

Passerelle industrielle 4.0

(partie commune)



BTS SN2 Option IR
Année scolaire 2023/2024

Lycée Lafayette
21 Bd Robert Schuman, 63000 Clermont-Ferrand

SOMMAIRE

SOMMAIRE	2
1. Présentation du système	3
2. Description du système	5
2.1 Cahier des charges / Applications utilisées	5
2.2 Matériel utilisé	5
2.3 Coût total du projet	8
3. Répartition des tâches et rôles du groupe	9
3.1 Elève n°1 (Clement)	9
3.2 Elève n°2 (Christelle)	9
3.3 Elève n°3 (Amine)	10
4. Objectifs du projet	11
5. UML	12
5.1 Diagramme des cas d'utilisation	12
5.2 Diagrammes de séquence	13
5.3 Diagramme des classes	15
5.4 Diagrammes de déploiement	16
6. Organisation du groupe	17
6.1 Planning prévisionnel	17
6.1.1 Planning prévisionnel de Clément	18
6.1.2 Planning prévisionnel de Amine	19

Présentation du système

Grâce aux percées dans les domaines des technologies de l'information, des communications mobiles et de la robotique, les technologies numériques sont de plus en plus utilisées dans les entreprises du monde entier.

Il s'agit d'une véritable révolution industrielle, la 4e , après celle de la mécanisation, celle de la production de masse et celle de l'automatisation.

Avec l'industrie 4.0 ou « usine connectée » l'industrie devient, grâce à l'arrivée de la numérisation, un système global interconnecté dans lequel les machines, les systèmes et les produits communiquent en permanence.

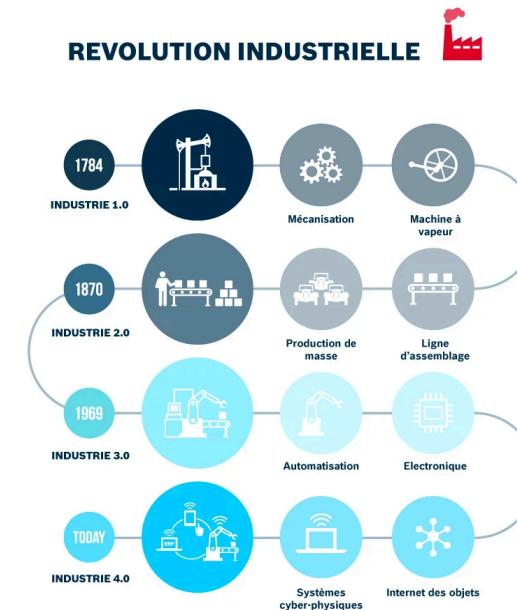
L'industrie 4.0 consiste donc à surveiller et à contrôler en temps réel les machines et les équipements en installant des capteurs à chaque étape du processus de production.

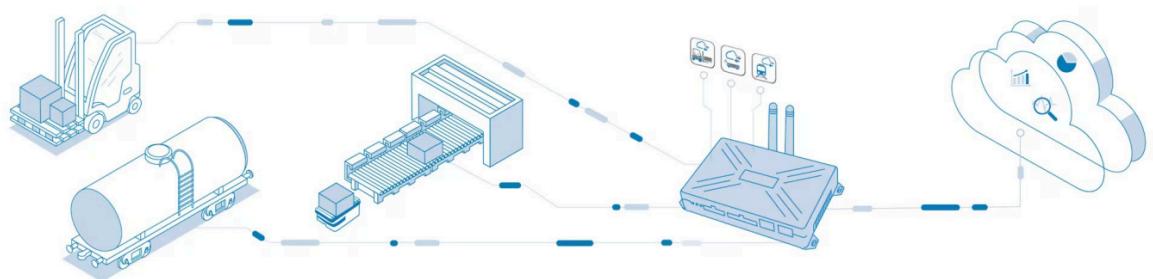
Afin de surveiller et piloter l'usine connectée, une solution consiste à rendre possible l'accès à ces données sur le réseau informatique de l'entreprise.

Il faut cependant tenir compte de la cybersécurité et donc ajouter une couche de protection.

La passerelle industrielle 4.0 met en réseau les capteurs, les machines et les plateformes IoT pour la collecte et le prétraitement de données de capteur et de processus locaux.

Elle interconnecte l'usine connectée au réseau de l'entreprise.





Description du système

2.1 Cahier des charges / Applications utilisées

La passerelle est interconnectée à deux réseaux, un côté machines industrielles et l'autre côté réseaux de l'entreprise.

Un firewall sera mis en place.

Les capteurs et actionneurs pourront utiliser différents protocoles (Ethernet TCP, ModBus TCP, WIFI etc.).

La programmation des automatisations et la visualisation de données utilisera Node Red.

Une sauvegarde dans une base de données permettra une exploration historique des données sur 30 jours.

Un serveur web intégré permettra l'observation des données.

Le paramétrage et la visualisation pourront se faire sur un écran tactile intégré.

Une API REST sera implantée pour la gestion de la passerelle.

Pour mettre en place ce système nous utiliserons les logiciels phpMyAdmin pour la création et la gestion de base de données, Apache 2 pour le côté serveur, QT afin de créer l'API REST, Visual Studio Code pour la création du site web et Node Red afin de gérer et piloter les capteurs.

Matériel utilisé

Le système sera composé d'une carte Raspberry PI 4 avec une extension USB-Ethernet, un écran ainsi que un OS adapté au microcontrôleur



Pour les mesures nous utiliserons des capteurs IFM et un maître IO-Link.

- Compteur d'air comprimé SD5500 :



<https://www.ifm.com/fr/fr/product/SD5500>

Ce capteur permet de mesurer la pression (en bar), le débit (en m³) et la température (en °C).



- Maître IO-Link avec interface PROFINET AL1302 :

<https://www.ifm.com/fr/fr/product/AL1302?tab=documents>

Le maître IO-Link, ou IO-Link Master, sert à connecter des appareils de terrain IO-Link comme des capteurs et des actionneurs.

Pourquoi utiliser IO-Link ?

La technologie IO-Link possède de nombreux avantages qui nous paraissent importants à exploiter.

Pour commencer, qu'est ce que IO-Link ?

IO-Link est une interface de communication point à point permettant aux capteurs et actionneurs de dialoguer avec les systèmes de commandes. Apparue à la fin des années 2000, il s'agit de la première technologie d'entrées/sorties normalisée, répondant à la norme

IEC 61131-9. Un système IO-Link se compose en général d'un moins un module "Maître" raccordé à des périphériques p e. Les appareils IO-Link peuvent par exemple être raccordés au moyen d'un simple connecteur M12.

Et voici donc notre premier avantage : les dysfonctionnements dus à des câblages incorrects sont donc exclus.

De plus, ce dispositif permet de réduire les coûts machines. Grâce à IO-Link l'entreprise peut réaliser des économies sur la maintenance, mais aussi sur les coûts d'ingénierie et de documentation. Il favorise aussi une réduction des stocks grâce à des appareils multi-usages intelligents

IO-Link va aussi numériser les communications des capteurs et actionneurs. IO-Link permet de passer facilement de l'analogique au numérique avec une connectique standard. Les équipements sont ainsi raccordés via un simple câble non blindé, ce qui facilite et réduit les coûts d'installation.

IO-Link permet aussi de gagner du temps en pilotant à distance les équipements. Les réglages des périphériques peuvent être modifiés à distance, sans intervenir physiquement sur site et sur l'ensemble des équipements simultanément.

Pour finir, cet outil permet de gagner en productivité et fiabilité. Grâce aux données transmises des diagnostics sont effectués. IO-Link permet de réduire les risques de pannes, de planifier plus facilement les opérations de maintenance et ainsi de réduire les temps d'arrêts inopinés. Un port IIOT (Internet Industriel des Objets ou en anglais "Industrial Internet of Things") est d'ailleurs dédié aux diagnostics.

Coût total du projet

Matériel	Cout
Raspberry PI4	63€
écran de 7 pouce	50€
adaptateur USB-Ethernet	15€

Le projet a donc un coût total de 128€ dans ce calcul nous ne comptons pas les capteurs que l'on nous a fournis. Les capteurs seront présent sur site.

Répartition des tâches et rôles du groupe

Différentes fonctionnalités ont été attribuées à différents étudiants en suivant le cahier des charges.

Elève n°1 (Clement)

Fonctionnalités en charge :

- Installation et paramétrage de la carte Raspberry.
Il faudrait donc installer un système d'exploitation compatible avec la carte, une extension pour avoir deux cartes réseaux ainsi qu'un écran tactile.
On utilisera un écran 7 pouces.
- Installation et paramétrage du Firewall Installation et paramétrage du Broker MQTT Mosquitto.
- Installation et paramétrage de Node Red.
- Création de l'application de gestion sur écran tactile.
- Création du boîtier pour la carte ainsi que l'écran.

Elève n°2 (Christelle)

Fonctionnalités en charge :

- Installation et paramétrage de la base de données.
Une base de données sera créée afin de stocker les données reçues par les capteurs.
- Sauvegarde des données sur 30 jours.

- Affichage des données issues des capteurs.
Les données recueillies devront être affichées sur un site web accessible.
- Pilotage des actionneurs.
Il faudra pouvoir modifier les réglages des capteurs à distance grâce au site web.

Elève n°3 (Amine)

Fonctionnalités en charge :

- Création de l'API REST pour la récupération des données et la commande :
 - Le but est de créer une API en PHP avec l'architecture REST pour pouvoir récupérer les données des capteurs et les afficher sous forme de JSON via le protocole HTTP.
- Création d'une application de démonstration utilisant l'API :
 - Il va falloir créer une interface et un programme sous QT Creator pour pouvoir lire les données récupérées sous format JSON généré par l'API.

Objectifs du projet

Ce projet à pour objectif d'apporter aux techniciens une vue simplifiée de l'automate utilisé.

La passerelle industrielle 4.0 permet de mesurer différentes données grâce à des capteurs afin de faciliter la visualisation et l'utilisation des données.

