Néanmoins, pour chacune d'entre elle un descriptif plus ou moins fourni de la technique est donné. Celui-ci est accompagné de liens⁵ vers des solutions existantes pour une architecture proche de notre système.

Il est à noter que les contre-mesures doivent, dans la mesure du possible, utiliser des librairies existantes et considérées comme sécurisées et à jour. Les solutions "*home made*" sont à éviter car généralement elles vont souffrir de manquements ou d'oublis.

1.5.1 Injection NoSQL

Pour protéger notre base de données de ce type d'attaque, il faudrait nettoyer les entrées de toutes lignes de codes exécutables. Pour ce faire, il faudrait utiliser la librairie suivante : mongo-sanitize.

Le code ci-dessous donne un exemple de quand utiliser cette librairie (source [NPM](#)) :

```
1 var sanitize = require('mongo-sanitize');
2
3 var name = sanitize(req.params.name);
4 var password = sanitize(req.params.password);
5
6 User.findOne({ "name" : name, "password" : password }, callback);
```

Librairies et documentation :

- Outil – Librairie NPM de mongo-sanitize
- Article – Injection NoSQL dans Node avec MongoDB
- Stackoverflow – Injection prevention in MongoDB
- OWASP – Testing for NoSQL injection

1.5.2 Lisibilité du contenu sensible

Afin de protéger les mots de passe des utilisateurs même en cas de vol de la base de données, il est indispensable de hacher les mots de passe avec des méthodes de hachages qualifiés de surs comme SHA256 ou SHA3. Ainsi, même en ayant tous les hashs des utilisateurs, l'attaquant n'a pas les mots de passe des utilisateurs et ne peut donc pas utiliser leurs comptes.

Afin d'augmenter encore la sécurité de notre application, nous pouvons ajouter du sel à notre application. Le sel consiste à concaténer au mot de passe de l'utilisateur une chaîne de caractère aléatoire afin que deux mots de passe identiques ne donnent pas le même hash. Cette chaîne de caractère est stockée dans la base de données. Le sel est une excellente protection contre les attaques par rainbow table car il faut brûler les mots de passe un par un.

5. En anglais, car la bonne documentation est en anglais

Librairies et documentation :

- OWASP – Password storage cheatsheet
- Stackoverflow – Storing passwords with Node.js and MongoDB
- Outil – Bcrypt pour Node.js
- Article – Node et bcrypt

1.5.3 Configuration de la base de données

Afin de sécuriser notre base de données NoSQL, l'idéal serait de définir les rôles selon un modèle RBAC (Role based Access Control).

Ainsi, chaque compte de la base de données ne donnera accès qu'aux fonctions dont a besoin le service ou l'utilisateur. Il faut aussi logger les accès à la base de données afin de pouvoir détecter d'éventuels comportements suspects ou bugs. De plus pour assurer une bonne disponibilité de cette BDD, il faudrait la monitorer pour pouvoir détecter les problèmes en amont.

Si la tolérance au down time pour notre application venait à devenir critique, nous devrions envisager de mettre en place un système de base de données redondant. Il serait aussi nécessaire de définir une politique de backup afin de pouvoir se prémunir contre une perte ou une altération de nos données.

Librairies et documentation :

- Documentation officielle – RBAC dans MongoDB
- Documentation officielle – Gestion des utilisateurs et des rôles
- Documentation officielle – MongoDB security checklist

1.5.4 Cross-site scripting

Pour protéger notre application contre les attaques de type xss, il faudrait passer les entrées fournies par l'utilisateur dans un filtre choisi afin d'échapper correctement tout code malveillant.

Librairies et documentation :

- OWASP – XSS prevention cheat sheet
- Article – XSS et applications Node.js
- Outil – Librairie NPM de xss nettoyage des inputs
- Outil – Librairie NPM de secure-filters nettoyage des inputs

Plein d'autres librairies existantes et correctes sont disponibles dans Node.js !

1.5.5 Brute-force du système de login

Afin de se protéger contre les attaques de type brute force ou envoi massif de formulaire, il faudrait mettre des captchas (comme les reCAPTCHAS de Google) afin de s'assurer que c'est bien un humain qui effectue l'action. Ainsi on rendrait impossible les attaques de type brute force effectuées à l'aide de scripts. De plus, le reCAPTCHA de Google est vraiment simple d'utilisation et gère de manière automatique une blacklist des adresses IP qui ont tenté de brute-forcer un formulaire.

Librairies et documentation :

- Documentation officielle – [Google reCAPTCHA](#)
- Tutoriel – [Google reCAPTCHA avec Node.js](#)
- Outil – Librairie NPM de [express-recaptcha](#)
- Outil – Librairie NPM de [recaptcha2](#)

1.5.6 Cross-site request forgery

La contre-mesure la plus répandue pour se protéger des failles web de type CSRF est l'autentification par token. Lorsqu'une requête est soumise au serveur, un header supplémentaire contenant le token est ajouté. Le token est vérifié avant de réaliser l'action. La génération du token est une étape critique. Il est nécessaire qu'il soit généré de manière aléatoire, n'utilisant que des librairies standards.

Il existe d'autres contre-mesures additionnelles, notamment ajouter des étapes de validation (confirmation, captcha) lors des actions jugées critiques. Optionnellement, il est possible de vérifier depuis quelle page la requête a été effectuée. Bien qu'une contre-mesure supplémentaire ne soit pas de refus, il faut savoir qu'il est possible de forger une requête et donc de modifier cette valeur.

Librairies et documentation :

Voir la documentation de la contre-mesure contrôle d'accès dans la sous-section 1.5.7.

1.5.7 Contrôle d'accès

Il est légitime de contrôler l'accès aux différentes parties de l'application afin de s'assurer que l'utilisateur possède les autorisations nécessaires pour accéder à l'information. Le contrôle d'accès prend la forme d'une page de login. Une fois authentifié, l'utilisateur doit recevoir en retour un *Json Web Token* par exemple. Il s'agit d'une technologie très utilisée dans les back-end Javascript. Ce token est envoyé à chaque requête de l'utilisateur. Il contient diverses informations :

- Rôle
- Date d'émission
- Signature
- Émetteur
- Etc...

Si le token est valide (**vérification de la signature**) et n'est pas blacklisté dans une base de donnée, alors la session reste active. Ce type de token empêche à un attaquant de déterminer l'identifiant de session de quelqu'un et donc d'usurper son identité.

Librairies et documentation :

- Documentation officielle – Json Web Token
- Outil – Librairie NPM de jsonwebtoken
- Norme – Json Web Token

1.5.8 Durée de vie de la session

La durée de vie d'une session dépend des informations et des actions dont l'utilisateur a besoin. Il est important de trouver un juste milieu ; si la durée de vie est trop courte, alors l'utilisateur sera obligé de s'authentifier plusieurs fois par jour. Dans le cas contraire, l'inattention d'un utilisateur quittant son bureau sans verrouiller sa session, permettrait à un collègue d'accéder à son compte pendant son absence. De plus, les actions et les données auxquelles un utilisateur standard ou un administrateur à accès sont différentes. C'est pourquoi une distinction entre ces 2 types de profils est nécessaires.

Utilisateur 10 jours

Administrateur 1 jour

En plus de cette distinction et de cette durée de vie, il convient d'avoir une base de données permettant de stocker les tokens correspondant à une session black-listée ou bannie. Pour cela, une simple collection MongoDB de tokens suffit amplement si on pense à la vérifier après chaque vérification de JWT.

Librairies et documentation :

- OWASP – Session management cheat sheet
- Toute la documentation de la contre-mesure 1.5.7 sur les Json Web Token.

1.5.9 Communication en clair

LoRa

Pour des questions de confidentialité des données, les trames ne peuvent pas circuler en clair sur le réseau. Il suffirait à un attaquant de sniffer le réseau pour avoir accès aux informations. Cette contre-mesure est implémentée de base par LoRaWAN.

Certificat SSL pour serveur nginx

Afin de chiffrer les communications entre nos clients et notre serveur web nginx. Pour cela, nous avons créé un dockerCompose qui nous permet de créer un serveur nginx proxy, un serveur nginx et un serveur *nginx let's encrypt companion*.

- **Nginx Proxy**

Notre serveur proxy nginx aura pour tâche de centraliser toutes les requêtes HTTPS et de les rediriger vers les serveurs adéquats (dans notre cas, vu que nous avons qu'un seul service web toutes les requêtes seront redirigées vers notre service nginx). Vu que notre proxy est accessible directement depuis internet, il possède un certificat SSL délivré par Let's Encrypt. Ce certificat est renouvelé à intervalle régulière par le container *nginx let's encrypt companion*

- ***Nginx let's encrypt companion***

Ce container se réveille périodiquement (une fois par heure) et vérifie que le certificat du proxy nginx est toujours valable et qu'il n'expire pas dans moins de 30 jours. Si le certificat est valide et que sa durée de validité est supérieure à 30 jours, il se rendort pour une heure. Dans le cas où le certificat serait absent, invalide ou expirerait dans moins de 30 jours, le conteneur en régénère un et le place dans le dossier certificats du proxy nginx et se rendort pour une heure.

- **Serveur nginx**

Ce conteneur est notre serveur web, il reçoit les requêtes des clients que lui transmet le proxy et lui renvoie le contenu demandé par les clients.

Librairies et documentation :

- Article – [Gestion multi-site avec nginx et let's encrypt](#)
- Outil – [Image Docker Hub JrCs/docker-letsencrypt-nginx-proxy-companion](#)
- Outil – [Image Docker Hub jwilder/nginx-proxy](#)
- Outil – [Image Docker Hub nginx](#)
- Guide – [LoRa security](#)

1.5.10 Messages d'erreur révélant des informations

Nos messages d'erreur sur notre version en production doivent donner le minimum d'informations. Par exemple en cas d'erreur de notre serveur nginx celui-ci doit seulement donner le numéro et le type d'erreur. En aucun cas, il ne doit donner la cause de cette erreur car le message indiquant la cause pourrait divulguer certaines informations utiles à un attaquant. De même, notre serveur nginx ne devra jamais fournir son numéro de build dans un message d'erreur car il suffirait à un attaquant de chercher sur internet les failles pour la version de notre serveur.

Un autre endroit où les messages d'erreur doivent être le plus vague possible est le login. Le message d'erreur en cas de saisie de mauvais identifiants ne doit pas dire si le compte existe ou ce qui est faux entre le nom d'utilisateur et le mot de passe. Il devra juste donner un message générique comme celui-ci : "La connexion a échoué. Veuillez vérifier vos identifiants".

Librairies et documentation :

- OWASP – [Error handling](#)
- MITRE-CWE 209 – [Information exposure through an error message](#)

1.5.11 Accès non-contrôlé au hardware

L'accès aux différents modules (gateway et capteurs) doit être protégé afin d'éviter à une tierce partie d'accéder au firmware ou programmes qui tournent sur ces derniers. En effet, si des clés de chiffrement sont disponibles sur ces modules afin de chiffrer les trames envoyées entre les senseurs et les gateway, il faut s'assurer que l'accès à cette information soit protégé.

Il serait envisageable, comme première mesure, d'enfermer ces modules dans des armoires techniques fermées à clé afin d'en limiter l'accès.

1.5.12 Composants réinscriptibles

Afin de garantir la persistance du hardware en cas de vol, les constructeurs ont mis en place un système appelé "fuse protection". Ces "protection fuse" peuvent être de type logiciel ou matériel. Ils permettent d'éviter que les éléments de la mémoire soient lus, et par conséquent, sont de plus en plus installés dans les chips de nos jours. Cette protection est mise en place, par exemple, en configurant certains bits dans le chip ou en fusionnant les lignes de contrôle correspondantes. Ces *fuse* ne sont pas toujours résistants à différentes techniques de manipulation spécifiques. Ces *Fuse bits* qui sont configurés peuvent être remis à 0 par une injection par laser. Une fois que ces *fuse* sont remis à 0, l'analyse du système (lecture des informations) peut continuer. Une contremesure efficace est un maillage de connecteurs qui est placé autour du chip et qui est électriquement connecté à ce dernier. Si le chip est ouvert ou attaqué physiquement, ce maillage est détruit ce qui résulte en la destruction du chip, rendant impossible la récupération des informations (voir Figure 1.11).

