

# Partie Personnelle

## Elyes Ghouaiel 2TS-SNIR

## Lycée des Eucalyptus

### I. Étude Informatique

#### 1. Introduction

Permettez-moi de vous présenter ma contribution personnelle en vous expliquant les tâches qui m'ont été assignées. Dans notre système de supervision, nous avons divisé les responsabilités en trois parties distinctes, et la mienne est de gérer tous les afficheurs du parking. En utilisant une base de données centralisée, telle que celle de M. Megret, je récupère les données nécessaires et je suis chargé de les afficher sur l'afficheur principal. Voici les missions qui m'ont été confié :

Étudiants	Missions	Matériels ou logiciels SPÉCIFIQUES
Étudiant 2 : Informatique et réseau	<b>Vous êtes chargé de :</b> <ul style="list-style-type: none"><li>- De gérer l'ensemble des afficheurs du parking<ul style="list-style-type: none"><li>o Possibilité de choisir un afficheur dans le réseau et de diffuser des messages d'information ou publicitaire.</li><li>o Possibilité de consulter un historique des messages.</li></ul></li></ul>	<ul style="list-style-type: none"><li>- Raspberry Pi</li><li>- Docker MQTT</li></ul>
Étudiant 1 : Physique appliquée	<b>Vous êtes chargé de :</b> <ul style="list-style-type: none"><li>- D'étudier la consommation électrique du système en plein air</li><li>- De faire des expériences et mesures</li></ul>	<ul style="list-style-type: none"><li>- Ensemble des afficheurs</li><li>- Plot LORA Bosch TPS110EU</li><li>- Matériel de mesures</li></ul>

Afin de réussir les tâches spécifiées dans mon cahier des charges, j'ai eu des discussions avec mes coéquipiers pour déterminer le modèle que nous allons adopter. Nous avons conclu que la meilleure approche serait de créer un site internet qui servirait de tableau de bord à M. Mégret, le superviseur, lui permettant de visualiser les informations des afficheurs. En plus de cela, nous avons décidé d'inclure une fonctionnalité supplémentaire : la possibilité de modifier le message affiché sur l'afficheur global. Cette modification se ferait à travers une liste déroulante proposant des choix prédefinis, ainsi qu'une zone de texte permettant de saisir un message personnalisé. Une fois le choix effectué, le superviseur serait en mesure de modifier le message affiché. Le site internet constitue donc une solution idéale, car il permet également de consulter l'historique des messages grâce à la base de données, affichant ces informations en annexe sur le site.

#### 2. Contraintes

J'ai rencontré plusieurs contraintes lors de ce projet :

- Pour mon système, il est nécessaire d'utiliser un Raspberry Pi qui communique via le réseau afin de récupérer les données de la base de données. (Il est important de noter qu'il m'a été demandé de créer un compte d'accès spécial en ligne afin de pouvoir me connecter au Raspberry Pi en dehors du réseau local, car sans cela, la connexion serait impossible.)

Afficheur relié au réseau : Ce type d'afficheur communique avec le serveur et va pouvoir disposer de toutes les informations relatives au parking. Il pourra être implanté à n'importe quel endroit du parking. Par exemple, un tel afficheur placé à l'entrée du parking pourra indiquer le nombre total de places restantes, les places disponibles dans chaque zone/étage, ...

- J'ai rencontré un retard important avec mon afficheur, et malheureusement, je n'avais aucune information préalable à son sujet, ce qui m'a empêché de me préparer adéquatement.
- Après avoir reçu l'afficheur avec un certain retard, il y a eu un changement de direction dans mon projet. Initialement, j'avais prévu de communiquer avec l'afficheur via le protocole MQTT. Cependant, il s'est avéré que l'afficheur utilise le protocole RS485, qui est complètement différent. Cette différence de protocole a nécessité des ajustements et des adaptations supplémentaires dans mon projet.