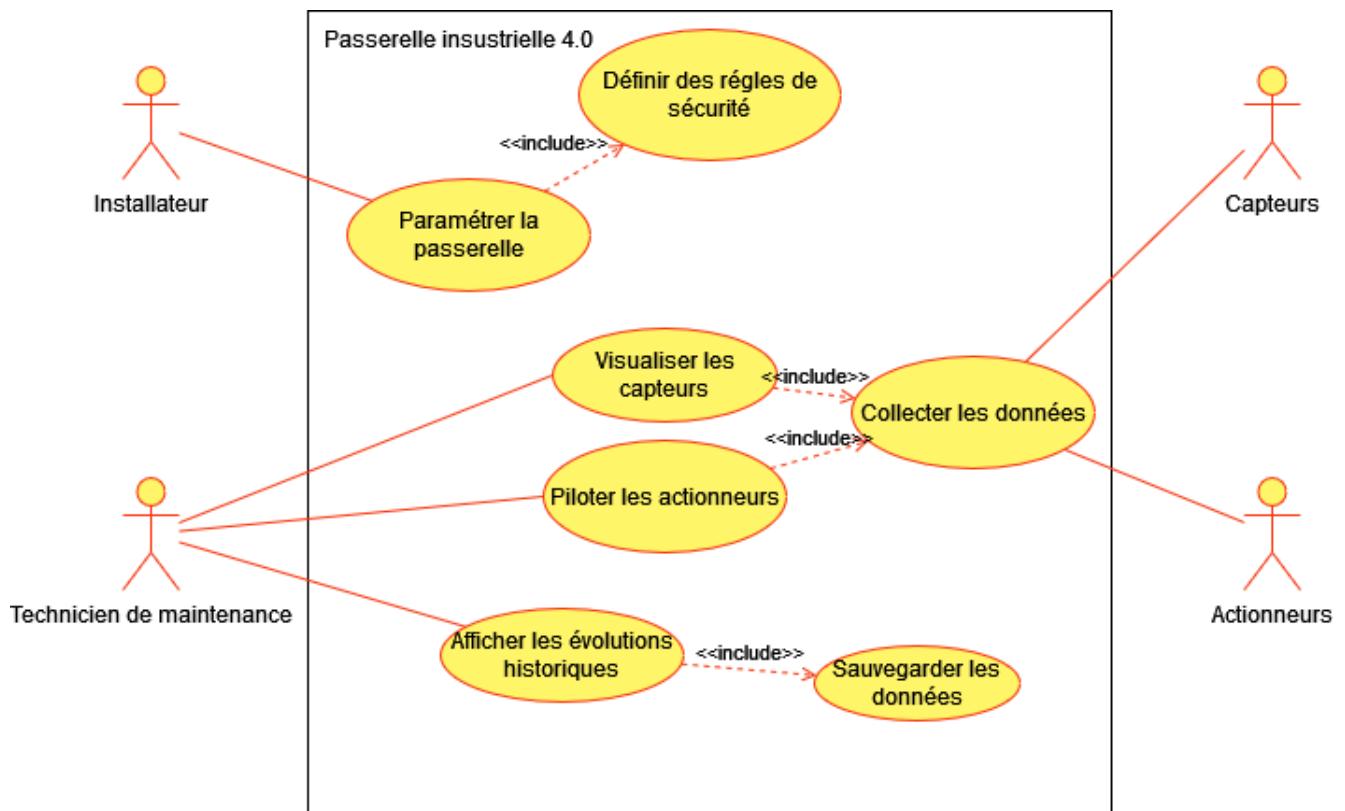
Le site web offre une vision simplifiée de toutes ces données sous forme de liste, de widget et de graphiques ce qui permet une lecture et une analyse plus rapide.

De plus, le site permet le paramétrage de certaines fonctions en temps réel et à n'importe quel endroit ce qui est beaucoup plus simple et qui permet de gagner énormément de temps.

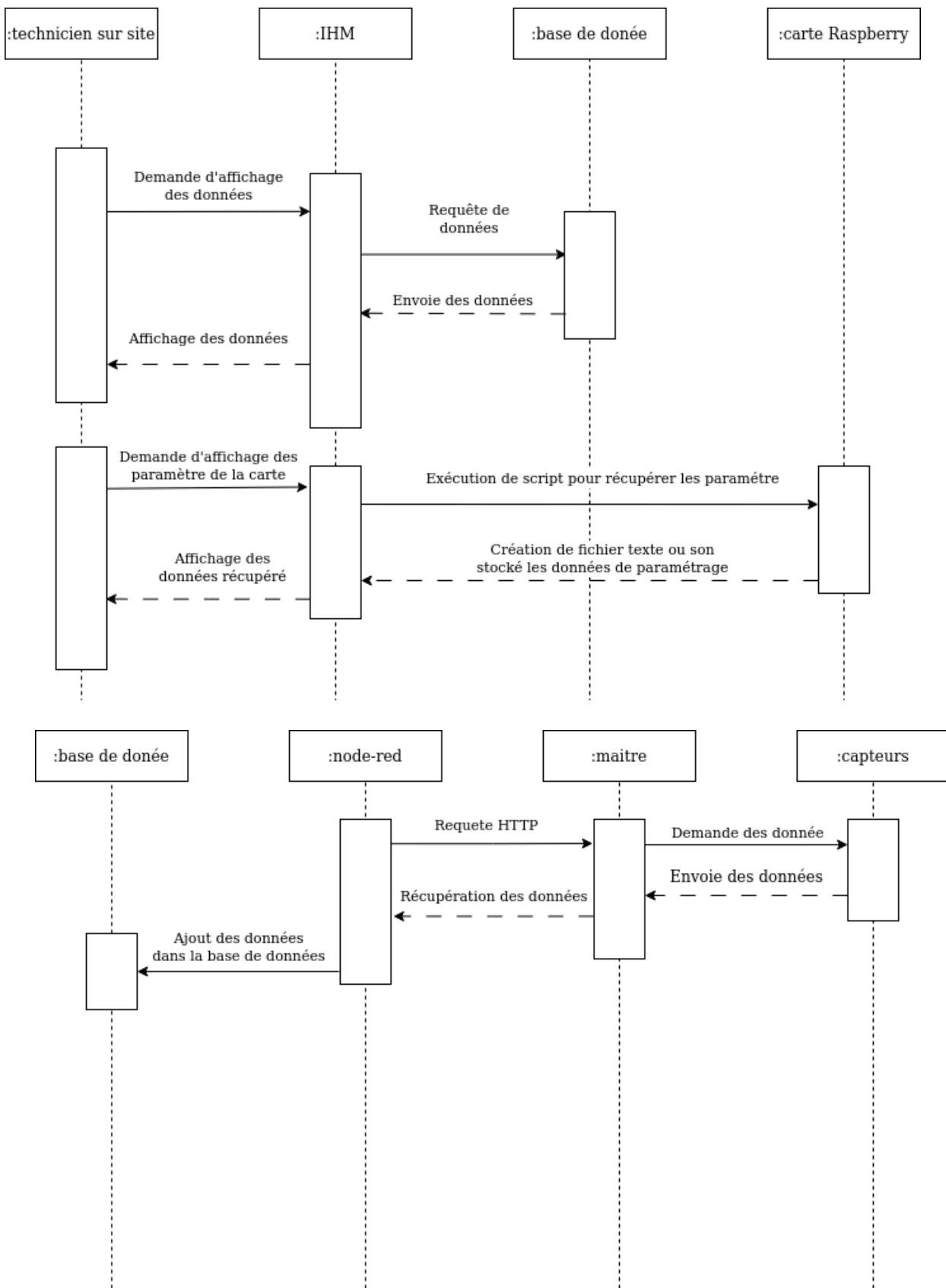
N'importe qui peut s'essayer à paramétrer sans avoir de connaissances spécifiques en informatique grâce aux widgets intuitifs.

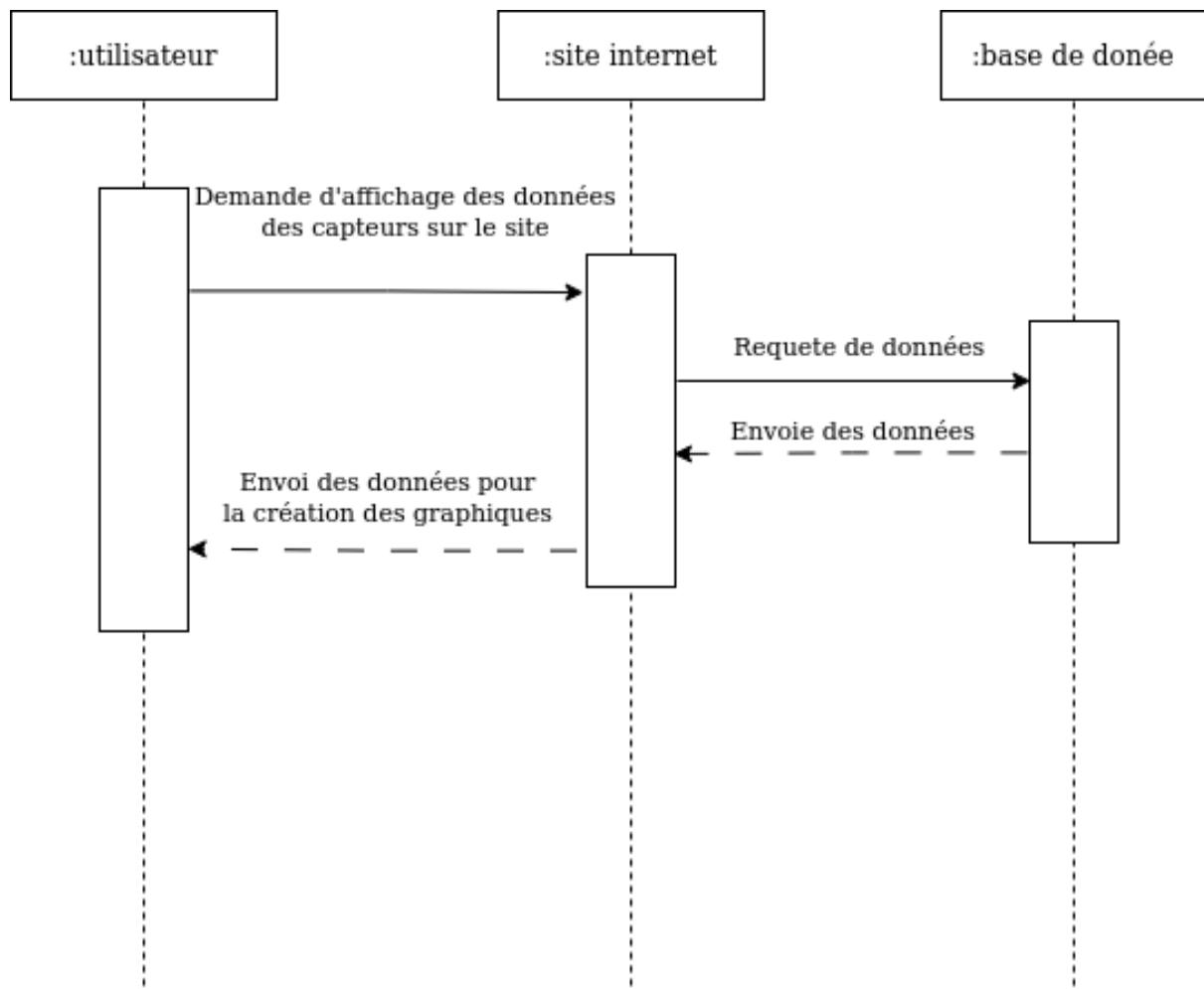
UML

Diagramme des cas d'utilisation



Diagrammes de séquence





Elève 3:

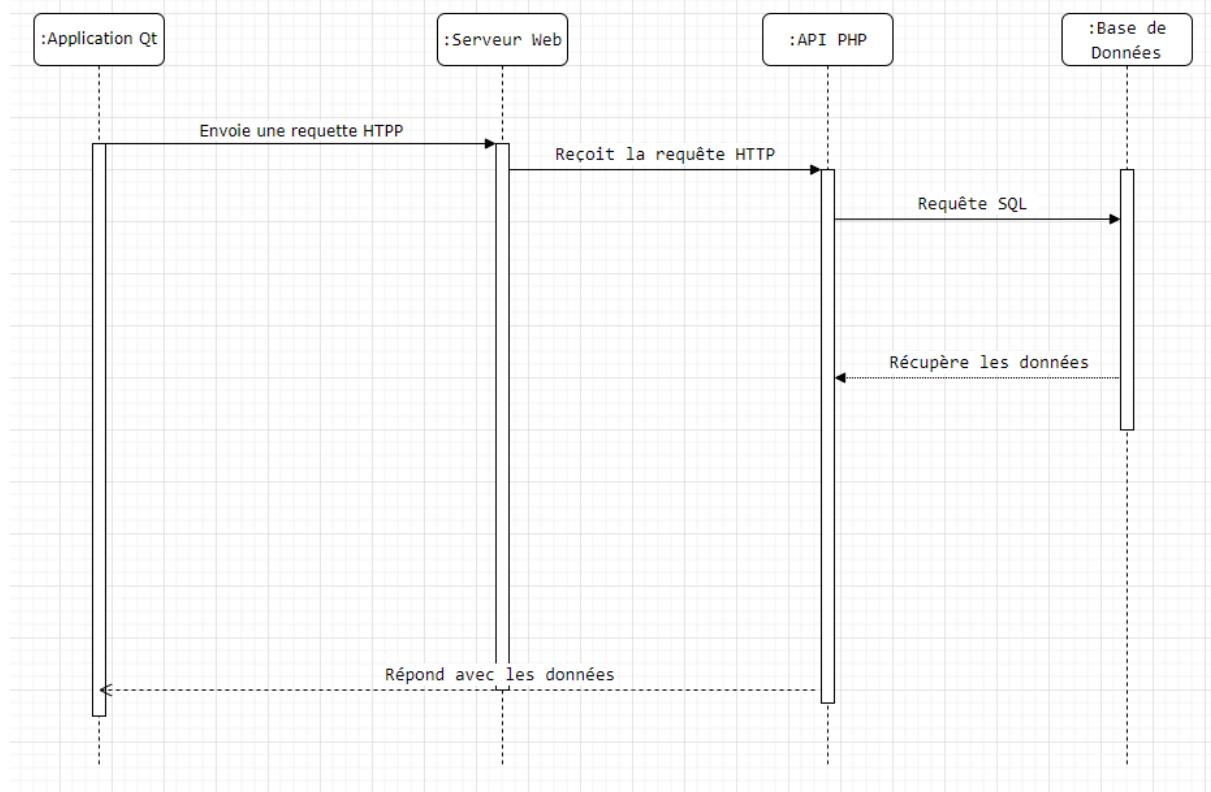
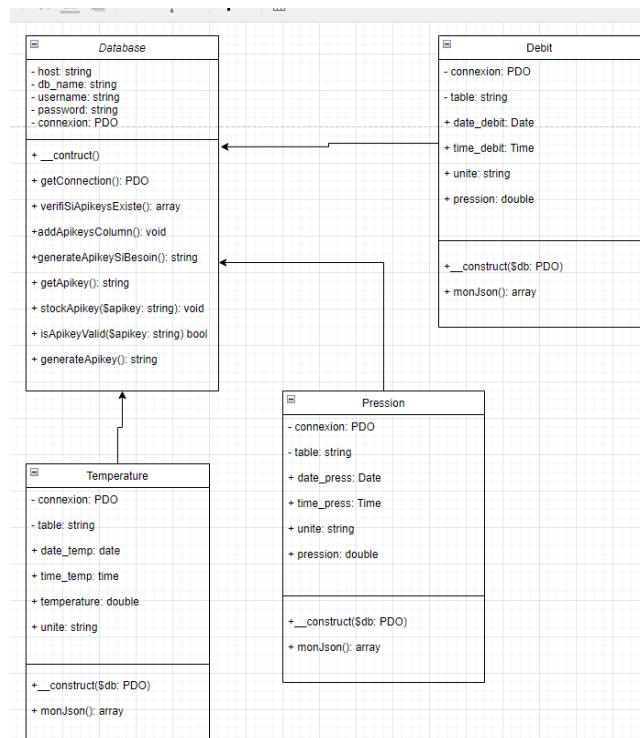
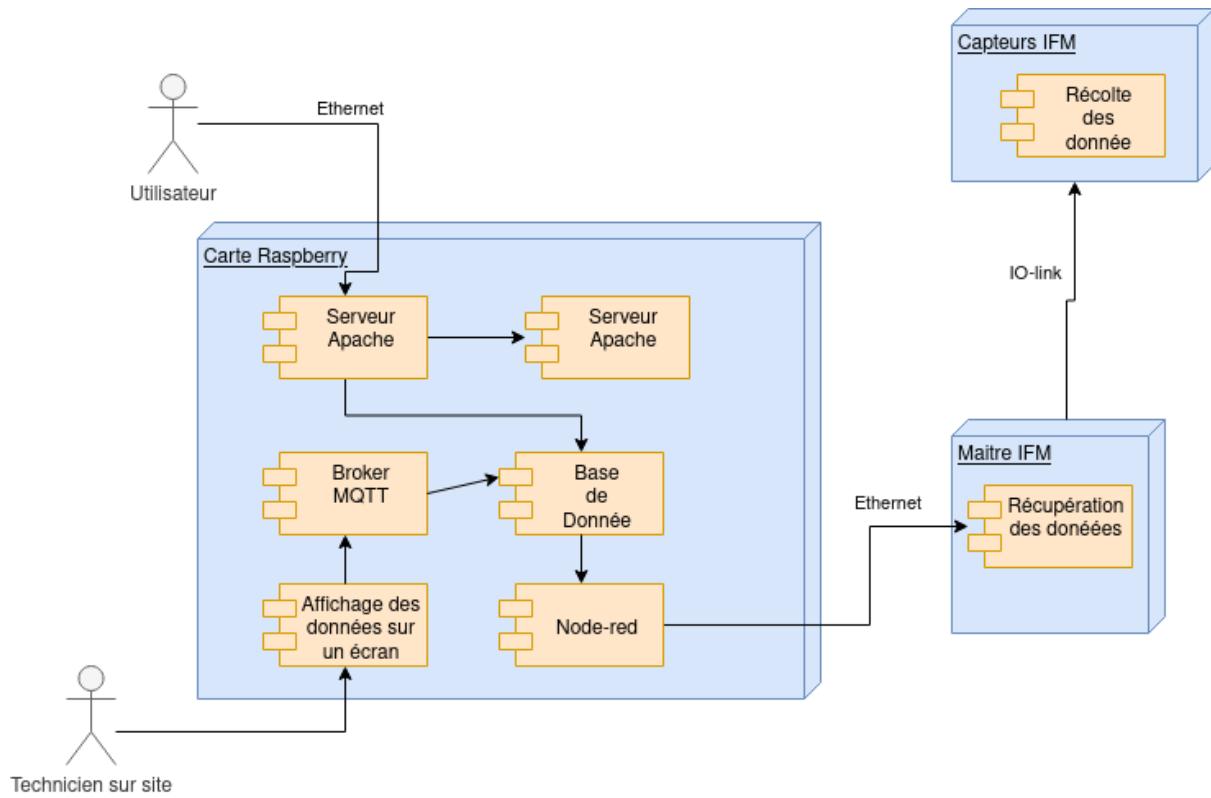


Diagramme des classes

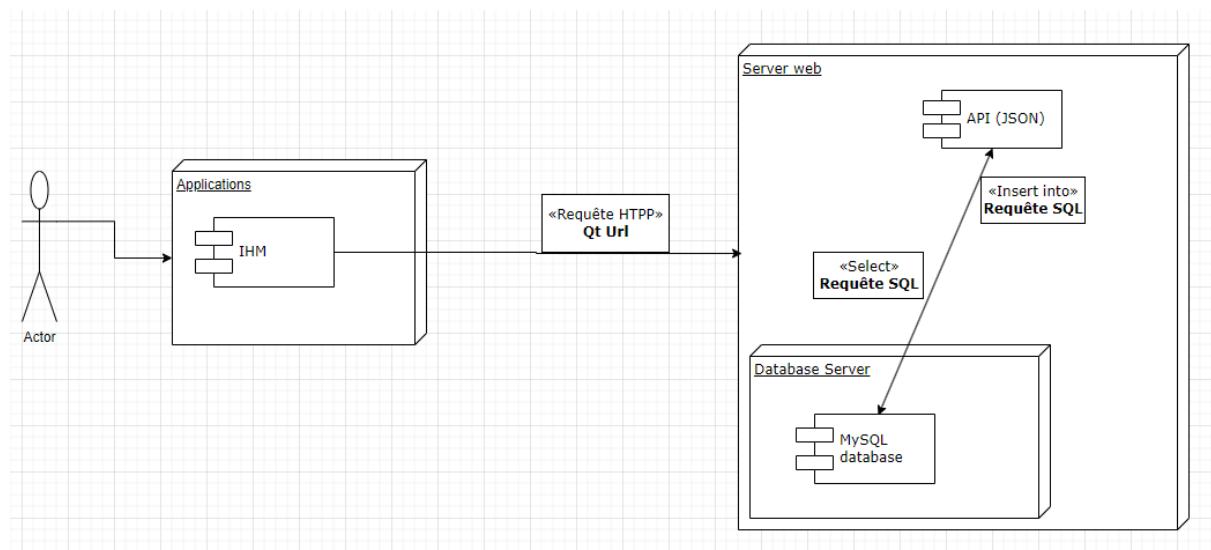
eleve3:



Diagrammes de déploiement



eleve3:

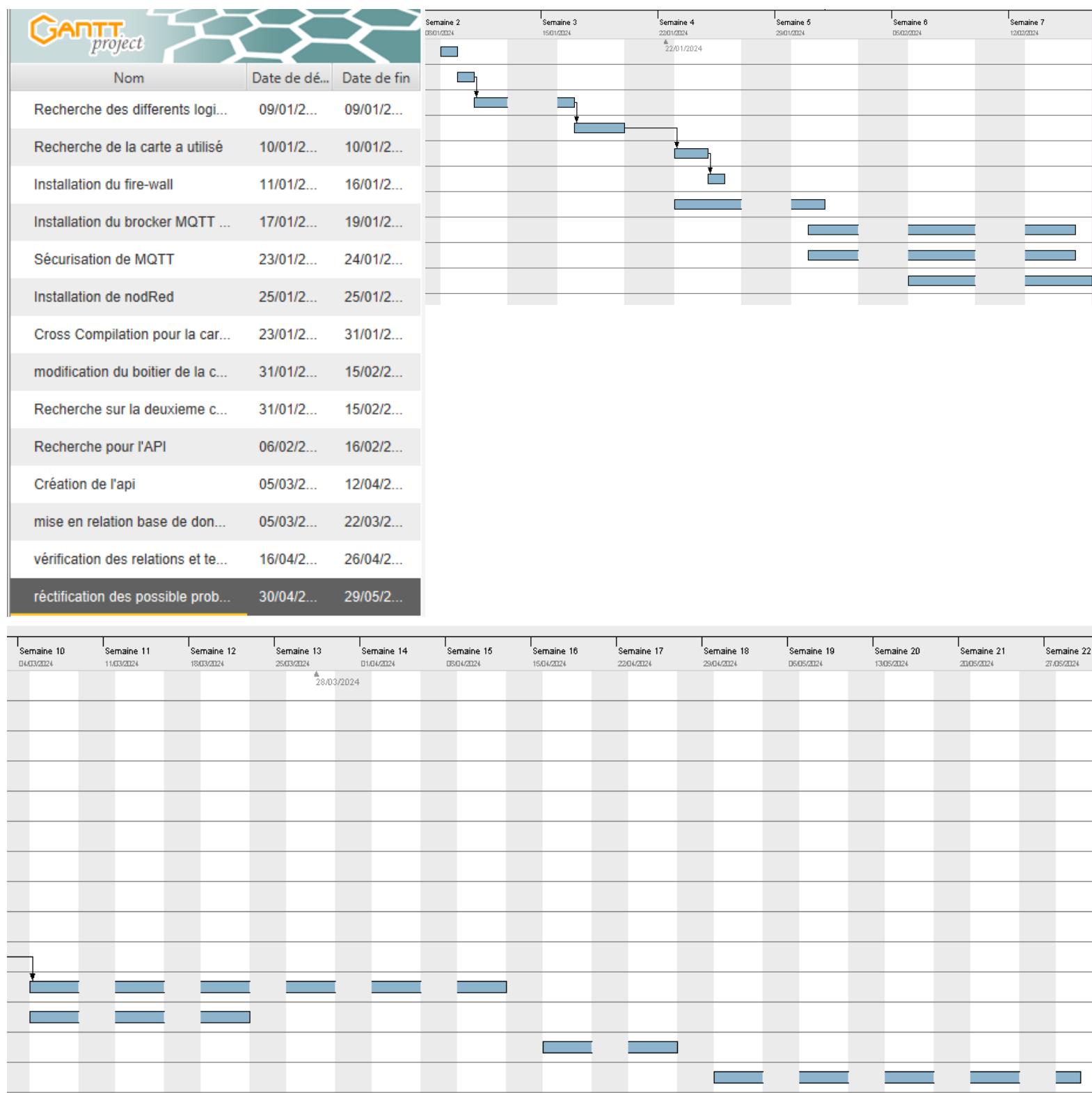


Organisation du groupe

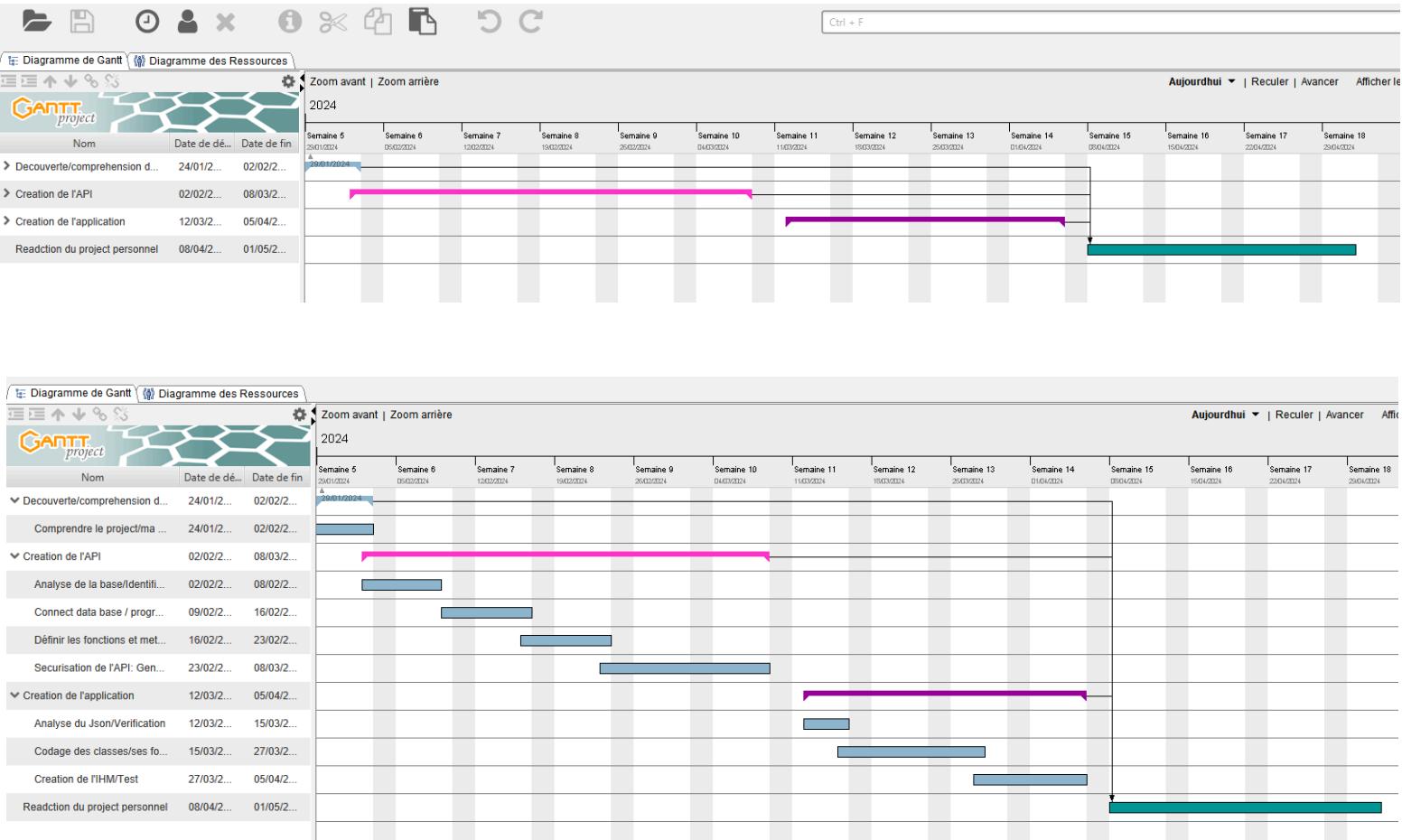
Planning prévisionnel

	S5/S6 29/01-09/02	S7/S10 12/02-08/03	S11/S12 11/03-22/03	
Amine	Rédaction et répartition des tache de la partie commune	Création de l'API: connection à la base/les classe et ses fonctions	définir les méthodes des fonctions securisation:generer des clés automatique	
Christelle	Rédaction et répartition des tache de la partie commune	Création du site web et de la base de données		
Clement	Rédaction et répartition des tache de la partie commune	Mise en place du parfeux serveur MQTT, nodred, 2ème carte réseaux,	Mise en relation de la base de donnée et les donnée des capteurs	
	S13/S14 25/03-05/04	S15/S18 08/04-29/04	S19/S20 06/05-17/05	S21/22 20/04-31/04
Amine	classe/ses fonctions/ ses definition	interface/afficher les données	Rédaction du rapport de la partie personnelle	
Christelle	Importation du site web et de la base de donnée dans l'interface	Tests et vérifications finales	Rédaction de la partie personnelle	
Clement	Création de l'application de gestion sur écran tactile	Rédaction du rapport de la partie personnelle		

Planning prévisionnel de Clément

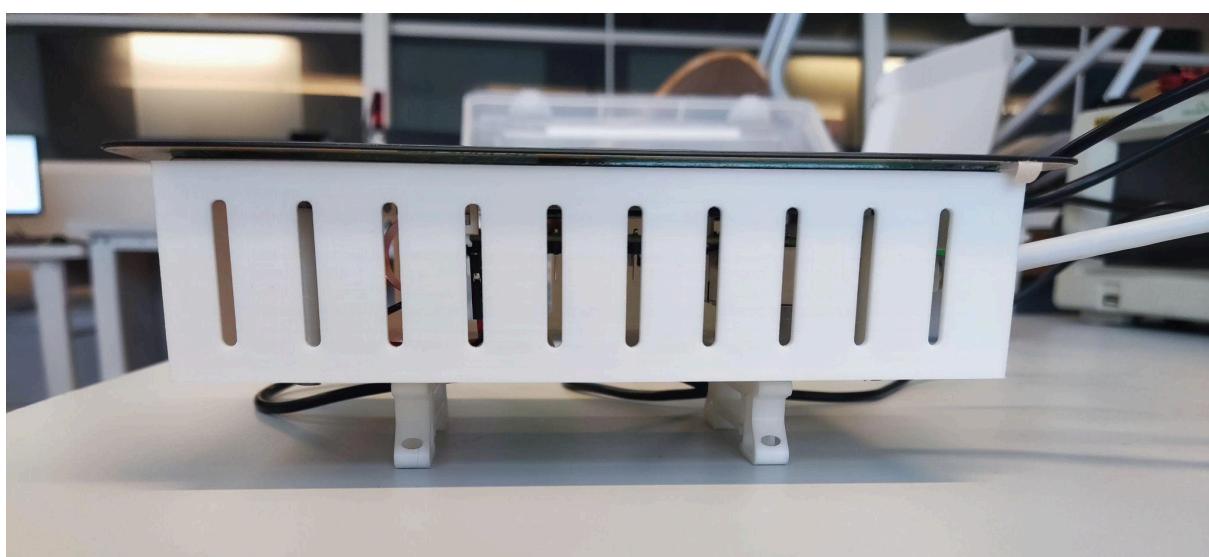
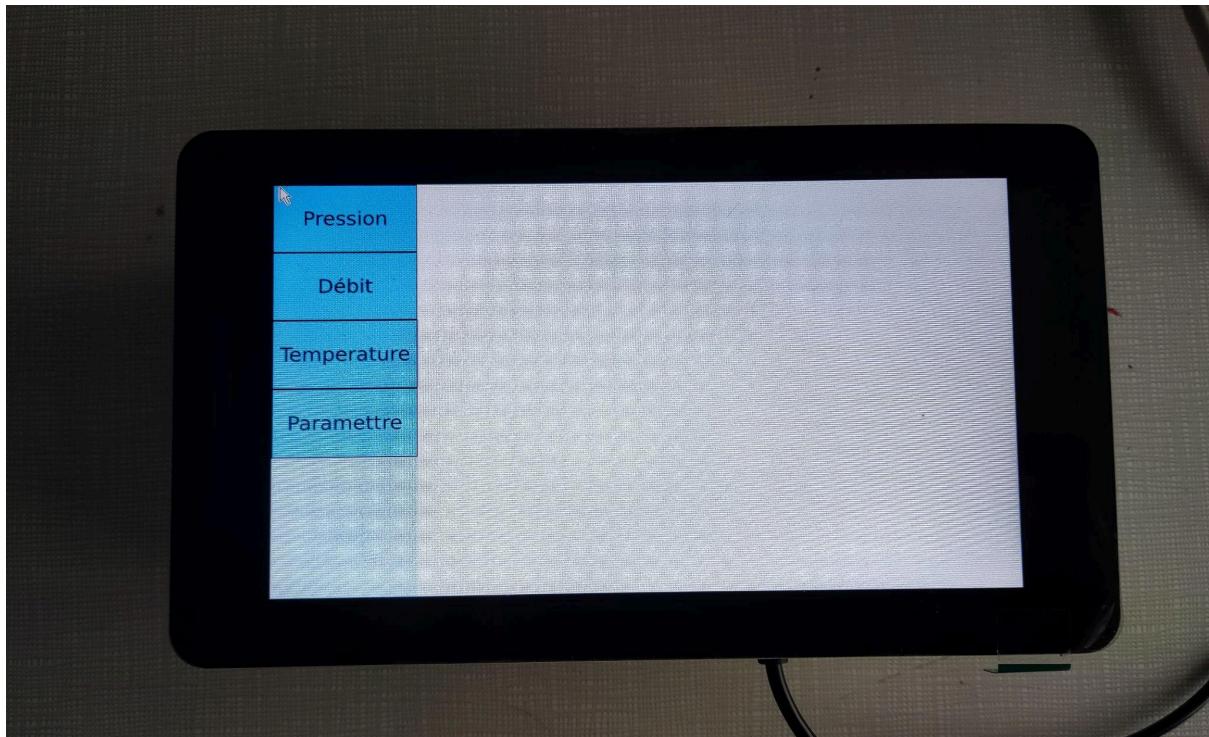


Planning prévisionnel de Amine



Rapport de projet

Passerelle industriel 4.0



Sommaire

Sommaire.....	21
Cahier des charges.....	22
Présentation des différents services et matériaux utilisés.....	23
UML.....	24
Solutions choisies.....	25
Prévision des tâches.....	26
Tâches finalement réalisées.....	26
Tâches adjacentes au projet.....	27
Sécurité et pare-feu.....	28
Broker MQTT.....	29
Nodered.....	31
Interface homme machine.....	34
Exemples.....	38
Annexe.....	39

Tout d'abord je tiens à remercier mes professeurs de spécialité, Monsieur Bulcke, Monsieur Barthomeuf et Monsieur Degoute ainsi que Madame Chopin professeure de sciences physiques pour leur aide et leur soutien apportés tout au long de l'année.