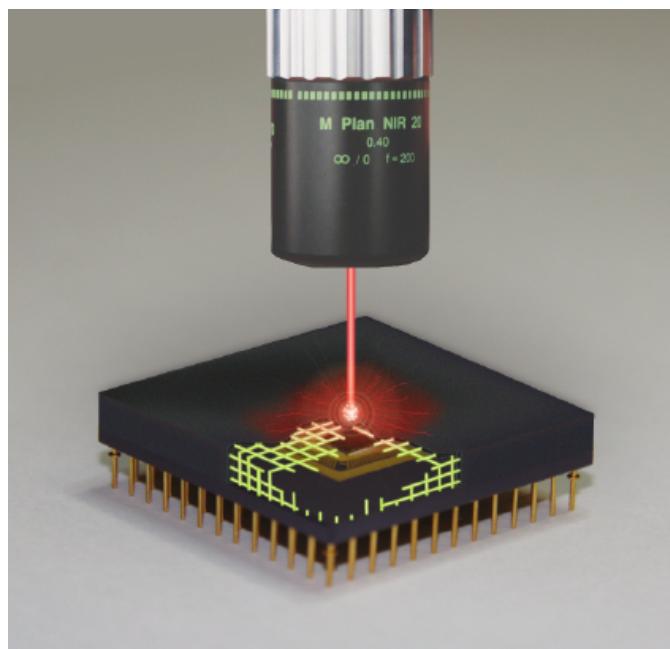


FIGURE 1.11 – Protection anti-modification par maillage

Librairies et documentation :

- Article – Protecting embedded systems against product piracy

1.5.13 Man in the middle

Afin de garantir l'intégrité des données envoyées sur le réseau LoRa, un contrôle d'intégrité de type CRC32 peut être effectué à l'instar du type de contrôle effectué lors de communications Wifi. Cette somme de contrôle serait mise à la suite des données envoyées et le tout pourrait être chiffré, par exemple avec AES dont la clé serait connue du capteur et de la gateway.

Une fois la trame reçue par la gateway, ou par le capteur, elle serait déchiffrée et le CRC32 contrôlé afin d'assurer l'intégrité de la trame reçue.

Librairies et documentation :

- OWASP – [Man in the middle attack](#)
- Toute la documentation de la contre-mesure 1.5.9 sur le chiffrement des communications et les tunnels TLS.

1.5.14 Association des capteurs à la passerelle

La sécurité du réseau LoRaWAN repose sur deux chiffrements AES-128.

- **Network Session Key** (NwkSKey) assure l'authenticité des nœuds du réseau. La *NwkSKey* permet à l'opérateur de sécuriser le réseau. Cette clé est connue par l'opérateur du réseau et par les fournisseurs d'applications autorisés.
- **Application Session Key** (AppSKey) assure la confidentialité des données. La *AppSKey* permet au fournisseur de l'application de protéger les données transitant par le réseau. Cette clé n'est connue que du fournisseur de l'application. Aucun service tiers peut donc déchiffrer le contenu des messages.

Les données envoyées sont chiffrées avec la *AppSKey* et un en-tête est ajouté. Cet en-tête contient notamment les adresses source et destination. Sur la base de cette nouvelle donnée, un MIC est calculé à l'aide de la *NwkSKey* pour garantir l'intégrité. Ce MIC permet au réseau de vérifier l'intégrité du message et authentifier le nœud de l'envoie.

Activation

Les équipements souhaitant communiquer sur le réseau LoraWAN doivent premièrement obtenir des clés de sessions. Deux méthodes sont disponibles : **Over-The-Air Activation** (OTAA) ou **Activation By Personalization** (APB) qui est hors scope dans notre cas.

Le nœud doit transmettre une *join request* au réseau sur la base de 3 paramètres.

1. Un identifiant unique (type EUI-64), **DevEUI**
2. L'identifiant du fournisseur de l'application, **AppEUI**
3. Une clé AES-128 déterminée par le fournisseur d'application, **AppKey**

Une fois ces données envoyées au serveur d'enregistrement, il vérifie le MIC de la requête. Si la requête est valide, le serveur envoie une réponse contenant les informations nécessaire pour dériver les clés de sessions (AppSKey et NwkSKey), l'adresse du nœuds sur 32 bits. Les clés de sessions sont générées à chaque nouvelle session.

Comme la sécurité s'appuie entièrement sur une chiffrement symétrique, il est important que le transfert de cette information entre les différentes entités se fasse d'une manière sécurisée. De plus, comme l'équipement stocke une information sensible, à savoir la clé de chiffrement, il faut le faire de manière sécurisée. En dehors de cela, nous pouvons considérer le système sécuritaire natif de LoRaWAN comme tout à fait convenable.

Librairies et documentation :

- Documentation officielle – LoRaWAN security
- Article – Inquiétude sur la sécurité de LoRaWAN

1.5.15 Attaque par déni de service

Afin d'éviter toute attaque du type déni de service, il serait envisageable d'effectuer un contrôle sur les ip se connectant au backend par exemple afin de limiter ou bloquer l'accès de cette dernière au backend si un certain seuil de requêtes par secondes ou minutes est atteint. De plus, il serait possible de mettre les différents serveurs (network server) en liste blanche. Ceci permettrait que seul les gateway et serveurs autorisés puissent se connecter au backend pour pousser des données, mitigeant ainsi le risque de DDoS.

En ce qui concerne le frontend, seul un utilisateur authentifié peut accéder aux données. On pourrait filtrer les requêtes ICMP afin d'éviter une attaque de type *smurf attack*. Afin d'éviter une attaque de type *SYN Flood*, on pourrait limiter dans le temps le nombre de requêtes effectuées par un certain client authentifié.

Il serait possible d'héberger le backend sur un fournisseur IaaS afin de permettre un *autoscaling* lorsque le serveur est surchargé. Toutefois, cette solution serait inefficace si l'attaque par DDoS est de grande envergure et le coûts engendrés seraient énormes.

Librairies et documentation :

- Documentation officielle – AWS auto-scaling group
- Article – VPN provide DDoS protection
- Wikipedia – Mitigation de DDos

1.5.16 Intrusion dans les systèmes hébergeurs

Si l'infrastructure est hébergée chez un fournisseur IaaS, les mises à jour du système sont à la charge du développeur. Si un service PaaS est utilisé, ces mises à jour sont à la charge du fournisseur de services.

Les applications utilisées ainsi que les systèmes d'exploitations doivent être régulièrement mis à jour afin de contrer les vulnérabilités découvertes et corrigées.

Librairies et documentation :

- Article – Risk running obsolete software

1.5.17 Ralentissement volontaire de la base de données

Les saisies utilisateurs sont sujets à beaucoup d'attaques différentes puisqu'elles constituent en réelles portes d'entrées à l'application. Il est primordial de contrôler les saisies et leurs tailles. C'est à dire vérifier les caractères pour écarter les caractères spéciaux (voir sous-section 1.5.4) et aussi contrôler la taille de la saisie. Cette contre-mesure permet d'éviter des attaques de type *buffer overflow*. De plus si on accepte des inputs trop grands, par exemple 1'000'000 de chars, on enregistra une entrée de 1MB dans le base de données. La base de données deviendra beaucoup trop lente pour être utilisable correctement.

Librairies et documentation :

- OWASP – Validation cheat sheet

1.6 Conclusion sécuritaire

Pour conclure, voici un court résumé des différentes vulnérabilités que nous avons rencontrées lors de ce projet de sécurisation accompagnées de quelques remarques.

Vulnérabilité	Correction ?
Injection NoSQL	✓
Lisibilité du contenu sensible	✓
Configuration de la base de données	✗
<i>Cross-site scripting</i>	✗
Brute-force du système de login	✗
<i>Cross-site request forgery</i>	✗
Contrôle d'accès	✓
Durée de vie de la session	✓
Communication en clair	✓
Messages d'erreur révélant des informations	✓
Accès non-contrôlé au hardware	✗
Composants réinscriptibles	✗
<i>Man in the middle</i>	✓
Association des capteurs à la passerelle	✓
Attaque par déni de service	✗
Intrusion dans les systèmes hébergeurs	✓
Ralentissement volontaire de la base de données	✓

Et plein d'autres... Bien évidemment, certaines manquent sûrement à l'appel mais les grosses ayant un gros impacts sont belles et bien listées ici.

Remarques diverses

- Durant ce projet, nous avons beaucoup travaillé sur la recherche de scénarios et sur la manière de les contrer. Parfois, les contre-mesures n'ont pas été appliquées mais les détails concernant leur mise en place sont données.
- D'autres vulnérabilités que nous ignorons se trouvent sûrement encore dans le projet. La sécurité est quelque chose de complexe et ne peut jamais être parfaite. Nous avons donné les pistes pour sécuriser l'application au mieux, avec nos connaissances et le temps à notre disposition.
- Ce document n'est bien évidemment pas là pour pointer du doigt les failles encore existantes. Il est là pour fournir toutes les informations nécessaires afin d'améliorer au maximum la sécurité de ce jeune projet.

Chapitre 2 : Partie firmware

2.1 Technologies utilisées

2.1.1 Capteur

<https://www.espruino.com/BME680>

La BME680 est un capteur d'environnement fabriqué par Bosch. Elle permet de mesurer la température, la pression, l'humidité et la résistance au gaz. Elle s'exécute à partir d'une tension de 3.3V et communique avec la sortie du microcontrôleur via les bus SPI ou I2C. Dans le cadre de ce projet, nous utilisons le protocole I2C.

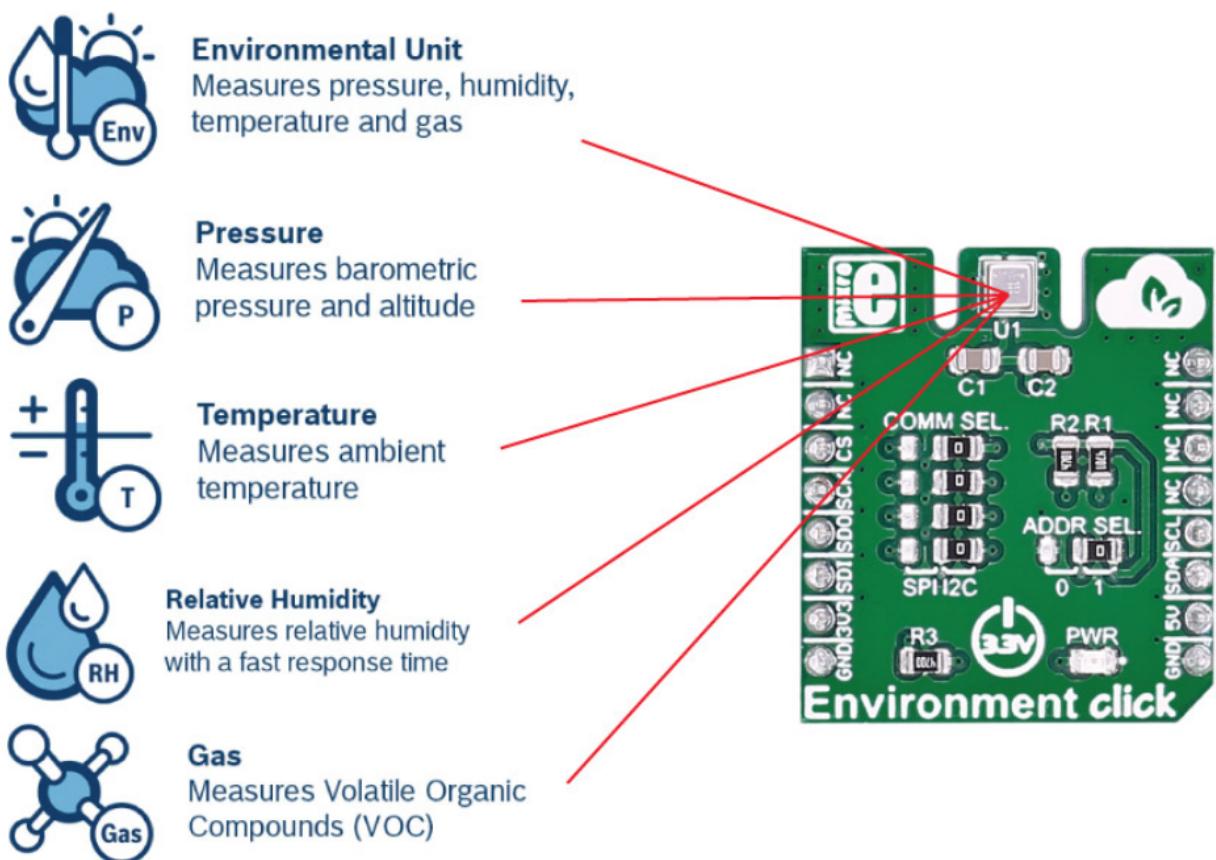


FIGURE 2.1 – Détails du capteur BME680

2.1.2 LoRa

<https://www.espruino.com/RN2483> <http://ww1.microchip.com/downloads/en/DeviceDoc/40001784B.pdf>

2.1.3 Connectique

<https://www.mikroe.com/arduino-uno-click-shield>

Arduino Uno est une extension pour Arduino Uno. Il contient deux micros bus permettant de connecter à arduino plus de 75 click board différents. Il permet d'ajouter des fonctionnalités comme GSM, Wifi, Zigbee, Bluetooth...