### 3. Protocoles de communication

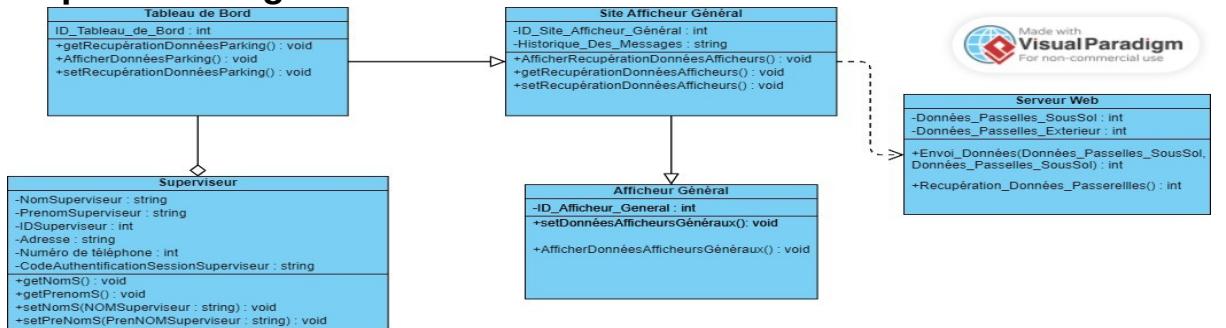
Pour les protocoles de communications, on a :

1. TCP/IP est utilisé pour permettre la communication entre les différents composants de notre configuration, tels que le Raspberry Pi, la base de données centralisée et l'afficheur connecté en RS485.
2. Protocole MySQL : Le Raspberry Pi communique avec la base de données centralisée via le protocole MySQL pour récupérer les données.
3. Protocole HTTP : Le site web exécuté sur le Raspberry Pi envoie des requêtes HTTP vers le serveur Flask via AJAX. Le serveur Flask traite ces requêtes et envoie les réponses correspondantes au site web via le protocole HTTP.
4. Protocole RS485 : L'afficheur est connecté au Raspberry Pi via le protocole RS485. Le Raspberry Pi envoie des commandes ou des données à l'afficheur en utilisant ce protocole pour afficher les informations sur l'afficheur.

En résumé, TCP/IP est utilisé pour la communication réseau générale et la récupération de données, le protocole MySQL est utilisé pour la communication avec la base de données, le protocole HTTP est utilisé pour la communication entre le site web et le serveur Flask, et le protocole RS485 est utilisé pour la communication entre le Raspberry Pi et l'afficheur via l'Arduino Omega.

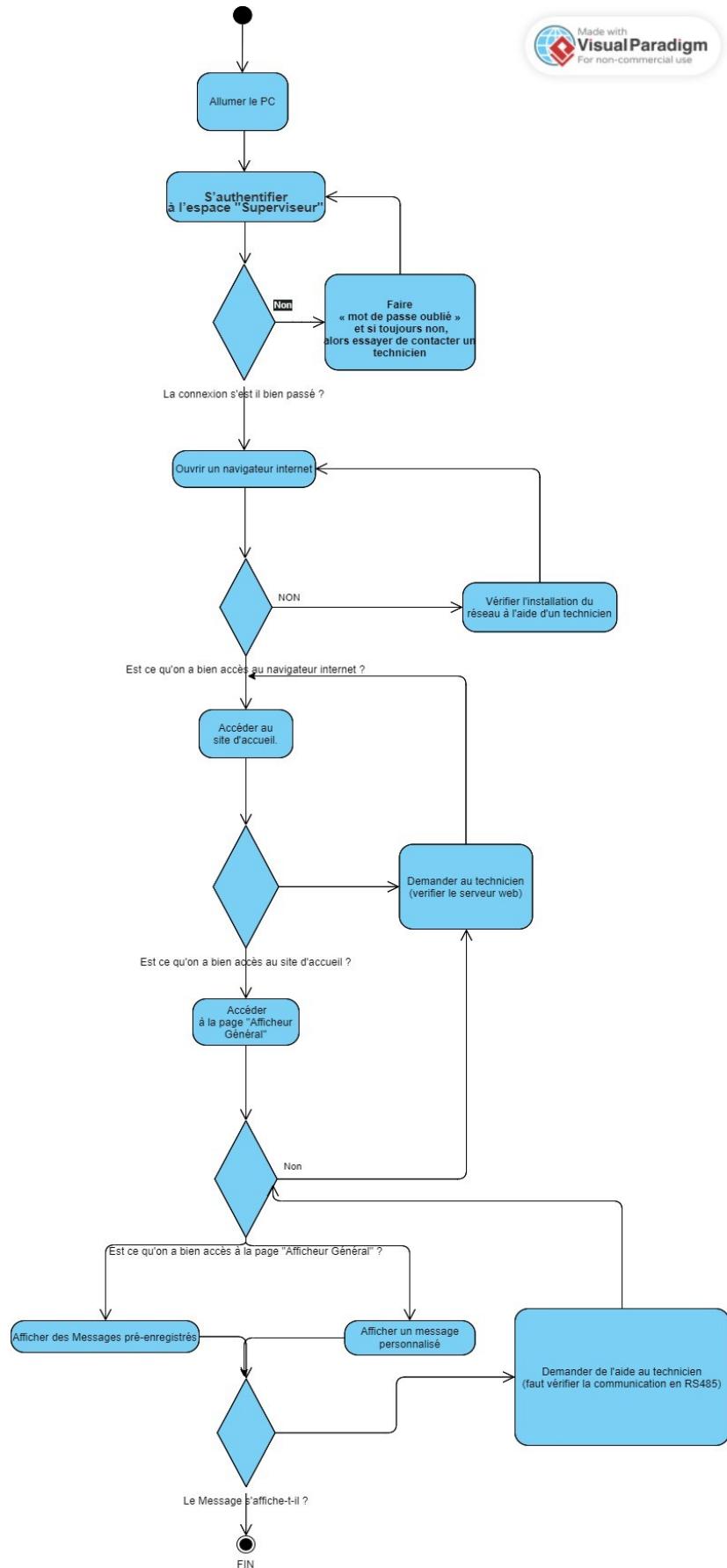
### 4. Analyse UML du système (Personnel)