Cahier des charges

Pour ce projet, on m'a imposé d'avoir un pare-Feux fonctionnel afin de sécuriser les capteurs ainsi que les données qui sont stockées dans la carte. Je devais utiliser un broker MQTT et Nodered.

J'avais pour objectif de créer une interface homme machine sur le microcontrôleur permettant de configurer la passerelle et d'accéder aux mesures des capteurs.

On m'a aussi fourni un capteur de débit d'air qui mesure la pression, la température, le débit d'air et la quantité d'air éoulée ainsi qu'un maître IFM utilisant la technologie IO-Link.

Il faut également que le projet soit connecté à deux réseaux: une connexion au réseau du côté industriel c'est-à-dire, du côté des capteurs et une connexion au réseau du côté externe donc du côté lycée et du côté internet.

Présentation des différents services et matériaux utilisés

UFW est un outil de configuration de pare-feu simple utilisant des lignes de commandes.

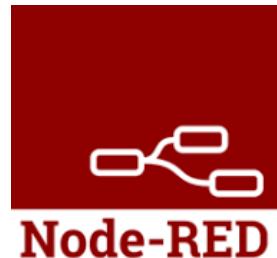


Ubuntu UFW

MQTT est un protocole de messagerie pour l'internet des objets (IOT). MQTT est une messagerie de type publications/abonnements



Node-RED est un outil de programmation graphique qui permet de relier plusieurs matériels. Il fournit un éditeur réalisant des graphiques simples, basé sur un navigateur qui facilite la programmation.



Le langage QML est un langage de programmation à balise, principalement utilisé pour les interfaces utilisateur.



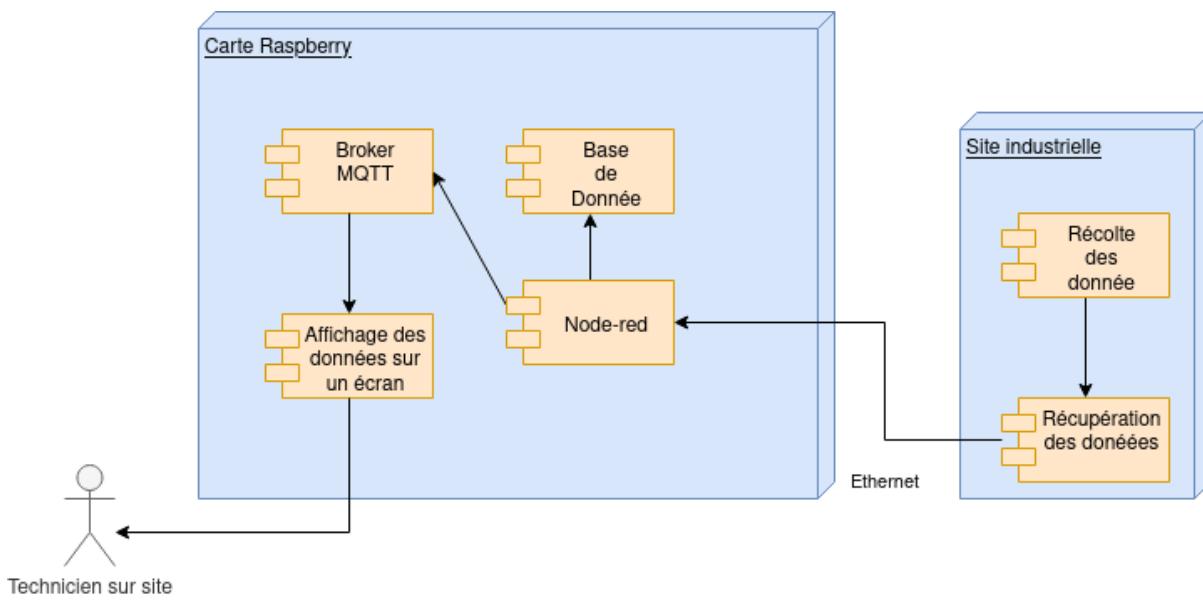
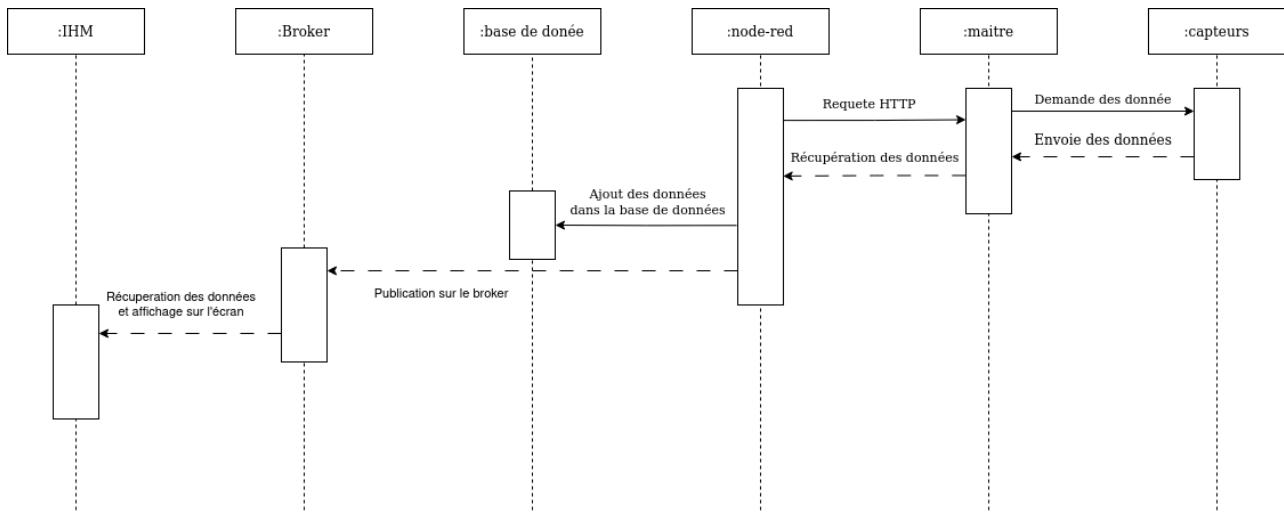
Voici la boite que l'on m'as fournie avec un maître IFM.



On m'as également fourni un capteur débile mètre.



UML



Solutions choisies

J'ai choisi la carte Raspberry pi 4 comme microcontrôleur et Raspbian qui est une distribution debian adaptée aux cartes raspberry comme système d'exploitation. J'ai choisi d'utiliser cette carte et ce système d'exploitation car nous les avons déjà utilisés lors de nos travaux pratiques en deuxième année de BTS.

En ce qui concerne le broker MQTT, j'ai choisi Mosquitto car c'est un logiciel open source, gratuit et beaucoup utilisé, donc bien documenté avec beaucoup de tutoriels d'installation disponibles sur internet.

Comme pare-feu j'ai choisi Uncomplicated FireWall (UFW) car il s'agit d'un pare-feu gratuit, simple à comprendre et simple d'utilisation.

J'ai choisi comme langage de programmation pour l'interface homme machine d'utiliser QML qui est un langage d'interface utilisateur et facilement portable.



Carte raspberry pi 4

Prévision des tâches

Tout d'abord il faut mettre en place le microcontrôleur en créant mon espace de travail via les commandes qui crée un utilisateur, puis en ajoutant cet utilisateur au groupe des super utilisateurs via les commandes suivantes : `sudo adduser clement` et `sudo adduser clement sudo`. Ensuite il faut réaliser l'installation du pare-feux pour sécuriser la carte Raspberry, pour ensuite mettre en place le broker MQTT, puis le serveur node-red. Il faut aussi réaliser le lien entre la base de données et les données des capteurs via le broker MQTT.

Une fois ces étapes réalisées, on peut enfin réaliser l'interface homme machine.

Ci-contre, voici la liste des tâches à réaliser dans l'ordre chronologique de haut en bas :

Nom	Date de dé...	Date de fin
Recherche des différents logi...	09/01/2...	09/01/2...
Recherche de la carte a utilisé	10/01/2...	10/01/2...
Installation du fire-wall	11/01/2...	16/01/2...
Installation du brocker MQTT ...	17/01/2...	19/01/2...
Sécurisation de MQTT	23/01/2...	24/01/2...
Installation de nodRed	25/01/2...	25/01/2...
Cross Compilation pour la car...	23/01/2...	31/01/2...
modification du boitier de la c...	31/01/2...	15/02/2...
Recherche sur la deuxieme c...	31/01/2...	15/02/2...
Recherche pour l'API	06/02/2...	16/02/2...
Création de l'api	05/03/2...	12/04/2...
mise en relation base de don...	05/03/2...	22/03/2...
vérification des relations et te...	16/04/2...	26/04/2...
récitification des possible prob...	30/04/2...	29/05/2...

Tâches finalement réalisées

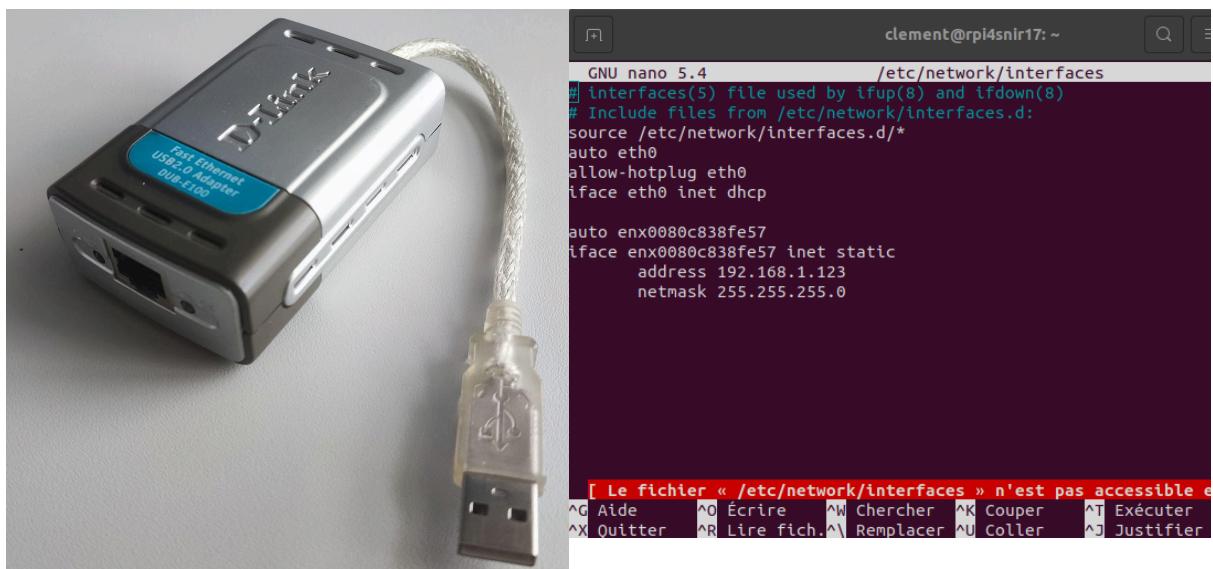
La plupart des tâches ont été réalisées dans le bon ordre sauf pour réaliser le lien entre les capteurs et la base de données. J'ai finalement utilisé des requêtes HTTP via Nodered pour ajouter les données récupérées dans la base de données.