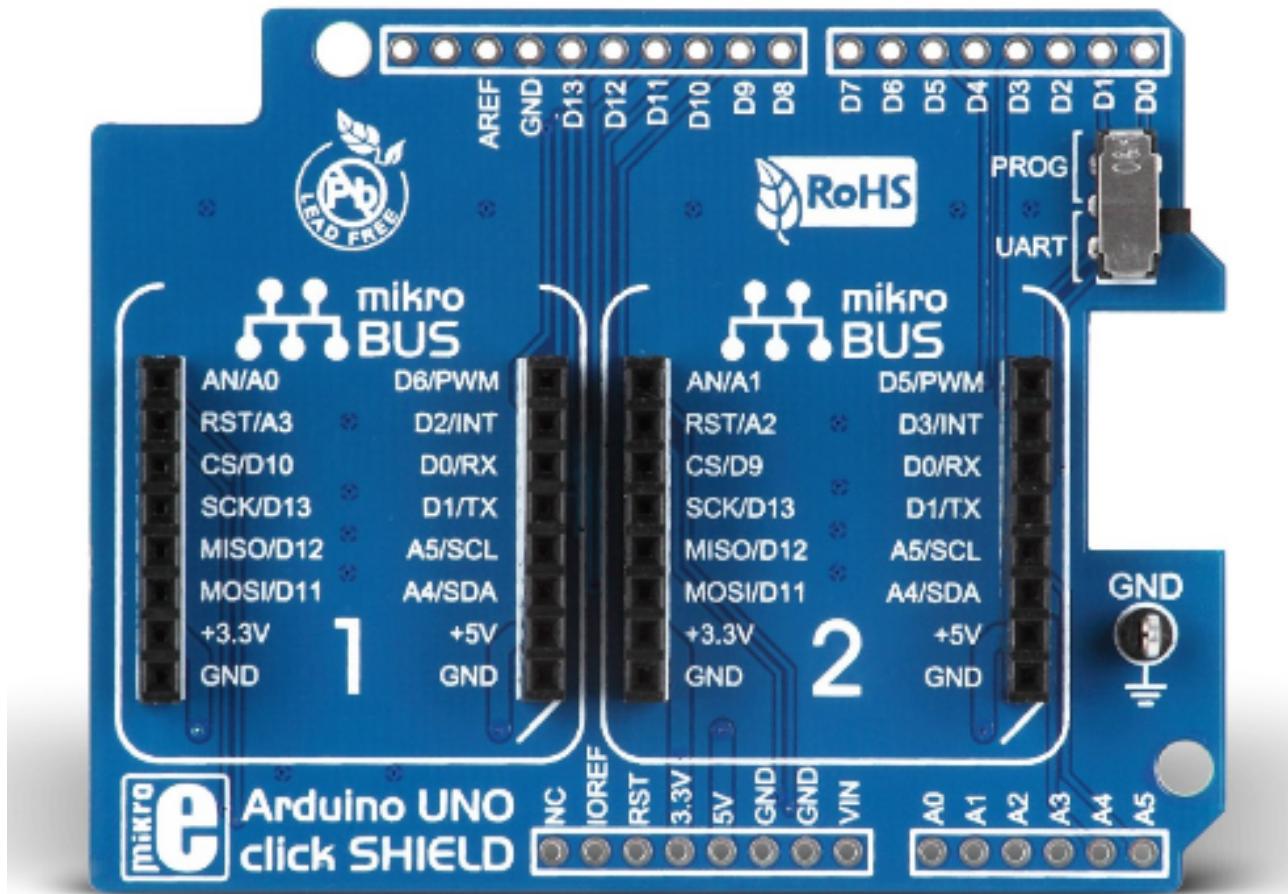


FIGURE 2.2 – Arduino uno click shield

2.1.4 Carte

<http://www.st.com/en/evaluation-tools/nucleo-f401re.html>

La carte STM32 est une carte développée par Nucleo STMicroelectronics. Cette carte dispose

- De deux types de ressources d'extension :
 - Une connectivité Arduino Uno révision 3
 - Un ST morpho pour les pins d'entête permettant un access à toutes les I/O de la STM32.
- Alimentation électrique de carte flexible

- Trois LEDs
- Deux boutons-poussoirs : UTILISATEUR et REMISE À ZÉRO
- Capacité de ré-énumération USB : trois différentes interfaces prises en charge sur USB
 - Port Com virtuel
 - Stockage de masse
 - Port de débogage
- Et toute une série de périphériques et de composants.

Le but est de connecter le module Lora (LoRa click) et le capteur BME680 (environment click) via la connectivité Arduino Uno (Arduino Uno click SHIELD) pour transférer les données du capteur vers l'application web. Pour ce faire, nous avons utilisé l'IDE Espruino et le langage javascript pour la configuration de la partie UART et la partie I2C.

2.1.5 IDE

<http://www.espruino.com/>

L'IDE web Espruino est un éditeur graphique open source conçu pour écrire et débugger du code sur des microcontrôleurs utilisant l'interpréteur javascript Espruino. Il peut également fonctionner en natif via Node.js et Electron, ou une version avec E / S limitée peut être utilisée comme site web.

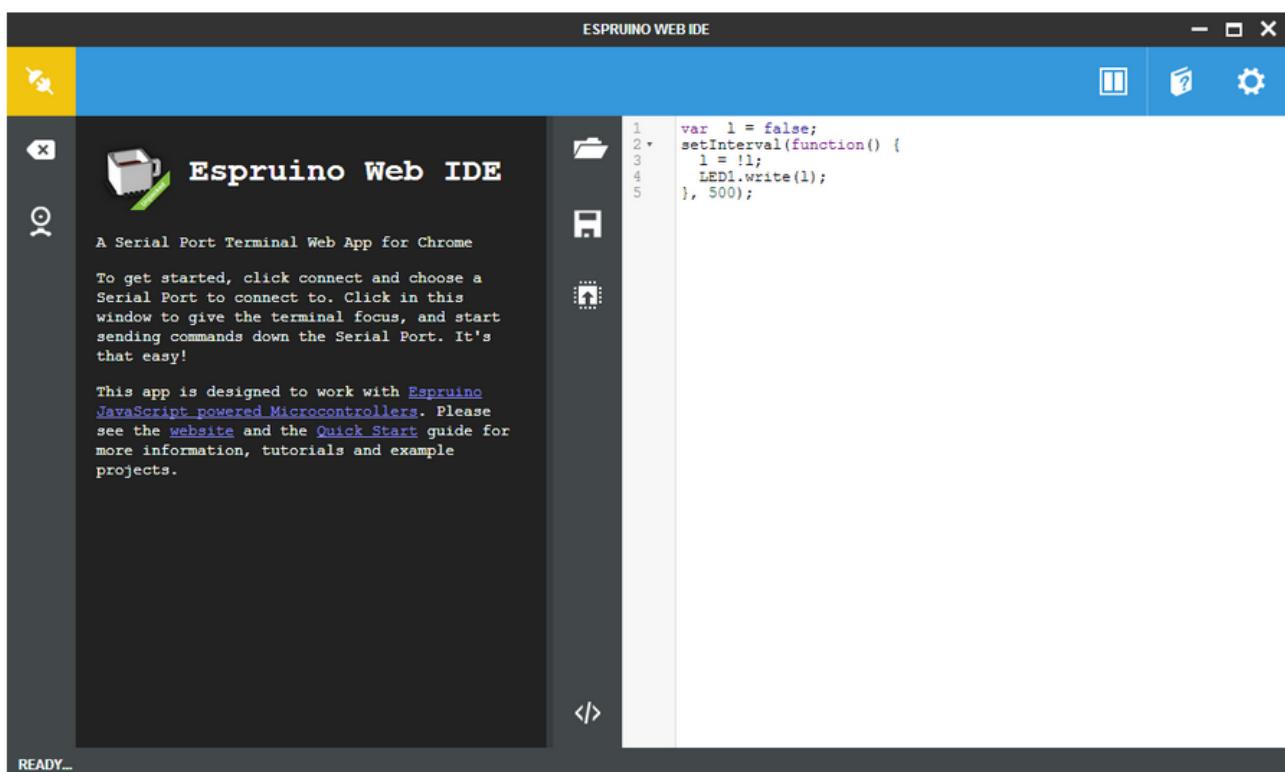


FIGURE 2.3 – Espruino web IDE 1

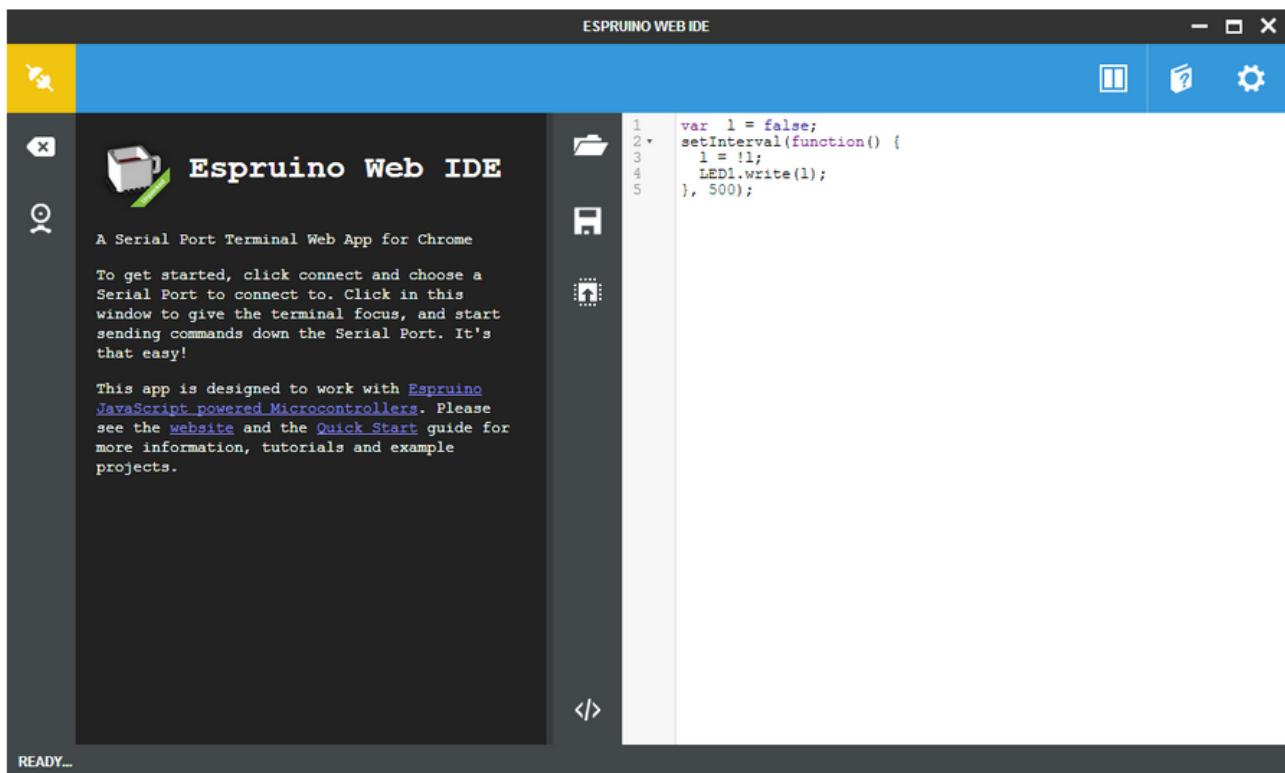


FIGURE 2.4 – Espruino web IDE 2

Version	Communications	Avantages
Chrome Web App	USB, Serial, Audio , TCP/IP	Facile à installer depuis Chrome Web Store
Node.js App / NW.js app	USB, Serial, Bluetooth Low Energy	Peut être lancé sur des systèmes sans web browser
Web Version	Audio , Bluetooth Low Energy (via Web Bluetooth)	Simple visite d'un URL

2.2 Spécificités

2.2.1 Noeuds

No. du device	EUI	Click type
1	0004A30B001ADE26	Environment
2	0004A30B001A798B	Environment
3	0004A30B001AD01A	Environment
4	0004A30B001A602D	Environment
5	0004A30B001A1E25	Ambiant
6	0004A30B001A5BE8	Ambiant

2.2.2 Envoi des données

Header

15	14	13	12	11	10-2	1	0
Temperature	Pressure	Humidity	Gas resistance	Light	Not used	Die temperature	Battery voltage

16 bits pour indiquer ou non la présence de la donnée. Pour les capteurs actuels, les headers sont les suivants :

- Environment (BME680) : F003 (temperature, pression, humidité, gas res., température carte, batterie)
- Ambient (OPT3001) : 0803 (lumière, température carte, batterie)

Données

Taille	Donnée	Conversion
16 bits	Temperature	°C * 10
16 bits	Pressure	hPa * 10
16 bits	Humidity	% * 100
16 bits	Gas resistance	Ohms / 10
16 bits	Light	Lux
16 bits	Die temperature	°C * 10
16 bits	Battery voltage	V * 10

Le module LoRa demande d'envoyer une chaîne de caractères hexadécimale. Les valeurs doivent donc être converties en temps que tel. Pour faciliter la lecture, chaque valeur est envoyée sur 4 digits (16 bits => 4 digits hex).

Exemple de payload envoyée :

- F003003004500000001000240021

On y voit le header F003 (qui signifie que toutes les valeurs sauf la lumière sont dans le payload, comme pour le capteur BME), suivit des champs qui y sont indiqués dans l'ordre (les valeurs ici sont non représentatives des valeurs réelles).