#### 1. Le premier diagramme de classes

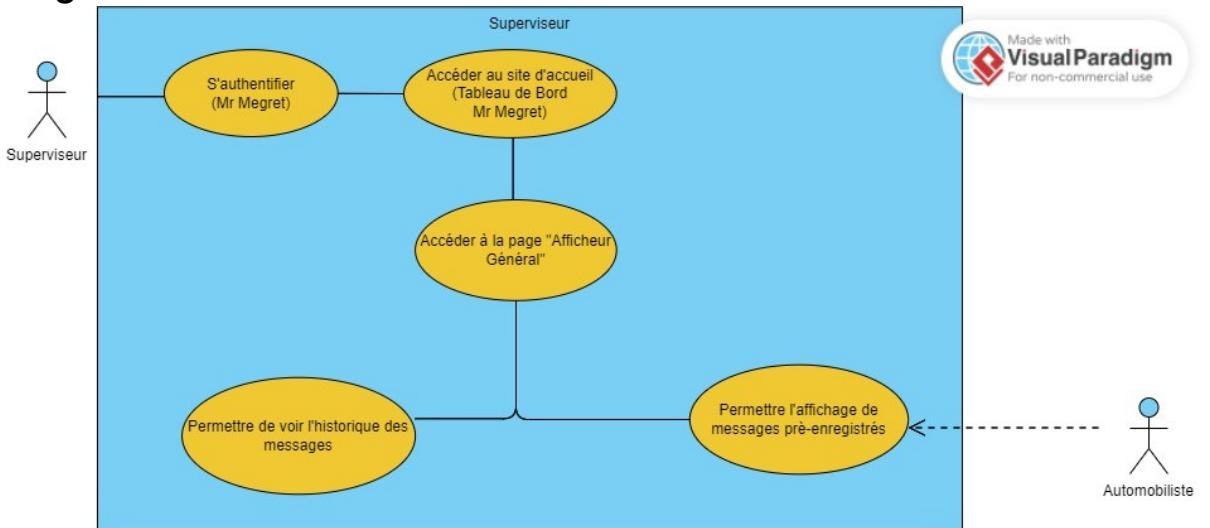


## **2. Diagramme d'activités (séquence)**

Diagramme d'activités  
(Elyes Ghouaiel)



### 3. Diagramme des cas d'utilisations



Voici les descriptions des scénarios de mes cas d'utilisations dans ce diagramme :

Titre :

Accéder à la page "Afficheur

Général"

Permettre de voir l'historique des messages

Permettre l'affichage de messages pré-enregistrés

Résumé :

Pour que le superviseur puisse contrôler les afficheurs généraux et afficher des messages, il doit accéder à la page "Afficheur Général" (depuis le Tableau de Bord).