Tâches adjacentes au projet

On m'a fourni un écran à connecter à la carte raspberry pi 4 via une fiche afin de réaliser l'interface homme machine.

J'ai également réalisé un boîtier pour y accueillir le microcontrôleur et l'écran, auquel j'ai ajouté des rails din pour pouvoir mettre le projet dans une armoire électrique sur site.

Pour ajouter la deuxième carte réseau, j'ai utilisé un connecteur USB - Ethernet, puis j'ai rajouté en statique l'adresse ip via la commande `nano /etc/network/interfaces`, cette adresse sera reliée à un seul équipement, le maître IFM.



J'ai fait en sorte que les programmes se lancent automatiquement au démarrage de la carte, pour cela j'ai utilisé la commande `sudo crontab -e`.

Il existe une version de cette commande sans le sudo mais dans mon cas lorsque que mon programme s'exécutait, mon IHM avait des problèmes de droit qui étaient nécessaires à son bon fonctionnement. La crontable est principalement utilisée pour lancer des script à un intervalle

régulier, mais il y a un argument qui permet de lancer des scripts au démarrage qui s'intitule `@reboot`.

```
GNU nano 5.4                               /tmp/crontab.VP7cAR/crontab
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
@reboot sleep 60 && /home/clement/apiPassrelle/ip/get_ip_address.sh
@reboot sleep 10 && sudo /home/clement/apiPassrelle/bin/apiPassrelle
*/10 * * * * /home/clement/suppression_base_de_donne_30_jour/suppression.sh
```

La première ligne non commentée a pour rôle de mettre dans des fichiers texte l'adresse IP ainsi que le masque. Dans ce contexte, on utilise l'argument `sleep` pour attendre une minute, le temps que la carte se lance correctement et que le réseau soit opérationnel.

La deuxième ligne lance au démarrage l'interface homme machine seulement au bout de dix secondes car on n'a pas besoin d'avoir le réseau d'opérationnel pour lancer l'interface homme machine.

Enfin la troisième ligne consiste à supprimer automatiquement les données de la base de données qui datent de plus de 30 jours, cette fois-ci il s'agit d'un lancement régulier qui a lieu toutes les 10 minutes.

Sécurité et pare-feu

J'ai choisi Uncomplicated FireWall car c'est un pare-feu simple d'utilisation et bien documenté. J'ai choisi comme politique de sécurité de tout restreindre et d'autoriser seulement certaines adresses IP. Je commence donc à installer le pare-feu via le terminal connecté en SSH à la carte Raspberry, et je désactive toutes les connexion entrante : `sudo ufw disable`, puis j'autorise les connexions venant du réseaux du lycée

pour qu'on puisse, les élèves du groupe et moi-même nous connecter sur la carte `sudo ufw allow from 172.21.28.0/128`.

Broker MQTT

Je commence par installer Mosquitto via les paquets avec la commande : `sudo apt-get install mosquitto`, puis je vérifie que l'installation s'est bien effectuée via la commande: `systemctl status mosquitto`. Par la suite, il faut ajouter Mosquitto au service pour que Mosquitto se lance automatiquement au lancement de la carte via la commande suivante: `sudo systemctl enable mosquitto.service`.

Ensuite, il faut sécuriser le serveur mqtt avec un mot de passe via la commande suivante : `sudo mosquitto_passwd -c /etc/mosquitto/passwd` (identifiant) puis il faut taper le mot de passe. Enfin il faut modifier un fichier pour renseigner le port sur lequel mosquitto fonctionnera, il faudra donc ajouter les lignes suivante `per_listener_settings true` puis préciser sur quel port mosquitto va fonctionner `listener 1883` et `protocol mqtt`, ensuite on précise dans quel fichier il faut vérifier le mot de passe : `password_file /etc/mosquitto/passwd`. Finalement, on redémarre Mosquitto pour que les modifications soient prises en compte. Mais il s'avère que lorsque je regarde la trame quand je me connecte à MQTT Explorer avec wireshark, l'identifiant et le mot de passe ne sont pas cryptés, il faut donc éviter cela pour avoir une meilleure sécurité.

```

0000 e4 5f 01 6d f9 e5 48 9e bd a1 e5 f3 08 00 45 00 m H E
0010 00 64 bc 2d 40 00 80 06 00 00 ac 15 1c 4a ac 15 d -@ J
0020 1c 43 e9 a3 07 5b c8 1d 62 33 1a aa 7c 53 50 18 C [ b3 | SP
0030 20 03 91 0e 00 00 10 3a 00 04 4d 51 54 54 04 c2 : MQTT
0040 00 3c 00 16 6d 71 74 74 2d 65 78 70 6c 6f 72 65 < mqtt-explore
0050 72 2d 62 35 38 64 63 63 62 32 00 0a 70 61 73 73 r-b58dcc b2 pass
0060 65 72 65 6c 6c 65 00 0a 70 61 73 73 65 72 65 6c erelle passerel
0070 6c 65 le

```

Time	Source IP	Destination IP	Protocol	Description
100 22.077638	172.21.28.74	172.21.28.67	MQTT	114 Connect Command
101 22.077939	172.21.28.67	172.21.28.74	TCP	60 1883 → 59811 [ACK] Seq=1 Ack=61 Win=64256 Len=0
102 22.079618	172.21.28.67	172.21.28.74	MQTT	60 Connect Ack
103 22.080423	172.21.28.74	172.21.28.67	MQTT	75 Subscribe Request (id=29169) [#], Subscribe Request (id=29170) [\$SYS/#]
104 22.080810	172.21.28.67	172.21.28.74	MQTT	60 Subscribe Ack (id=29169)
105 22.134508	172.21.28.74	172.21.28.67	TCP	54 59811 → 1883 [ACK] Seq=82 Ack=10 Win=2097920 Len=0
106 22.136011	172.21.28.67	172.21.28.74	MQTT	1333 Subscribe Ack (id=29170), Publish Message [\$SYS/broker/version], Publish Message [\$S

Afin de pallier ce problème, j'ai utilisé le protocole websocket. J'ai dû à nouveau aller dans le fichier de configuration de Mosquitto et ajouter au

fichier les lignes suivantes : **listener 9001** et **protocol websockets**. On se rend alors compte qu'en utilisant à nouveau wireshark, on observe que la tram Connected Command n'est plus présente, la connexion est donc maintenant cryptée.

46 5.473096	172.21.28.74	172.21.28.67	WebSoc...	61 WebSocket Binary [FIN] [MASKED]
47 5.473392	172.21.28.67	172.21.28.74	TCP	60 9001 → 62057 [ACK] Seq=160 Ack=266 Win=64128 Len=0
48 5.473589	172.21.28.74	172.21.28.67	WebSoc...	119 WebSocket Binary [FIN] [MASKED]
49 5.473939	172.21.28.67	172.21.28.74	TCP	60 9001 → 62057 [ACK] Seq=160 Ack=331 Win=64128 Len=0
50 5.475271	172.21.28.67	172.21.28.74	WebSoc...	60 WebSocket Binary [FIN]
51 5.478198	172.21.28.74	172.21.28.67	WebSoc...	61 WebSocket Binary [FIN] [MASKED]
52 5.478586	172.21.28.74	172.21.28.67	WebSoc...	80 WebSocket Binary [FIN] [MASKED]
53 5.478907	172.21.28.67	172.21.28.74	TCP	60 9001 → 62057 [ACK] Seq=166 Ack=364 Win=64128 Len=0
54 5.479281	172.21.28.67	172.21.28.74	WebSoc...	61 WebSocket Binary [FIN]
55 5.479281	172.21.28.67	172.21.28.74	WebSoc...	61 WebSocket Binary [FIN]
56 5.479368	172.21.28.74	172.21.28.67	TCP	54 62057 → 9001 [ACK] Seq=364 Ack=180 Win=262400 Len=0
57 5.479411	172.21.28.67	172.21.28.74	WebSoc...	103 WebSocket Binary [FIN]
58 5.479411	172.21.28.67	172.21.28.74	WebSoc...	89 WebSocket Binary [FIN]
59 5.479482	172.21.28.74	172.21.28.67	TCP	54 62057 → 9001 [ACK] Seq=364 Ack=264 Win=262400 Len=0
60 5.479526	172.21.28.67	172.21.28.74	WebSoc...	103 WebSocket Binary [FIN]
61 5.479526	172.21.28.67	172.21.28.74	WebSoc...	103 WebSocket Binary [FIN]
62 5.479571	172.21.28.74	172.21.28.67	TCP	54 62057 → 9001 [ACK] Seq=364 Ack=362 Win=262400 Len=0

Je vais utiliser le broker MQTT pour publier les dernières données, afin de les afficher sur mon interface homme machine.

Nodered

Nodered est un outil de programmation graphique utilisant des ‘nodes’. Nodered est principalement utilisé pour programmer des serveurs, dans notre cas nous allons utiliser Nodered pour réaliser des graphiques simples et ajouter les données récupérées dans la base de données. La mise en place de Nodered est rapide et sans grande difficulté; il suffit de faire la commande suivante puis de renseigner l’identifiant, le mot de passe et une passe phrase : `bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)`.

Nodered utilise le port 1880. Pour aller sur la page de connexion de Nodered il suffit de taper l’adresse ip du microcontrôleur et d’ajouter à la suite “:1880”

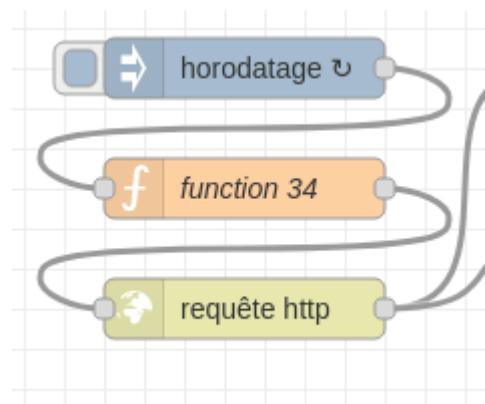


Voici la page de connexion, où il faudra renseigner le mot de passe et l’identifiant renseigné plus haut.