Remarques :

Le niveau de batterie est, pour l'instant une valeur factice car la carte Nucleo ne permet pas de la récupérer.

2.2.3 Communication du changement de fréquence d'échantillonnage

Pour changer la fréquences d'échantillonnage d'un nœuds, il faut lui envoyer un payload en JSON, converti en hexadécimal.

Le JSON se présente sous le format suivant : {"newInterval": value}

La **value** est en millisecondes.

La fréquence d'échantillonnage de base reste à définir

2.3 Déploiement

Avant de faire quoi que ce soit, il faut s'assurer que le FW qui sera loader contient bien les bonnes informations concernant les appKey et appEUI.

Le déploiement des capteurs se fait de manière simple :

1. Changer la position du jumper central droite (pins gauches connectées) pour router l'alimentation sur les batterie et non plus sur USB.
2. Insérer les piles.
3. Load le programme de façon classique grâce à l'IDE.
4. Débranché le câble USB.

Remarques :

Le fait de lancer le FW à l'allumage ne fonctionne pas. Une erreur invalid_param est reçue lors de l'envoi de l'appkey. Nous pensons que c'était du à un problème de capacités mal chargées donc nous avons mis un timeout de 2s avant de lancer l'initialisation des modules et des bus.

Malheureusement, cela n'a pas suffi. Pour palier à ce problème, la marche à suivre en ci-dessus propose d'insérer les piles avant de load le FW. Cela fonctionne.

2.4 Tests effectués

Nous avons tester la connexion au LoRa server avec la gateway TTN. Cela fonctionne sans aucun problème et les données transitent des deux sens. Avec la gateway Raspberry PI, nous rencontrons des problèmes liés à un timeout lors de l'attente de la réponse du join (denied reçu). Nous pensons que cela vient de la gateway qui doit mal gérer la requête.

2.5 Conclusion

Dans l'ensemble, les points qui ont été demandés au groupe FW ont été atteint. Comme dit précédemment, des soucis surviennent lorsque le FW est loader depuis la mémoire directement après allumage et la communication avec la gateway bloque au join. Pour le premier problème, nous avons tenter de le résoudre mais cela n'a pas fonctionné et nous ne comprenons vraiment pas pourquoi. Pour le 2ème, des problèmes du côté de la gateway en sont probablement la cause, vu qu'en utilisant celle de TTN, nous ne rencontrons aucun problème.

En conclusion, ce projet était une bonne approche à l'IoT et nous sommes heureux d'avoir pu travailler sur le FW particulièrement. Le fait d'avoir utilisé le Framework Espruino était une bonne découverte au vu de sa simplicité (debug et API offerte).

2.6 Sources

Payload :

<https://docs.google.com/spreadsheets/d/1PmNmLVsm3uHxBaLA38uTDL-t3WrtRHb4Pu9JmiWj3Kg/edit>

Datasheet :

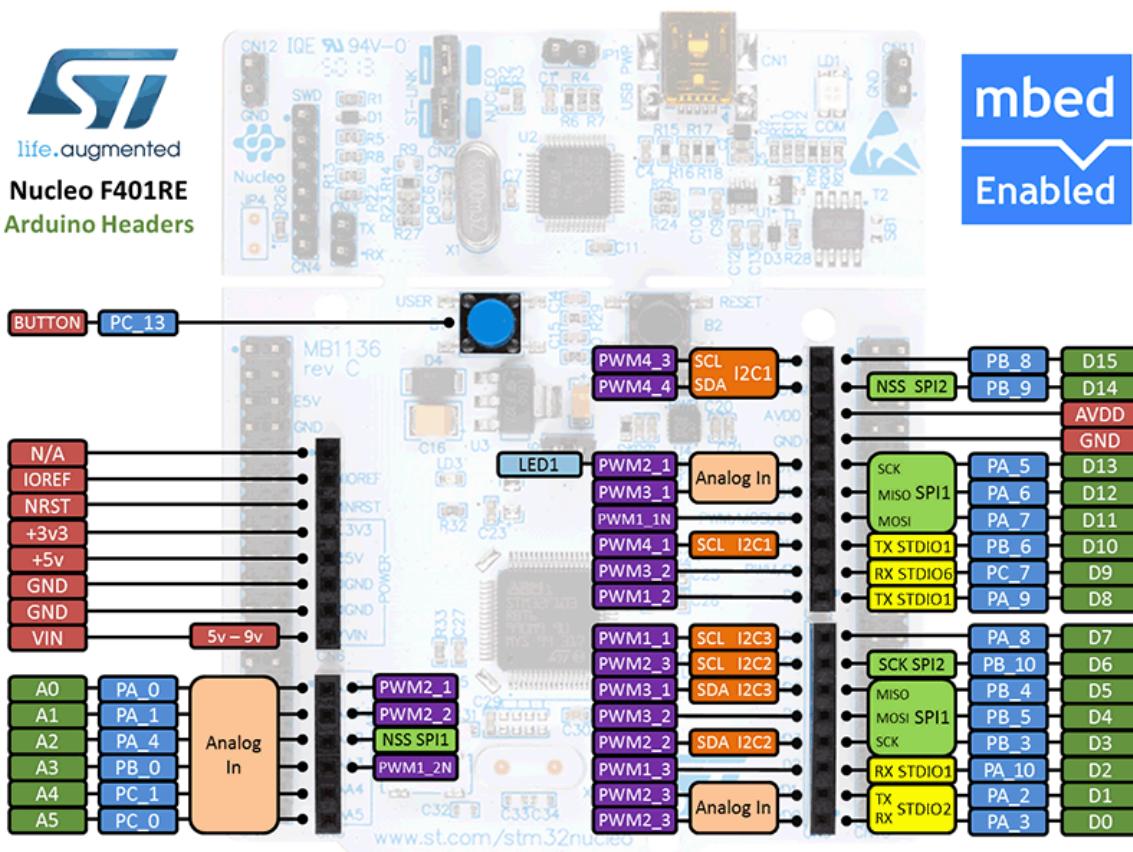


FIGURE 2.5 – Entêtes arduino

Chapitre 3 : Partie infrastructure

3.1 Introduction

Le groupe infrastructure est responsable de mettre en place une solution permettant à des capteurs de communiquer par réseau LoRaWAN avec une application web développée par les groupes front-end et back-end.

Pour cela, il a été décidé de mettre en place notre propre infrastructure LoRaWAN afin de pouvoir héberger l'application web sur un serveur LoRaWAN que nous allons installer sur une machine se trouvant dans le réseau interne de l'école. Une base de données PostgreSQL tourne sur le serveur LoRaWAN pour stocker des données de configuration relatives au protocole LoRaWAN.

L'objectif est donc de mettre en place une infrastructure réseau en installant un network-server LoRaWAN et ses différents composants afin que les capteurs LoRa et l'application web puissent communiquer entre eux.

3.1.1 Architecture LoRaWAN simplifiée

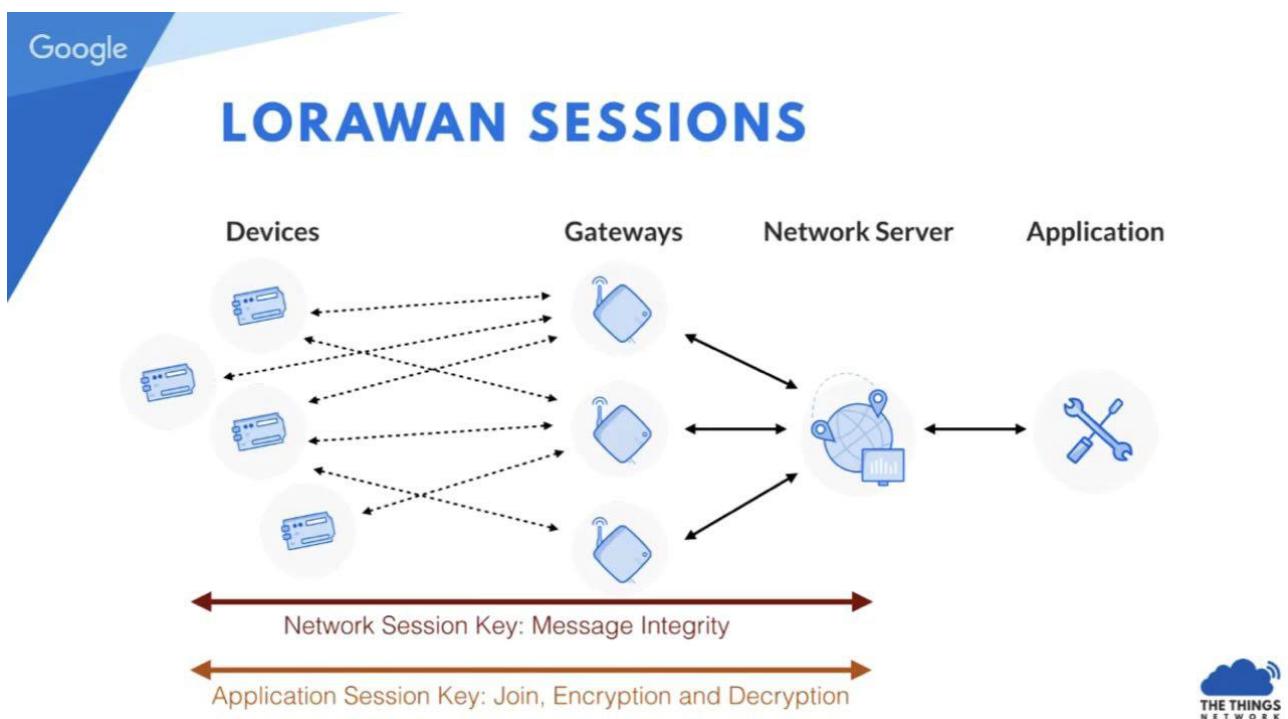


FIGURE 3.1 – Architecture LoRaWAN simplifiée

3.1.2 Architecture LoRaWAN détaillée

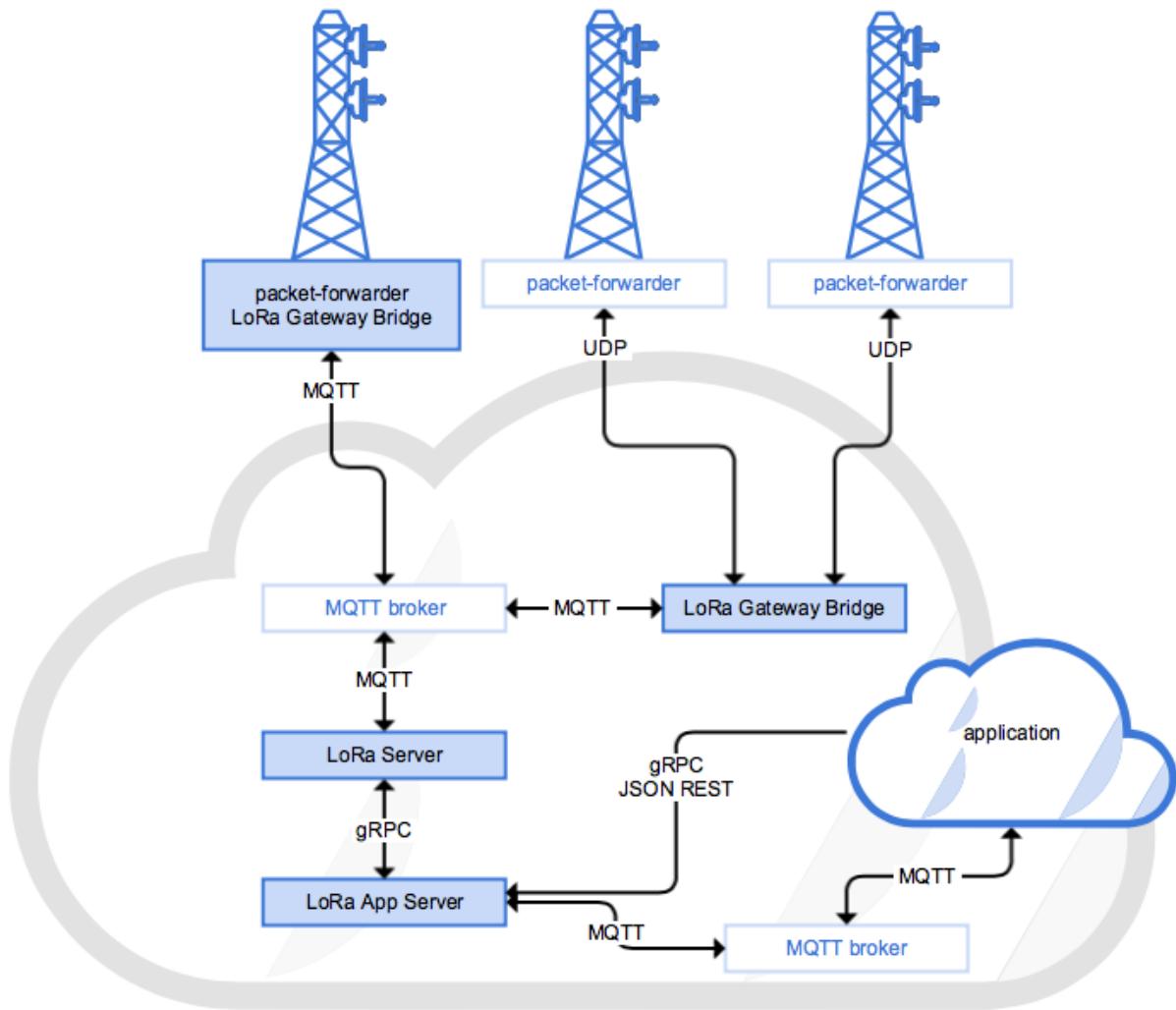


FIGURE 3.2 – Architecture détaillée

3.2 Technologies utilisées

Pour commencer à mettre en place notre infrastructure réseau, nous avons choisi d'utiliser un network-server LoRaWAN open-source mis à disposition par CableLabs sur le site loraserver.io.