Pour que le superviseur ait la possibilité de voir l'historique des messages, il le voit à l'aide du serveur web, donc d'abord de la page "Afficheur Général" depuis le Tableau de Bord.

Pour pouvoir afficher des messages pré-enregistrés ou personnalisés de la part du superviseur, il a besoin de la page "Afficheur général", pour que l'acteur secondaire "automobiliste" puisse les voir.

Acteurs principaux :

En acteur principal, on a le superviseur.

En acteur principal, on a le superviseur.

En acteur principal, on a le superviseur.

Acteurs secondaires :

On n'a pas d'acteur secondaire pour ce cas d'utilisation.

On n'a pas d'acteur secondaire pour ce cas d'utilisation.

En acteur secondaire, on a l'automobiliste.

Date de création : 24/01/2023

Date de mise à jour : 28/05/2023

Version : 1.0 Responsable : M. Ghouaiel

#### Description des scénarios

Description pour le cas d'utilisation : « Accéder à la page "Afficheur Général »

##### Préconditions

Le PC doit être allumé.

L'authentification doit être réussi.

L'accès à la page d'accueil est un succès.

L'accès à la page Afficheur Général est un succès.

##### Scénario nominal

1. Le Superviseur doit allumer son PC.
2. Le Superviseur, ensuite, s'authentifie à sa session privée.
3. Ouvrir un navigateur Internet
4. Se connecter au site d'accueil (Tableau de bord).
5. Atteindre la page « Afficheur Général »

##### Scénarios alternatifs

A1 : Le code pour accéder à la session n'est pas valide.

Le scénario nominal reprend au point 1.

A2 : Ne pas avoir accès au réseau

6.Vérifier le câblage.

7.Vérifier les adresses IP, la carte réseau, etc....

Le scénario nominal reprend au point 1.

A3 : La page d'affichage général n'est pas accessible depuis le site d'accueil.

Le scénario nominal reprend au point 4.

#### Description des scénarios

Description pour le cas d'utilisation : « Permettre de voir l'historique des messages »

##### Préconditions

Le PC doit être allumé.

L'authentification doit être réussi.

L'accès à la page (Tableau de bord) est un succès.

L'accès à la page Afficheur Général est un succès.

Visualiser l'historique des messages.

#### Scénario nominal

1. Le Superviseur doit allumer son PC.
2. Le Superviseur, ensuite, s'authentifie à sa session privée.
3. Ouvrir un navigateur Internet.
4. Se connecter au site d'accueil (Tableau de bord).
5. Atteindre la page « Afficheur Général ».
6. Visionner l'historique des messages.

#### Scénarios alternatifs

A1 : Le code pour accéder à la session n'est pas valide.

Le scénario nominal reprend au point 1.

A2 : Ne pas avoir accès au réseau

6. Vérifier le câblage.

7. Vérifier les adresses IP, la carte réseau, etc...

Le scénario nominal reprend au point 1.

A3 : La page d'affichage général n'est pas accessible depuis le site d'accueil.

Le scénario nominal reprend au point 4.

A4 : Ne pas avoir l'historique des messages.

7. Se renseigner sur la BDD.

8. Se renseigner sur le serveur Web

9. Réessayer.

Le scénario nominal reprend au point 5.

#### Description des scénarios

Description pour le cas d'utilisation : « Permettre l'affichage de messages pré-enregistrés »

#### Préconditions

Le PC doit être allumé.

L'authentification doit être réussie.

L'accès à la page d'accueil (Tableau de bord) est un succès.

L'accès à la page Afficheur Général est un succès.

Possibilité d'afficher des messages pré-enregistrés dans les afficheurs généraux.