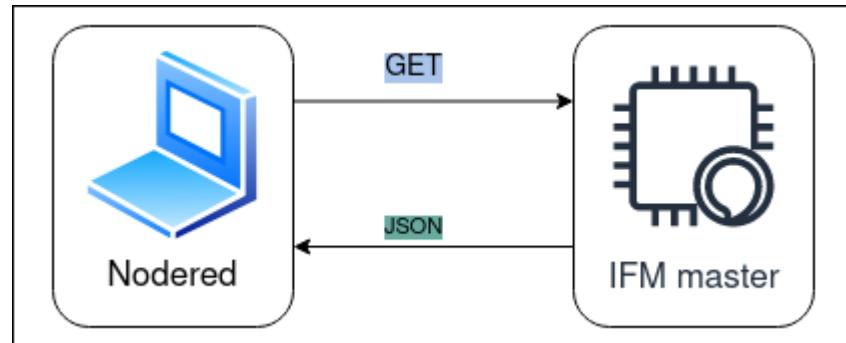
J’ai réalisé un programme qui ajoute les données dans la base de données puis les publie sur le broker et affiche les données sur la partie graphique de node red.

La partie ci-jointe réalise le cadencement des données à récupérer tous les x temps.

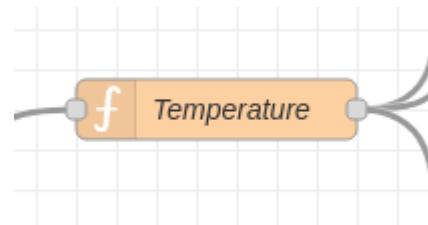
La fonction 34 est en réalité un lien internet, qui a pour rôle de demander les données au maître IFM qui envoie les données au format JSON.



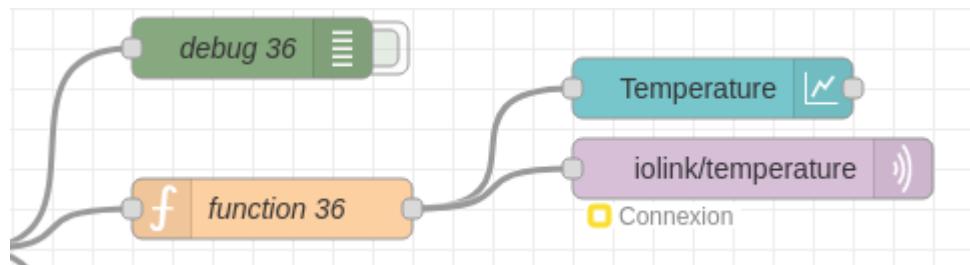
Le bloc “requête http” fonctionne de la manière suivante : Nodered envoie une requête GET à une adresse spécifique, puis le master ifm envoie un JSON en réponse.



Dans le bloc “Temperature“ on isole la donnée de température du JSON de réponse dans lequel il y a énormément d'information.

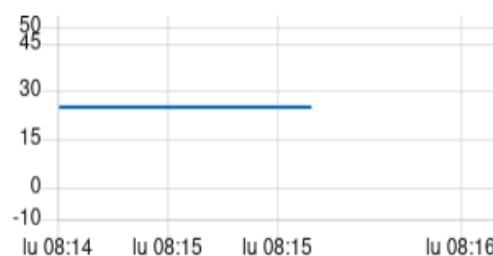


Ensuite, dans la fonction 36 on utilise la dernière donnée de température pour la publier sur le broker et pour l'afficher sur le tableau de bord de nodered.



iolink

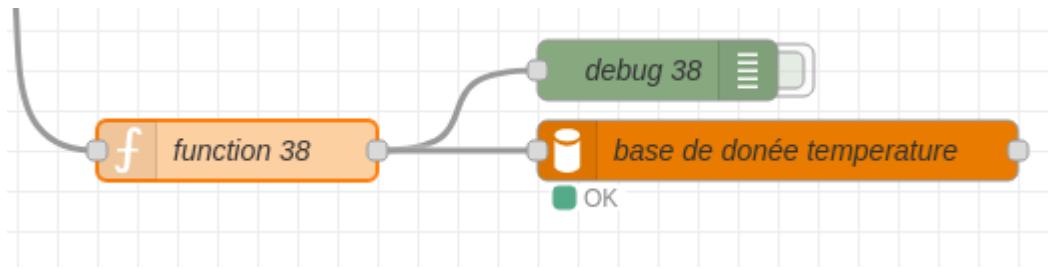
Temperature



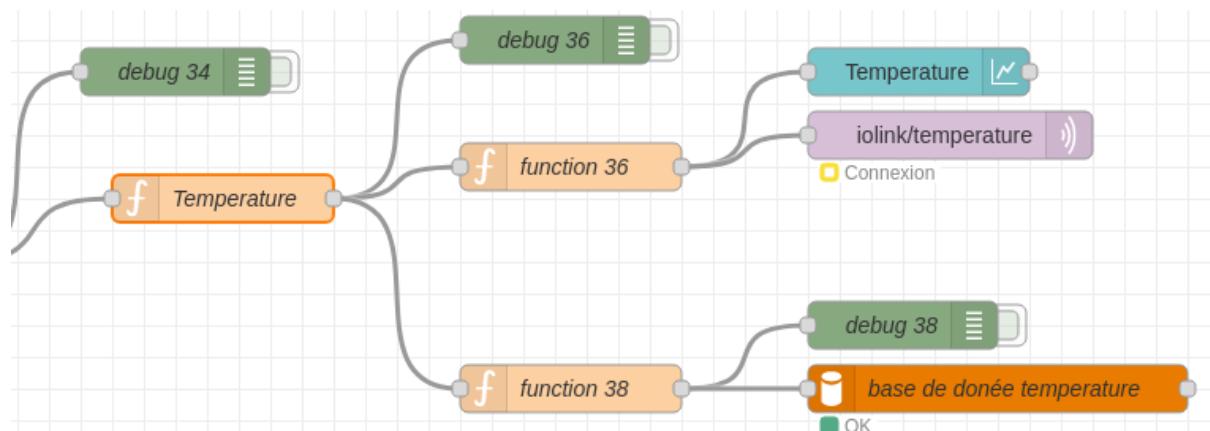
Enfin, dans la fonction 38 on met en forme une insertion SQL pour ajouter cette nouvelle donnée à la base de données.

```
Code présent dans la fonction 38: msg.topic = "INSERT INTO
temperature (temperature,unite,date_temp,time_temp)
VALUES(" + msg.payload.temperature +
",,"°C,CURRENT_DATE,CURRENT_TIME);";
return msg;
```

Dans les données envoyées pour la base de donnée la première colonne est la température, la seconde est l'unité, la troisième est la date au format année-mois-jour enfin la quatrième est l'heure au format heure:minute:seconde



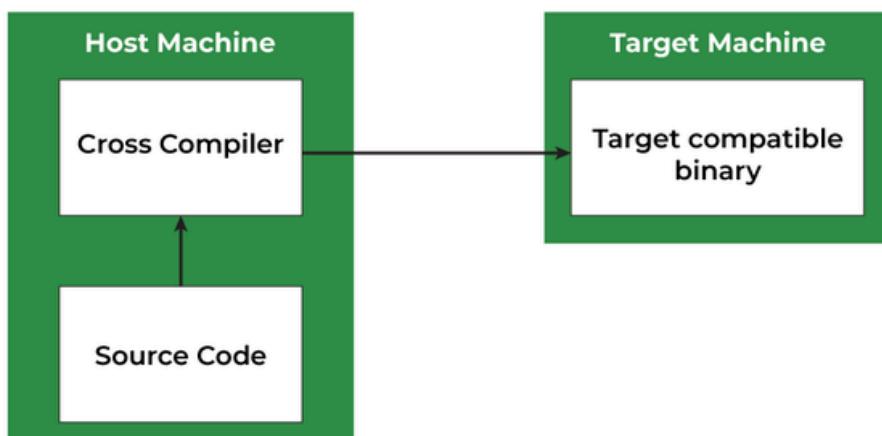
Ce programme est répété deux fois pour les autres données à un cadencement différent.



Interface homme machine

Pour pouvoir réaliser l'interface homme machine, il faut d'abord réaliser la compilation croisée ou “cross compilation”, il s'agit de programmer un exécutable sur une autre cible que celle sur laquelle le compilateur s'est exécuté. Le code source est donc compilé sur la machine “hôte” avec un ensemble d'outils de développement spécifique, mais le résultat est conçu pour être exécuté sur une autre machine, la machine “cible”, cette machine peut avoir un tout autre système d'exploitation.