Le network-server est formé de plusieurs composants et fonctionnalités que nous allons décrire dans ce chapitre.

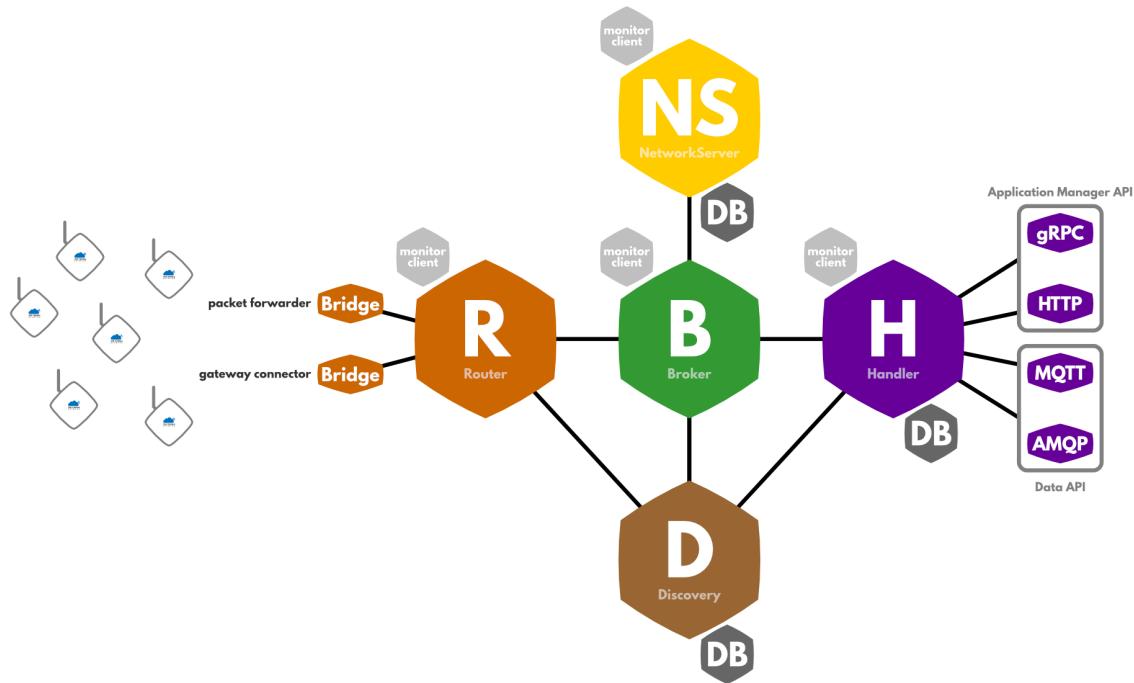


FIGURE 3.3 – Architecture du network-server

3.2.1 Devices LoRa

Les devices LoRa sont dans notre cas les capteurs qui envoient des données au réseau LoRaWAN en passant par les LoRa gateways.

3.2.2 LoRa Gateway

Antenne LoRa qui est connectée au module de routage. Elle envoie les transmissions radio LoRa qu'elle reçoit des capteurs au network-server. Elle implémente un packet-forwarder permettant d'interfacer le hardware LoRa.

3.2.3 LoRa Gateway Bridge

Le LoRa Gateway Bridge est responsable de la communication avec la gateway. Il transforme les paquets transportés sur le protocole UDP en payloads JSON sur le protocole MQTT.

3.2.4 Router

Ce module est chargé de gérer le statut de la gateway et d'organiser les transmissions (scheduling) car une gateway peut effectuer uniquement une transmission à la fois. Il est connecté à un ou plusieurs brokers.

3.2.5 Broker

Le broker est la partie centrale du réseau. Il est responsable du mapping entre les devices LoRa et la ou les applications tournant sur le serveur d'applications. Son rôle est de rediriger un message en provenance d'un certain capteur à la bonne application et dans le cas du chemin inverse de distribuer le message au bon routeur qui va communiquer avec une gateway qui va elle à son tour relayer le message par transmissions radio LoRa au capteur destinataire.

À noter qu'il peut y avoir plusieurs brokers au sein du network-server.

3.2.6 Handler

Le handler doit gérer les données relatives à une ou plusieurs applications. Pour ce faire, il se connecte à un broker où sont enregistrés les différents capteurs et applications ainsi que le mapping entre eux.

C'est également dans le handler que les données sont cryptées ou décryptées.

3.2.7 Gateway Protocol Translation

Quand une gateway reçoit un message transmis avec LoRa, elle encapsule et redirige le message au réseau LoRaWAN. Beaucoup de gateways utilisent le même protocole de référence proposé par LoRa mais d'autres protocoles ont été développés pour des backends spécifiques.

3.2.8 Downlink configuration

Avec LoRaWAN, le temps de réponse du downlink, c'est-à-dire le message allant de l'application au capteur, est dépendant de la localisation géographique de la gateway.

Le module de routage est chargé de gérer les informations relatives aux gateways existantes dont leur position géographique et doit déterminer comment retourner une réponse au capteur.

Après chaque requête qu'un capteur envoie à l'application (message uplink), le router crée deux fenêtres de réponse. La première a lieu exactement 1 seconde après avoir reçu le message du capteur et la deuxième 2 secondes après avoir reçu le message. Donc pour chaque requête reçue sur chaque gateway, le router construit deux downlink configurations.

Afin de pouvoir sélectionner la meilleure option de gateway par laquelle envoyer la réponse au capteur, le router calcule un score pour chaque downlink configuration. Le score pour chaque configuration est influencé par plusieurs facteurs tels que le temps de propagation dans l'air, la force du signal, le taux d'utilisation de la gateway et les réponses déjà planifiées par le router

(scheduling). Par exemple plus le signal est fort, meilleure est la qualité de la transmission et plus la gateway est utilisée, plus celle-ci doit probablement être fiable.

3.2.9 Déduplication

LoRaWAN est un protocole radio à longue portée, ce qui signifie qu'il est plus probable que plusieurs gateways reçoivent et retransmettent un message envoyé par un device LoRa, ce qui génère une déduplication du message envoyé. Le broker est chargé de gérer les messages reçus afin de filtrer les messages dupliqués et de n'envoyer qu'une seule fois le message à l'application.

Les messages dupliqués peuvent quand même s'avérer utiles. Les métadonnées de ces messages peuvent être analysées pour trouver la position exacte du device LoRa qui a envoyé la requête et aussi pour mettre à jour les données permettant au module de routage d'améliorer le calcul du score des downlink configurations de manière à rediriger les réponses (messages downlink) aux gateways de façon optimale.

3.2.10 LoRa App Server

Le LoRa App Server est comme son nom l'indique un serveur d'applications. Il est open-source et est responsable de faire l'inventaire des devices communicant avec l'infrastructure LoRaWAN. Il gère les demandes d'accès des devices aux applications et la réception des payloads de l'application.

LoRa App Server dispose d'une interface web permettant à l'administrateur de gérer les utilisateurs, les devices et les applications autorisés à passer par l'infrastructure du réseau LoRaWAN. Grâce aux APIs RESTful et gRPC, il est possible de faire de l'intégration de services externes.

Lors des transmissions entre les applications et les devices LoRa, les données sont envoyées et reçues en utilisant les protocoles MQTT et/ou HTTP(S).

Exemple d'applications ajoutées

The screenshot shows a web-based application management interface for a LoRa Server. The title bar reads "LoRa Server". The URL in the address bar is "localhost:8080/#/organizations/1/applications?_k=hx4d5g". The top navigation bar includes links for "Organizations", "Users", and "admin". Below the navigation, there are tabs for "Applications", "Gateways", "Organization configuration", and "Organization users". A prominent red "DELETE ORGANIZATION" button is located on the right side of the header. On the far right, there is a "CREATE APPLICATION" button. The main content area displays a table with three rows of application data:

ID	Name	Description
1	air-quality	Air-quality application
7	parking-sensor	Parking sensor application
8	temperature-sensor	Temperature sensor application

FIGURE 3.4 – Application ajoutée

REST API pour le LoRa App Server

The screenshot shows a web browser window displaying the LoRa App Server REST API documentation at localhost:8080/api. The title bar says "LoRa App Server REST API". A "JWT TOKEN" input field is present in the header.

LoRa App Server REST API

For more information about the usage of the LoRa App Server (REST) API, see <https://docs.loraserver.io/lora-app-server/api/>.

Application

		Show/Hide List Operations Expand Operations
GET	/api/applications	List lists the available applications.
POST	/api/applications	Create creates the given application.
DELETE	/api/applications/{id}	Delete deletes the given application.
GET	/api/applications/{id}	Get returns the requested application.
PUT	/api/applications/{id}	Update updates the given application.
GET	/api/applications/{id}/users	ListUsers lists the users for an application.
POST	/api/applications/{id}/users	AddUser adds a user to an application.
DELETE	/api/applications/{id}/users/{userID}	DeleteUser deletes the user's access to the associated application.
GET	/api/applications/{id}/users/{userID}	GetUser gets the user that is associated with the application.
PUT	/api/applications/{id}/users/{userID}	UpdateUser sets the user's access to the associated application.

DownlinkQueue

		Show/Hide List Operations Expand Operations
GET	/api/nodes/{devEUI}/queue	List lists the items in the queue for the given node.
POST	/api/nodes/{devEUI}/queue	Enqueue adds the given item to the queue. When the node operates in Class-C mode, the data will be pushed directly to the network-server.
DELETE	/api/nodes/{devEUI}/queue/{id}	Delete deletes an item from the queue.

Gateway

		Show/Hide List Operations Expand Operations
GET	/api/gateways	List lists the gateways.
POST	/api/gateways	Create creates the given gateway.
DELETE	/api/gateways/{mac}	Delete deletes the gateway matching the given mac address.
GET	/api/gateways/{mac}	Get returns the gateway for the requested mac address.
PUT	/api/gateways/{mac}	Update updates the gateway matching the given mac address.
GET	/api/gateways/{mac}/stats	GetStats lists the gateway stats given the query parameters.

Internal

		Show/Hide List Operations Expand Operations
POST	/api/internal/login	Log in a user
GET	/api/internal/profile	Get the current user's profile

Node

		Show/Hide List Operations Expand Operations
GET	/api/applications/{applicationID}/nodes	ListByApplicationID lists the nodes by the given application ID, sorted by the name of the node.
POST	/api/nodes	Create creates the given node.
POST	/api/nodes/getRandomDevAddr	GetRandomDevAddr returns a random DevAddr taking the NwkID prefix into account.

Go to #!/Gateway/Delete on this page

FIGURE 3.5 – REST API pour le app server

3.3 Spécificités

Il faut être connecté au réseau de l'école ou utiliser un VPN pour pouvoir atteindre les différents composants du système.

VM sur le serveur de l'école :

- Ubuntu 18.04 LTS bionic beaver
- 50GB de mémoire
- 2GB de RAM
- CPU DualCore
- Adresse IP : 10.192.72.26
- Nom de domaine : `iot_lora.lan.iict.ch`
- Docker : installation de docker-ce, package du 04 mai 2018

Gateway :

- Raspberry Pi 3 B
- RASPBIAN STRETCH LITE 4.14 (pas de bureau graphique)
- SD card 4GB
- Connexion à un écran : HDMI
- Connexion à la console Raspberry via un câble série. L'émulateur de terminal (PuTTY par exemple)
- Connexion internet : Ethernet (réception adresse du pool DHCP de l'école)
- Carte Lora : iC880A-SPI

3.4 Déploiement

Dans cette section, nous allons voir comment a été déployée l'infrastructure et comment installer et accéder aux différents composants du système.

3.4.1 Accès à la VM sur le serveur de l'école

Pour accéder à la VM sur le serveur de l'école, il faut tout d'abord se connecter au réseau de l'école puis ouvrir un terminal et taper une des deux commandes suivantes :

```
ssh -l heiguser 10.192.72.26  
ssh -l heiguser iot_lora.lan.iict.ch
```

Remarque : il est possible qu'à ce stade, un message d'erreur apparaisse en indiquant qu'il n'est pas possible d'ajouter l'hôte à la liste des hôtes connus. Ignorez et tapez "yes" pour continuer.

Un mot de passe est ensuite demandé pour établir la connexion à distance. Tapez le mot de passe suivant pour accéder au compte "heiguser" (compte root) :

```
groupe-infra-pass
```

3.4.2 Accès à l'interface graphique du LoRa Server

Ouvrez votre navigateur et accéder au lien suivant :

https://iot_lora.lan.iict.ch:8080

Votre navigateur vous dira qu'il ne fait pas confiance au certificat du serveur : ajoutez une exception pour continuer.

Il faut ensuite saisir l'utilisateur "admin" et le mot de passe "groupe-infra-pass-interface".

3.4.3 Lancement de la topologie LoRa depuis la VM

Dans le cas où la topologie docker ne serait pas lancée sur le serveur (impossibilité de se connecter à l'interface graphique du LoRa Server), il faut accéder à la VM via un terminal comme expliqué plus haut, au compte "heiguser" puis au dossier /home/heiguser/home/Documents/repos/loraserver. Tapez la commande suivante pour démarrer le LoRa Server :

```
$> docker-compose up
```

3.4.4 Accès à la gateway

Pour accéder à la gateway, il y a deux possibilités : soit on veut accéder à distance via l'interface graphique du LoRa Server, soit directement via un câble série ou HDMI.

Quelle que soit la méthode de connexion, le user et le mot de passe sont respectivement "pi" et "raspberry".

Câble HDMI

La connexion est aisée puisqu'il suffit simplement de connecter un câble HDMI au Raspberry faisant office de gateway et de le relier à un écran pour le output. Si le Raspberry est en train de tourner, l'image l'interface s'affiche automatiquement à l'écran.

Câble série

Avec le câble série, il faut être attentif au branchement suivant sur la gateway :

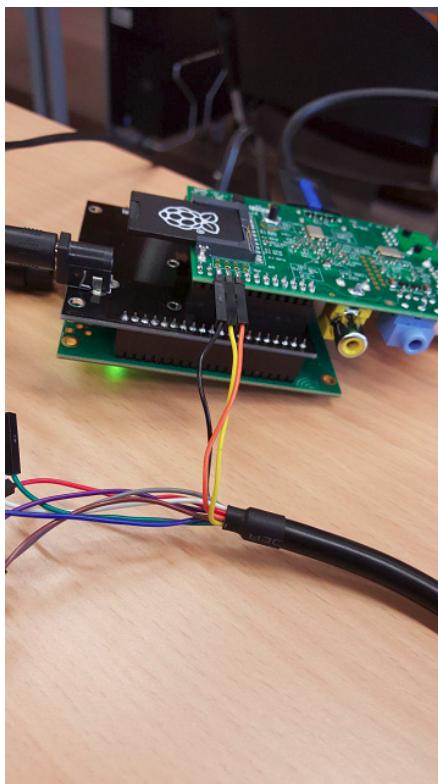


FIGURE 3.6 – Branchement de la Raspberry Pi en série

Branchez ensuite la partie USB sur votre machine et à l'aide d'un émulateur de terminal (PuTTY par exemple), spécifiez une connexion série avec un port 115200 et l'hôte COM qui correspond à celui configuré pour le périphérique USB de votre machine (peut être configuré sur le gestionnaire de périphériques).

Remarque : vous ne pourrez pas vous connecter en série si vous n'avez pas activé le SPI sur le Raspberry (voir installation gateway pour plus de détails).

Depuis le LoRa Server

En se connectant à l'interface graphique du LoRa Server (connexion nécessaire au réseau de la HEIG-VD) à l'adresse suivante https://iot_lora.lan.iict.ch:8080/ (user : admin, password : groupe-infra-pass-interface), cliquer sur l'onglet "Gateways", puis sélectionner "IotGateway" qui est la seule gateway ajoutée pour l'instant. Il est possible de consulter la localisation de cette gateway, ainsi que de configurer le LoRa network server auquel elle est liée.

3.4.5 Installation de la gateway

Cette section explique comment mettre en place une gateway à partir d'un Raspberry Pi modèle 2B.

Installation de l'OS Raspbian

La première étape consiste à installer un système d'exploitation sur la carte SD du Raspberry. L'OS choisi est décrit plus haut dans les spécificités.

Il faut ensuite insérer la carte SD dans le Raspberry et brancher l'alimentation pour le démarrer.

La procédure est vraiment très simple. Vous trouverez plus de détails à l'adresse <https://www.raspberrypi.org/learning/software-guide/>, notamment comment préparer la carte SD.

Installation LoRa-Gateway sur la gateway

Il faut désormais équiper le Raspberry du LoRa-gateway. Pour commencer, connectez la carte Lora (iC880A-SPI) au Raspberry.

Accédez au terminal du Raspberry, puis tapez la commande :

```
$> sudo raspi-config
```

Une interface de configuration va s'ouvrir. Il faudra activer le SPI en allant sur "Interfacing options". Ensuite, dans "Advanced options" choisissez "Expand filesystem".

Ces modifications seront prisent en compte lors du redémarrage.

Par la suite, il va falloir configurer la "time zone" :

```
$> sudo dpkg-reconfigure locales  
$> sudo dpkg-reconfigure tzdata
```

Installer git si ce n'est pas déjà fait :

```
$ sudo apt-get update  
$ sudo apt-get upgrade  
$ sudo apt-get install git
```

Installation de Lora Gateway :

1. Cloner le repo https://github.com/Lora-net/lora_gateway.git
2. \$> cd lora_gateway/
3. \$> make all
4. \$> ./reset_lgw.sh start

Installation du Packet Forwarder :

1. Cloner le repo https://github.com/Lora-net/packet_forwarder.git
2. \$> cd /packet_forwarder
3. \$> sudo ./compile.sh
4. \$> make all
5. \$> sudo nano /packet_forwarder/local_config.json

Puis, faire les modifications suivantes :

```

1 {
2 ...
3 "gateway_conf":{
4     "gateway_ID": B827EBFFFE1EE042
5 }
6 }
```

6. \$> sudo nano /packet_forwarder/global_config.json

Puis, faire les modifications suivantes :

```

1 {
2 ...
3 "gateway_conf":{
4     "gateway_ID": B827EBFFFE1EE042 ,
5     "server_address": "10.192.72.26",
6     "srrv_port_up": 1700,
7     "srrv_port_down": 1700,
8 ...
9 }
10 }
```

Lancement de la gateway :

```
$> cd packet_forwarder/lora_pkt_fwd/
$> sudo ./update_guid.sh ./local_config.json
$> cd packet_forwarder/lora_pkt_fwd/
$> sudo ./lora_pkt_fwd
```

La gateway devrait être opérationnelle à présent.

Ajouter la gateway au Network Server

Comme expliqué au chapitre "Accès à la gateway", il suffit de se connecter à l'interface graphique du LoRa server et accéder à l'onglet de configuration des gateways pour établir un lien avec un network server. Cependant, il faut créer au préalable un network server. Pour cela, il faut accéder à la page "Network Servers" puis cliquer sur "Add Network Server". Il faut spécifier le nom et l'adresse du network server. Actuellement, le network server en place possède le nom "infra-net-serv" et le hostname "loraserver :8000".

3.4.6 Connection entre le LoRa App Server et le front-end

Pour héberger une application frontend, une image appelée "passenger" est à disposition sur la VM du serveur de l'école : /home/Documents/passenger.

Dans ce dossier, un dockerfile permet de construire une image mettant en place un serveur web Nginx et définissant l'application à déployer et sa configuration.

L'application frontend à déployer se trouve dans /home/Documents/passenger/web-app et sa configuration dans /home/Documents/passenger/conf.

Dans le dockerfile, les lignes spécifiant le déploiement du frontend sont les suivantes :

```
COPY --chown=app:app web-app/helloWorld.js /home/app/webapp  
ADD conf/webapp.conf /etc/nginx/sites-enabled/webapp.conf
```

Remarque : ici, il s'agit d'un exemple avec une app "HelloWorld". Il faut la remplacer par l'app que l'on souhaite déployer.

L'image Docker "passenger" est donc indépendante du reste de l'infrastructure. Par conséquent, il faut build cette image et la run avec la commande suivante (attention à respecter le bon port-mapping) :

```
$> docker run -name passenger -p 3000:3000 infra/phusion-passenger
```

3.4.7 Connection entre le LoRa App Server et le back-end

Le backend possède sa propre infrastructure. Il faut cependant établir, sur le LoRa server, un lien entre le backend et le device (ici : un capteur).

Pour cela, il faut créer un "service-profile", un "device" et une "application". Tout cela est possible via l'interface graphique du LoRa server sous les différents onglets correspondants.

Il est à noter, qu'un "device" est défini par son nom, une description, son identifiant EUI et le "service-profile" utilisé. Actuellement le nom est "test-sensor-5", son EUI "0004a30b001a1e25" et son "service-profile" est "firm-dev-profile". Le "service-profile", défini un état de communication en spécifiant par exemple le newtwork server (infra-net-serv) utilisé ou la fréquence des requêtes de status du "device". L'application est définie par son nom est le "service-profile" utilisé (ici respectivement "backend" et "firm-dev-profile").

3.4.8 LoRa Server REST API

Il est possible de communiquer avec le LoraServer grâce à une API REST détaillée à l'adresse suivante : https://iot_lora.lan.iict.ch:8080/api

3.5 Conclusion

L'infrastructure est en place et fonctionne. On constate que la connexion fonctionne entre la gateway et le device. Voici le trafic reçu par la gateway :

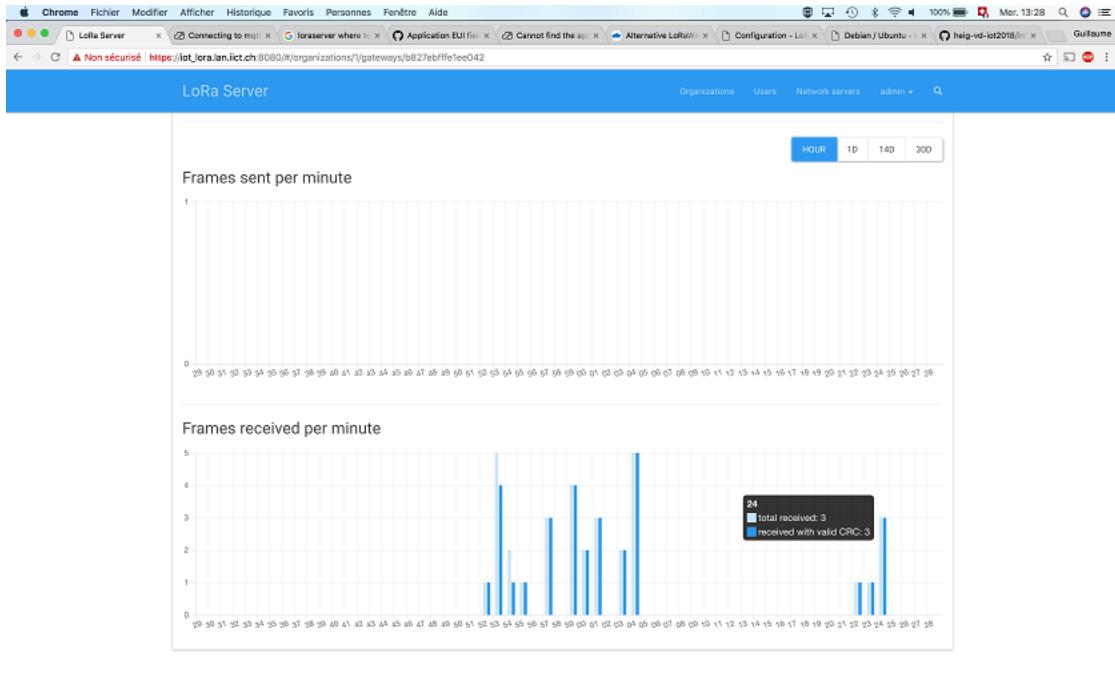


FIGURE 3.7 – Trafic entrant dans la gateway

En ce qui concerne la communication entre le device et le LoRa server, voici ce qui est reçu :

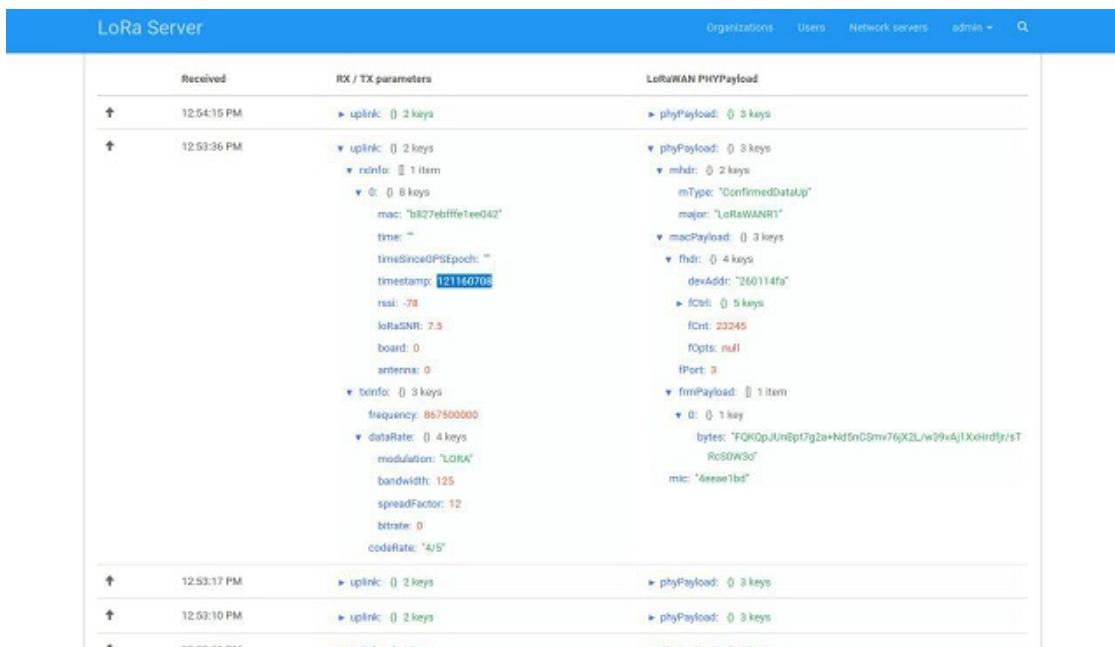


FIGURE 3.8 – Communication entre device et LoRa server

3.6 Documentation supplémentaire

- loraserver.io website
- Network documentation on TTN website

Chapitre 4 : Partie back-end

4.1 Introduction

Le back-end est responsable de faire le lien entre les capteurs (qui nous fournissent les données en passant par le serveur LoRa de la partie "Infrastructure"), de traiter/normaliser les données et être capable de les fournir au front-end. De plus, le front-end doit être capable de changer l'intervalle de rafraîchissement du capteur.