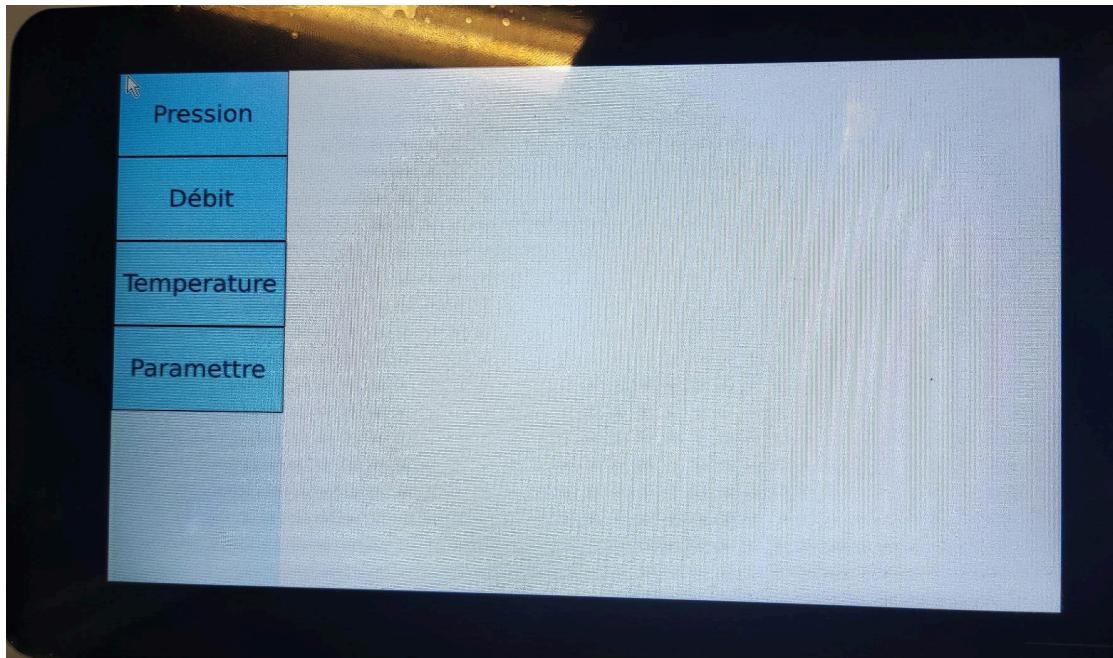
Cross Compiler Operation



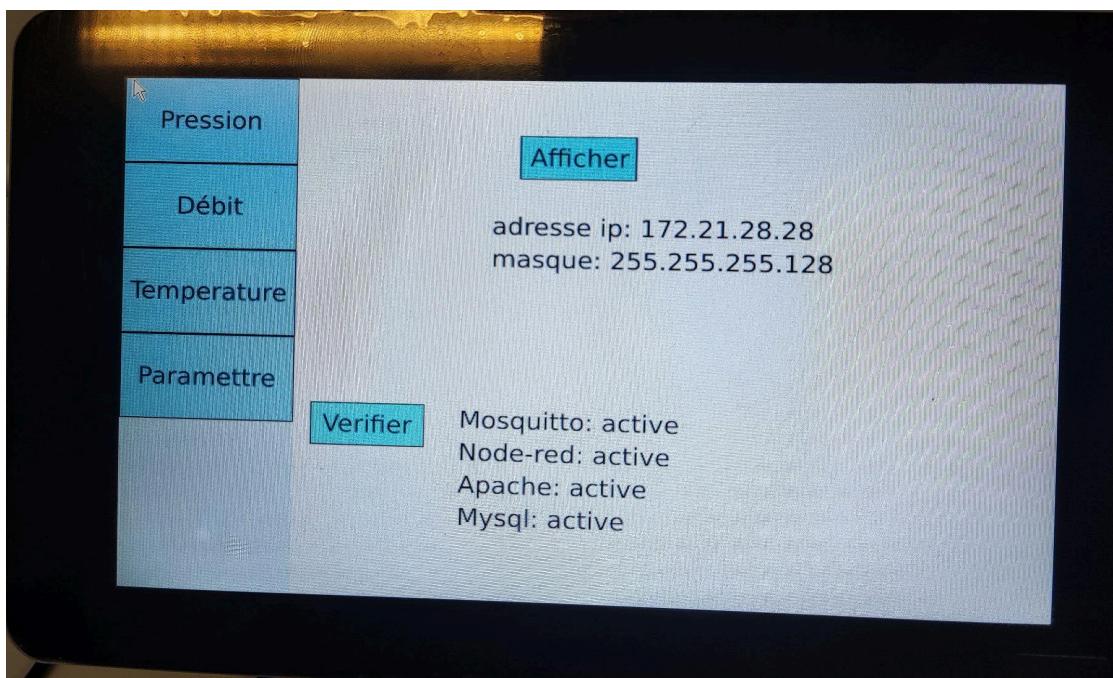
La compilation croisée est un outil qui permet aux développeurs de créer des programmes pour des systèmes différents de ceux sur lesquels ils travaillent. Il s'agit d'un outil énormément utilisé pour les systèmes embarqués.

Une fois la compilation croisée finie, je cherche une manière de réaliser plusieurs pages sur l'écran, et de changer de page lorsque l'on appuie sur un bouton, mais il s'avère que pour créer plusieurs pages il aurait fallu que je choisisse un autre compilateur croisé. J'ai donc trouvé une alternative, en activant et désactivant les différents boutons et graphiques.

Appui sur le bouton “Débit”:



Appui sur le bouton “Paramètre”:



Sur l'image d'au-dessus on peut voir les paramètres réseaux de la carte. Les paramètres sont récupérés via un script shell puis en affichant un document texte qui contient l'adresse ip et le masque réseau. Le script ci-dessous récupère les paramètres ip et les met dans les fichiers respectif ipaddress et masqueaddress.

```

clement@rpi4snir17: ~/apipassrelle/ip
GNU nano 5.4
get_ip_address.sh
ifconfig | grep inet | head -n1 | tail -n2 | cut -c 14-26 > /home/clement/apipassrelle/ip/ipadresse
chmod 777 ipadresse
ifconfig | grep inet | head -n1 | tail -n2 | cut -c 36-51 > /home/clement/apipassrelle/ip/masqueadresse
chmod 777 masqueadresse

```

^G Aide ^O Écrire ^W Chercher ^K Couper ^T Exécuter ^C Emplacement M-U Annuler
^X Quitter ^R Lire fich. ^V Remplacer ^U Coller ^J Justifier ^A Aller ligne M-E Refaire

J'ai aussi réalisé un script shell pour savoir si les services de la carte sont opérationnels.

```

clement@rpi4snir17: ~/apipassrelle/service
GNU nano 5.4
service.sh
tes=$(systemctl status mosquitto | grep Active | cut -c 21)
if [ "$tes" =~ ")" ]
then
    systemctl status mosquitto | grep Active | cut -c 13-20 > service/mosquitto
    chmod 755 service/mosquitto
else
    systemctl status mosquitto | grep Active | cut -c 13-21 > service/mosquitto
    chmod 755 service/mosquitto
fi

```

Ce script récupère l'état du service et le met dans un fichier au même nom que le service, dans ce cas la mosquitto. Il y a deux possibilités: "active" ou "inactive". Ce script est répété au total 4 fois pour les services suivants: Mosquitto, Node-red, Apache et Mysql.

Pour ensuite afficher le contenu des fichiers texte il faut ajouter une classe au qml qui va pouvoir lire un fichier; le code de la classe est disponible dans l'annexe. Sur le fichier QML il faut donc déclarer un "Text", l'objet "FileIO" dans celui-ci on précise donc le chemin qu'il faut emprunter pour accéder au fichier. Une fois l'objet fini, sur l'écran il y aura donc un message texte précisant dans ce cas : "Mosquitto: puis le statut du service".

```

Rectangle {
    id:mosquitto_service
    width: 150
    height: 100
    visible: false
    x: 300
    y: 300

    Text {
        id: textmosquitto
    }

    FileIO {
        id: mosquitto
        source: "/home/clement/apipassrelle/service/mosquitto"
        onError: console.log(msg)
    }

    Component.onCompleted: {
        console.log(mosquitto.read());
        textmosquitto.text = "Mosquitto:" + mosquitto.read();
    }
}

```

Sur la page des paramètres il y a aussi deux boutons qui exécute les scripts décrit plus haut. Il faut également ajouter une classe pour pouvoir exécuter des scripts lors d'appuis sur un bouton.

```
Button
{
    id: verifier
    visible: false
    text:"Verifier"
    Rectangle {
        implicitWidth: 100
        implicitHeight: 40
        color: "#57e5e0"
        border.color: "#000000"
        border.width: 1
        radius: 1
    }
    onClicked:
    {
        function go()
        {
            service_mosquitto.start("/bin/bash", [/home/clement/apiPassrelle/service/service.sh"])
            textmosquitto.text = "Mosquitto:" + mosquitto.read();
            textnode_red.text = "Node-red: " + node_red.read();
            textapache.text = "Apache: " + node_red.read();
            textmysql.text = "Mysql:" + mysql.read();
        }
        go()
    }
    x: 170
    y: 300
}
```

Malheureusement, au jour de la rédaction du rapport de projet je n'ai pas encore réussi à afficher les données sur l'écran car une erreur s'affiche sur la construction de la classe qui permet de se connecter au broker pour récupérer les données.

Exemples

Exemple de chemin qu'une donnée peut emprunter.

Premièrement Nodered envoie une requête http au maître ifm.

```
1 msg.url = "http://172.21.28.3";
2 msg.payload = {
3     "code": "request",
4     "cid": 1,
5     "adr": "/iolinkmaster/port[1]/iolinkdevice/pdin/getdata"
6 }
7 return msg;
8
```

Le maître ifm demande au capteur et le capteur répond avec les données récupérées puis l'envoie au maître ifm, Nodered récupère donc un json de toutes les données récupérées en hexadécimal.

```
cid: -1
▼ data:
  value: "3DB1F8A10000FD000A14FE0000000300"
  code: 200
```

Ensuite dans node red on traite la donnée et on la rend lisible et accessible à tout le monde puis on l'ajoute à la base de données.



En même temps, la donnée est publiée sur le broker mqtt



Puis la donnée est affichée sur l'interface homme machine.

Conclusion

Pour conclure ce projet m'a apporté une nouvelle approche des travaux de groupes. Ce projet m'as aussi permis d'approfondir mes connaissances dans les systèmes UNIX et dans le QML, également dans la construction UML.

Annexe

Classe fileio:

```
1  #ifndef FILEIO_H
2  #define FILEIO_H
3
4  #include <QObject>
5
6  class FileIO : public QObject
7  {
8  |   Q_OBJECT
9
10 public:
11 |   Q_PROPERTY(QString source
12 |               READ source
13 |               WRITE setSource
14 |               NOTIFY sourceChanged)
15 |   explicit FileIO(QObject *parent = 0);
16
17     Q_INVOKABLE QString read();
18
19     QString source() { return mSource; }
20
21 public slots:
22     void setSource(const QString& source) { mSource = source; }
23
24 signals:
25     void sourceChanged(const QString& source);
26     void error(const QString& msg);
27
28 private:
29     QString mSource;
30 };
31
32 #endif // FILEIO_H


---


1  #include "fileio.h"
2  #include <QFile>
3  #include <QTextStream>
4
5  // Constructeur de la classe
6  FileIO::FileIO(QObject *parent) :
7  |   QObject(parent)
8  {
9
10 }
11 // Methode qui permet de lire les données d'un fichier
12 QString FileIO::read()
13 {
14     if (mSource.isEmpty()){
15         emit error("source is empty");
16         return QString();
17     } // Vérification si la source n'est pas vide
18
19     QFile file(mSource);
20     QString fileContent; // initialisation de la variable qui va récupérer le contenu du fichier
21     if (file.open(QIODevice::ReadOnly)) { // ouverture du fichier
22         QTextStream t(&file);
23         do { // ajoute ligne par ligne dans la variable fileContent des données du fichier
24             QString line;
25             line = t.readLine();
26             fileContent += line;
27         } while (!line.isNull());
28
29         file.close(); // fermeture du fichier
30     } else {
31         emit error("Unable to open the file");
32         return QString();
33     }
34     return fileContent; // envoie du contenu du fichier
35 }
```

Classe process:

```
1 #include <QProcess>
2 #include <QVariant>
3
4 ▼ class Process : public QProcess {
5     Q_OBJECT
6
7 public:
8     ▼ Process(QObject *parent = 0) : QProcess(parent) {
9         QProcess::setWorkingDirectory("/home/clement/apipassrelle");
10    }
11
12    ▼ Q_INVOKABLE void start(const QString &program, const QVariantList &arguments) {
13        QStringList args;
14
15        // Convertit un QVariantList provenant de QML en QStringList pour le QProcess
16
17        for (int i = 0; i < arguments.length(); i++)
18            args << arguments[i].toString();
19
20        QProcess::start(program,args);
21    }
22
23    ▼ Q_INVOKABLE QByteArray readAll() {
24        return QProcess::readAll();
25    }
26 };
27
```

Notice d'utilisation

Il faut d'abord brancher le câble d'alimentation, puis la prise ethernet et enfin le module USB-Ethernet une fois que l'IHM s'affiche, on peut vérifier l'adresse ip du côté externe via le bouton paramètre.

Pour l'adresse ip du côté capteur il s'agit d'une adresse statique : 192.168.1.123/24.

Il faut maintenant se connecter à Nodered en mettant comme lien l'adresse ip et rajouter “:1880”.

Ensuite, il faut se connecter avec les identifiants “clement” et mot de passe “!lafayette-63”

Une fois sur nodered il y aura une page IOlink, dans laquelle il va falloir modifier les blocs pour le broker mqtt (blocs violet) et modifier l'adresse ip du broker utilisé par l'adresse de la carte.

Il faut prendre exemple sur les blocs déjà mis en place pour adapter le système aux capteurs présents dans le nouvel environnement.