4.2 Technologies utilisées

Les technologies utilisées sont les suivantes :

- [NodeJS](#) - Le cœur du système.
- [Swagger](#) - Permet de gérer les endpoints.
- [MongoDB](#) - Permet de stocker les données de façon persistante.
- [Docker](#) - Pour gérer le déploiement

4.3 Spécificités

Voici l'infrastructure du back-end :

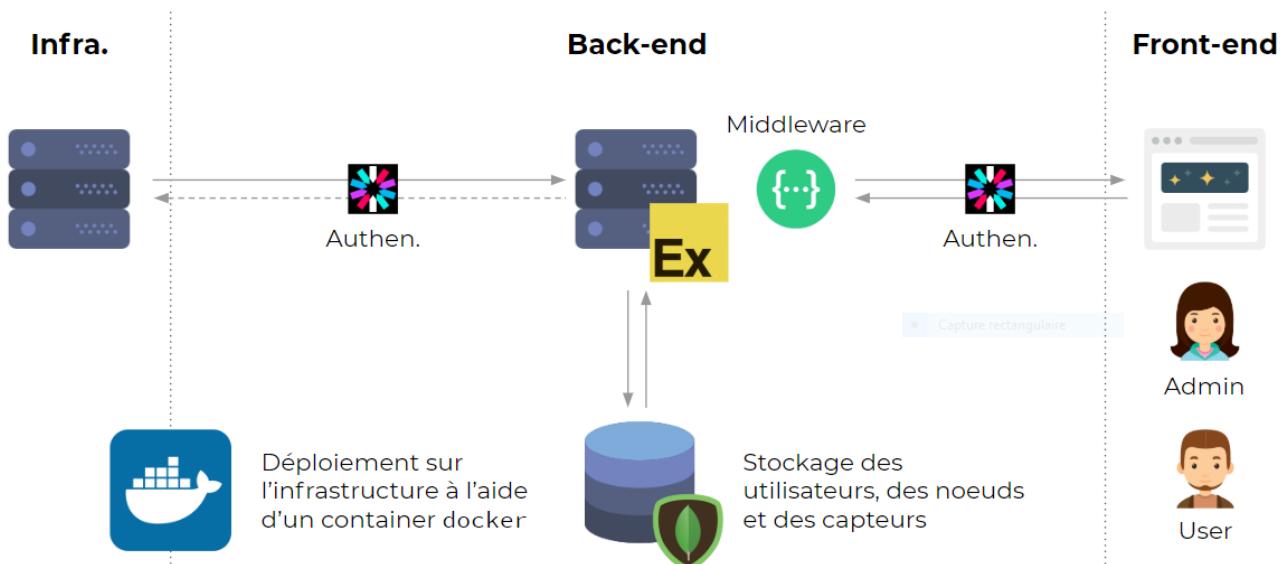


FIGURE 4.1 – Infrastructure du back-end

4.3.1 Contraintes

Le back-end doit mettre à disposition une API permettant à un client de récupérer les informations des différents capteurs. Voici les différentes contraintes de ce dernier :

- Le back-end doit savoir quel est le format envoyé de la part de chacun des capteurs.
- Le back-end doit pouvoir traiter et normaliser les données issues des capteurs.
- Le back-end doit pouvoir fournir les données des capteurs par noeuds (regroupement de plusieurs capteurs) ou par capteurs uniques.
- Le back-end doit pouvoir modifier le taux de rafraîchissement des capteurs.
- Le back-end doit permettre de pouvoir gérer des droits d'accès sur les endpoints.

4.3.2 Authentification

Afin de garantir que seuls les utilisateurs autorisés puisse accéder/utiliser l'API, des **JSON Web Tokens** (JWT) seront utilisés et devront être utilisés lors des communications afin de savoir si un client est authentifié et autorisé à accéder à la ressource souhaitée.

Obtenir un JWT

Un JWT peut être obtenu en se connectant avec un nom d'utilisateur et un mot passe valide. Pour ce faire il faut effectuer une requête HTTP POST sur le endpoint `/auth`. En paramètre, dans le body de la requête, il faut fournir un payload json contenant les informations mentionnées ci-dessus.

```
1 {
2     "username": "user",
3     "password": "user1234"
4 }
```

Si l'utilisateur existe et que le mot de passe est valide, l'API REST retournera un réponse au format `text/plain` et le JWT généré se trouvera dans le body de la requête.

Accéder à un endpoint protégé

Pour accéder à un endpoint protégé, il faudra ajouter le header `Authorization` à vos requêtes avec comme valeur `Bearer <JWT>` ou `<JWT>` et à remplacer par le token récupéré à l'étape précédente.

Si l'utilisateur pour lequel ce JWT a été généré a les droit d'accès (autorisation) au endpoint au question, la requête sera authentifiée et autorisée. Les deux rôles existants sont **administrateur** ou **utilisateur** par défaut.

Pour plus de détails, notamment sur les contraintes de validation du payload, consulter la [spécification de l'API REST](#).

Utilisateurs existants par défaut

Deux utilisateurs existants sont créés par défaut lors du déploiement de l'API sur un serveur. Par défaut et en mode développement ces utilisateurs sont :

Username	Password	Role
user	user1234	utilisateur par défaut
admin	admin1234	administrateur

Attention à ne pas garder ces mots de passe et noms d'utilisateurs lors du déploiement. Se référer à la section déploiement pour plus d'information.

4.3.3 Représentations

Les différents éléments du monde réels sont représentés de la façon suivante :

- Un nœud est un regroupement de capteurs physiquement placés au même endroit.
- Un capteur comporte, à première vue, un capteur physique et le module de communication de LoRa.
- Un capteur a un identifiant unique qui lui est associé et que l'on peut récupérer pour identifier précisément le capteur.

4.3.4 Éléments stockés dans la base de données

Deux principales collections seront stockés dans la base de donnée MongoDB :

- La collection `Users` qui contiendra les utilisateurs autorisés à accéder à l'API.
 - Dans le cadre de ce projet, chaque utilisateur sera stocké en dur.
 - Lorsqu'un utilisateur est authentifié, il se verra attribuer un identifiant unique qu'il devra utiliser à chaque communication avec le serveur qui l'autorisera à accéder aux endpoints. Voir le chapitre Authentification pour plus de détails.
 - La structure de l'objet est la suivante :
 - `username` - Le nom de l'utilisateur
 - `password` - Le mot de passe (stocké de façon sécurisée)
 - `role` - le rôle (lié aux droits d'accès de l'utilisateur) [0 : utilisateur normal | 1 : administrateur]
- La collection `BlacklistedTokens` qui contiendra les tokens qui ne sont plus autorisés à accéder à l'API.
 - Dans le cadre de ce projet, cette collection ne sera pas nettoyée automatiquement.
 - La structure de l'objet est la suivante :
 - `blacklisted_token` - Le token banni.

- La collection **Sensors** qui contiendra toutes les descriptions des différents capteurs.
 - La structure de l'objet est la suivante :
 - **id** - Identifiant unique du capteur (récupéré par le LoRa serveur).
 - **documentationLink** - Lien vers la documentation officielle du constructeur.
 - **dateCreated** - La date à laquelle l'objet a été créé.
 - **dateUpdated** - La dernière date de mise à jour de l'objet.
 - **active** - Permet de savoir si le capteur est encore actif ou s'il a été désactivé.
 - **refresh_interval** - Fréquence à laquelle le capteur doit fournir ses mesures.
- La collection **Nodes** qui contiendra toutes les descriptions des différents capteurs situés aux même endroit.
 - La structure de l'objet est la suivante :
 - **id** - Identifiant unique du noeud.
 - **dateCreated** - La date à laquelle l'objet a été créé.
 - **dateUpdated** - La dernière date de mise à jour de l'objet.
 - **active** - Permet de savoir si le noeud est encore actif ou s'il a été désactivé.
 - **latitude** - La latitude du noeud.
 - **longitude** - La longitude du noeud.
 - **sensors** - Tableau d'entiers correspondant aux identifiants des capteurs regroupés dans le noeud.

4.3.5 Endpoints

Les différents endpoints et leurs définitions sont décrits [ici](#) et sont générés à l'aide du module [NPM swagger-markdown](#).

La définition des endpoints sera régulièrement mise à jour.

4.4 Déploiement

Pour le déploiement, une configuration Docker a été créée. Les instructions sont disponibles [ici](#).

4.5 Conclusion

Le back-end a réussi à être connecté au front-end. Des données mock-up ont été créées pour permettre au front-end d'afficher des données même si la connexion au serveur LoRa n'est pas une réussite que nous ne parvenons pas à obtenir des informations en provenance de vrais capteurs.

4.6 Documentation supplémentaire

Toutes les librairies utilisées pour ce projet sont disponibles dans le [package.json](#).

Chapitre 5 : Partie front-end

5.1 Introduction

Ce bout de projet est consacré à la partie front-end. Il consiste en la mise en place d'une interface utilisateur permettant de voir les différentes valeurs fournies par les capteurs. Les données ont été traitées de manière à présenter des informations compréhensibles et utilisables. Divers graphiques représentent donc les résultats afin de les présenter visuellement. Une carte représente les emplacements des différents groupes de capteurs (température, humidité, particules fines,...). Un clic suffit à l'ouverture des informations liées au groupe choisi.

Vous pouvez atteindre l'application sur ce lien : [IOT - project link](#)

5.2 Technologies utilisées

- [React](#)



Il s'agit d'une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page (ou portion) HTML à chaque changement d'état. Nous avons décidé d'utiliser cette technologie car Matthias la connaît très bien. De cette manière, nous avons pu gagner énormément de temps, car React permet de créer des composants de manière rapide et propre. Les dits composants utilisent des données entrantes qui peuvent changer au cours du temps, permettant ainsi une gestion dynamique de leur affichage.

- [D3](#)



Il s'agit d'une bibliothèque graphique JavaScript qui permet l'affichage de données numériques sous une forme graphique et dynamique. Nous avions déjà appris à l'utiliser en cours, ce qui nous avait permis de constater son utilité. Cette bibliothèque permet en effet de créer des graphiques recherchés et facilement lisible, de manière à représenter les données sous une forme propre et utilisable. Comme de nombreux codes de graphique sont déjà mis à disposition en libre utilisation, il est facile de créer un graphique et de le modifier afin d'obtenir ce qui est nécessaire à la partie front-end.

- Mapbox.js



Il s'agit d'une bibliothèque JavaScript, basée sur [Leaflet](#), qui permet l'affichage de cartes interactives et dynamiques. Cette librairie est relativement bien documentée et permet de créer une carte avec différentes caractéristiques, permettant par exemple de limiter le terrain visible ou encore le zoom arrière-avant disponible.

Le projet est build avec [webpack](#) qui aide à organiser l'application en modules

5.3 Spécificités

5.3.1 Contraintes

Le front-end doit mettre à disposition différentes fonctionnalités selon le rôle de l'utilisateur connecté. Deux types d'utilisateurs existent : admin et lambda. Dès que quelqu'un souhaite avoir accès à l'application, il doit se connecter avec son mot de passe. Pour obtenir les accès de connexion, veuillez vous référer à la documentation du backend.

admin

L'utilisateur admin a tous les droits. Il a donc accès à toutes les fonctionnalités de l'application. En plus des pages que l'utilisateur lambda peut voir, il peut aussi modifier le temps de rafraîchissement des capteurs.

lambda

L'utilisateur lambda a accès aux pages permettant de visionner les données et les graphiques.

5.3.2 Pages

Plusieurs pages sont disponibles, chacune correspondantes à une fonctionnalité. La première page accessible est la page de login. Une fois ceci fait, une deuxième page permet de voir une carte où se trouvent des points, lesquels représentent la position de chaque groupe de capteurs. Au clic de l'un de ces points, il est possible d'atteindre une troisième page présentant les données liées aux capteurs visés. De cette manière, il est possible d'observer séparément les capteurs selon leur emplacements.

5.3.3 Endpoints

Pour plus d'informations sur les endpoints, veuillez vous référer aux documents de la partie backend (chapitre 4).

5.3.4 Visuel

Le visuel de l'application met à disposition trois pages principales, dont voici la description.

Login

La page de login présente un formulaire de connexion. Deux types d'utilisateurs existent : **admin** et **lambda**. Pour créer un nouvel utilisateur, il faut le faire directement depuis la base de données. Le passage par la page de login est obligatoire. L'application n'est pas utilisable tant que l'utilisateur n'est pas connecté.

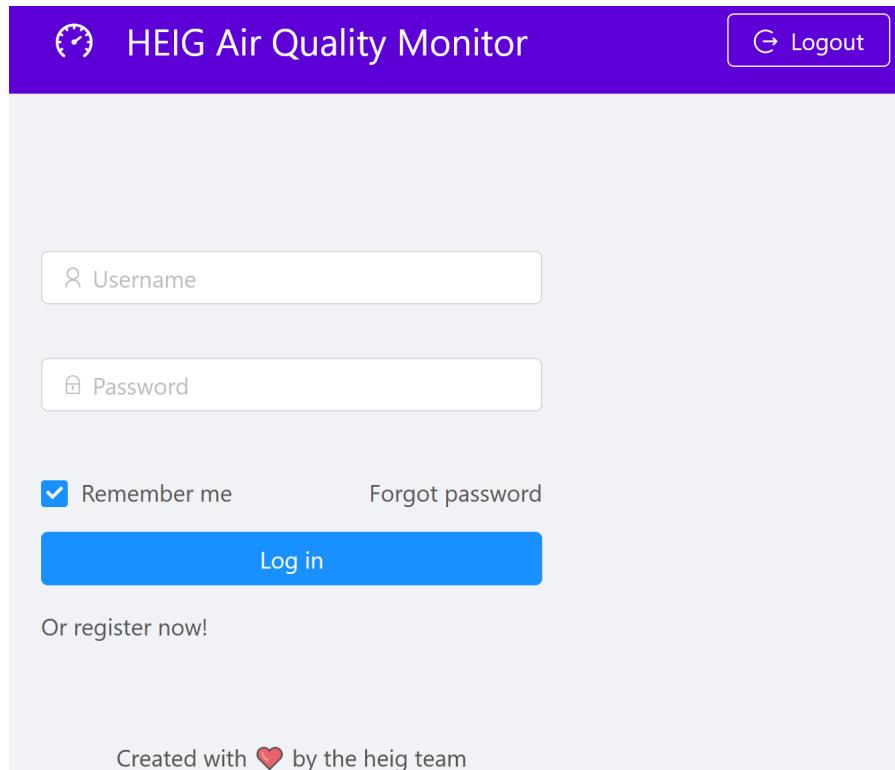


FIGURE 5.1 – Aperçu de la page de connexion

Carte

Une page présente une carte avec des points représentant les groupes de capteurs. Ces points sont sur l'emplacement réel des capteurs. Chacun de ces repères est cliquable. Cette action redirige l'utilisateur sur la page des capteurs du groupe lié au noeud choisi.

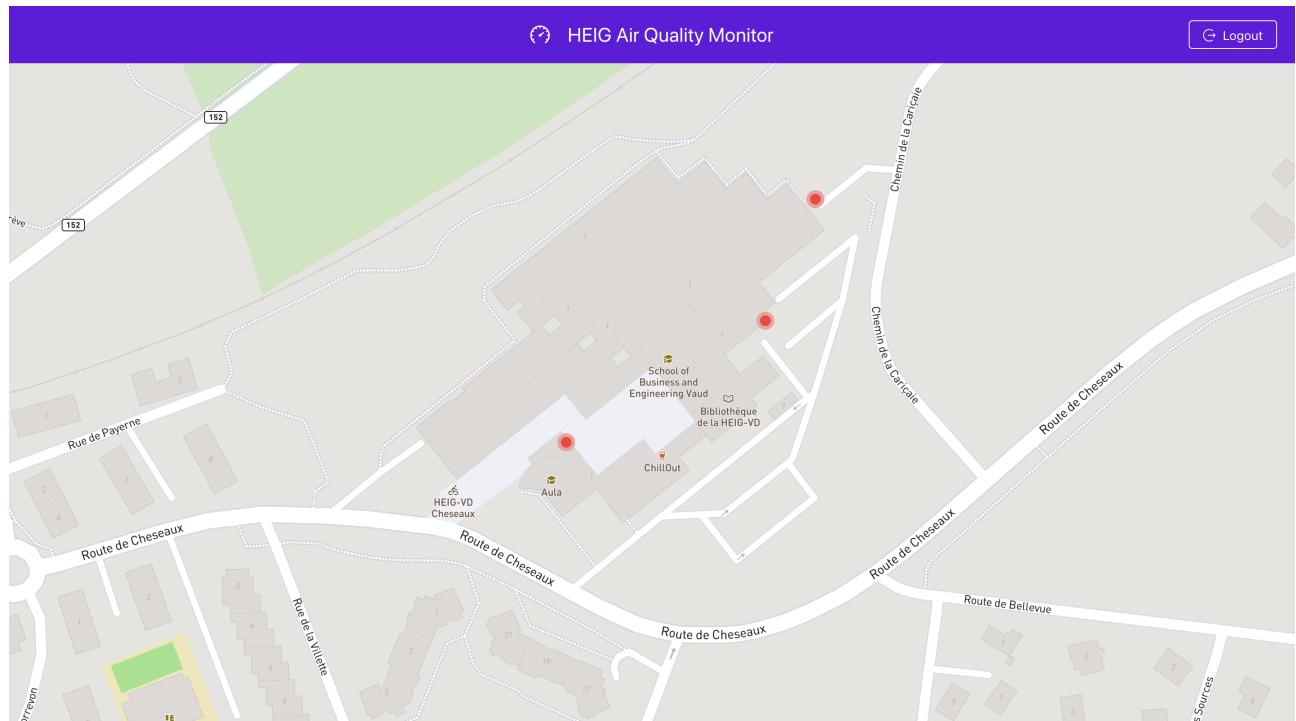


FIGURE 5.2 – Aperçu de la page carte

Capteurs

Chaque page des capteurs (atteignable par un clic sur un repère de la carte) présente plusieurs informations. La première partie (en haut de la page) présente les dernières valeurs relevées pour les capteurs, sous forme de jauge. La partie du bas, quant à elle, présente les différents graphiques par type de données. Ces graphiques permettent d'observer les changements de valeurs sur une durée relativement longue.

Voici un exemple concernant les jauge susmentionnées :

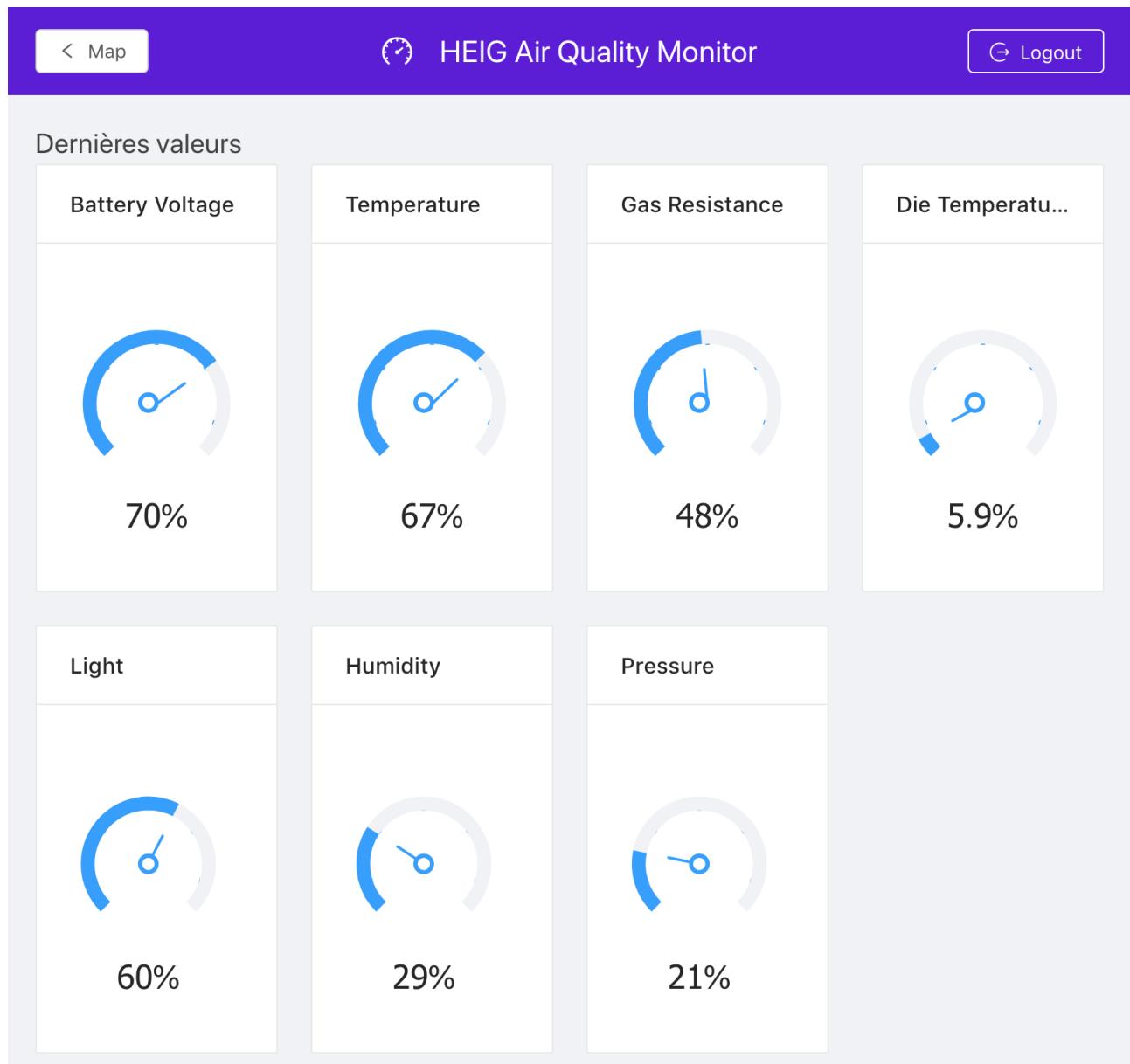


FIGURE 5.3 – Aperçu de la page d'un capteur avec les jauge

Et voici un exemple concernant les graphiques "normaux" :



FIGURE 5.4 – Aperçu de la page d'un capteur avec les graphiques classiques

Concernant ces derniers graphiques, les données affichées concernent les mêmes données que pour les jauge. Cependant, la durée sur laquelle s'étendent les données sont bien plus longues.

5.4 Déploiement

Le projet s'exécute au travers d'une image docker que nous vous fournissons.

- build avec `docker build -t front-end` .
- run avec `docker run -p 80:80 -d front-end`

5.4.1 En mode développement

Vous pouvez l'exécuter avec les commandes suivantes :

- `webpack dev => yarn run webpack`
- `webpack build => yarn run build`

5.5 Architecture

Les dossiers les plus importants sont :

- **src** => fichiers sources du front-end
- **server** => fichiers du serveur
- **build** => configuration webpack
- **app** => front-end buildé

5.6 Problèmes rencontrés et difficultés à prendre en compte

Un des problèmes principaux rencontré a été le manque de temps. En effet, le temps alloué à ce projet était relativement faible, ce qui fait que le visuel est simpliste.

5.7 Conclusion

Comme nous n'avons pas eu beaucoup de temps, le visuel est améliorable. Nous sommes restés relativement basiques, et nous avons fait le minimum de pages possibles. Une des améliorations possibles à ce sujet est de mieux diviser les résultats, afin de les rendre plus lisibles.

D'autres améliorations peuvent aussi être faites :

- Possibilité de s'inscrire à l'application
- Représentation plus précise de la carte : par exemple, permettre l'utilisation de cartes personnalisées (bâtiments au lieu du monde)
- En passant la souris sur les points, afficher les informations importantes (par exemple, les dernières valeurs des capteurs)
- Tout ce qui est configuration de visuel : choix de la date des données affichées (par exemple, du 20.02.2018 au 25.02.2018), etc.

5.8 Liens utiles

- React : <https://reactjs.org/>
- D3 : <https://d3js.org/>
- Mapbox : <https://www.mapbox.com/mapbox.js/api/>