#### Scénario nominal

1. Le Superviseur doit allumer son PC.
2. Le Superviseur, ensuite, s'authentifie à sa session privée.

3. Ouvrir un navigateur Internet.
4. Se connecter au Tableau de bord.
5. Atteindre la page « Afficheur Général ».
6. Permettre l'affichage de messages pré-enregistrés

#### Scénarios alternatifs

A1 : Le code pour accéder à la session n'est pas valide.

Le scénario nominal reprend au point 1.

A2 : Ne pas avoir accès au réseau

6.Vérifier le câblage.

7.Vérifier les adresses IP, la carte réseau, etc...

Le scénario nominal reprend au point 1.

A3 : La page d'affichage général n'est pas accessible depuis le site d'accueil.

Le scénario nominal reprend au point 4.

A4 : Ne pas avoir l'historique des messages.

7. Se renseigner sur la BDD.

8. Se renseigner sur le serveur Web

9. Réessayer.

Le scénario nominal reprend au point 5.

A5 : Les afficheurs ne marchent pas et pas visible pour les automobilistes.

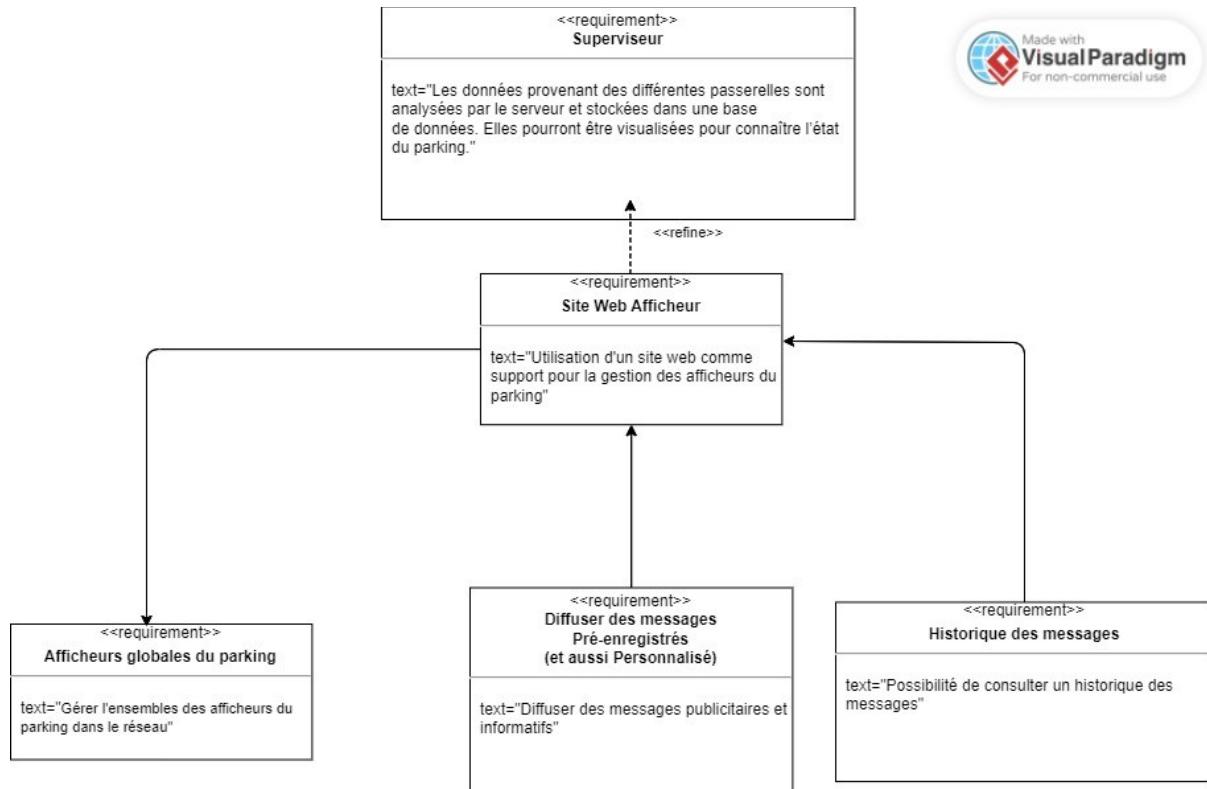
7. Regarder la consommation électrique du panneau.

8. L'état de la batterie

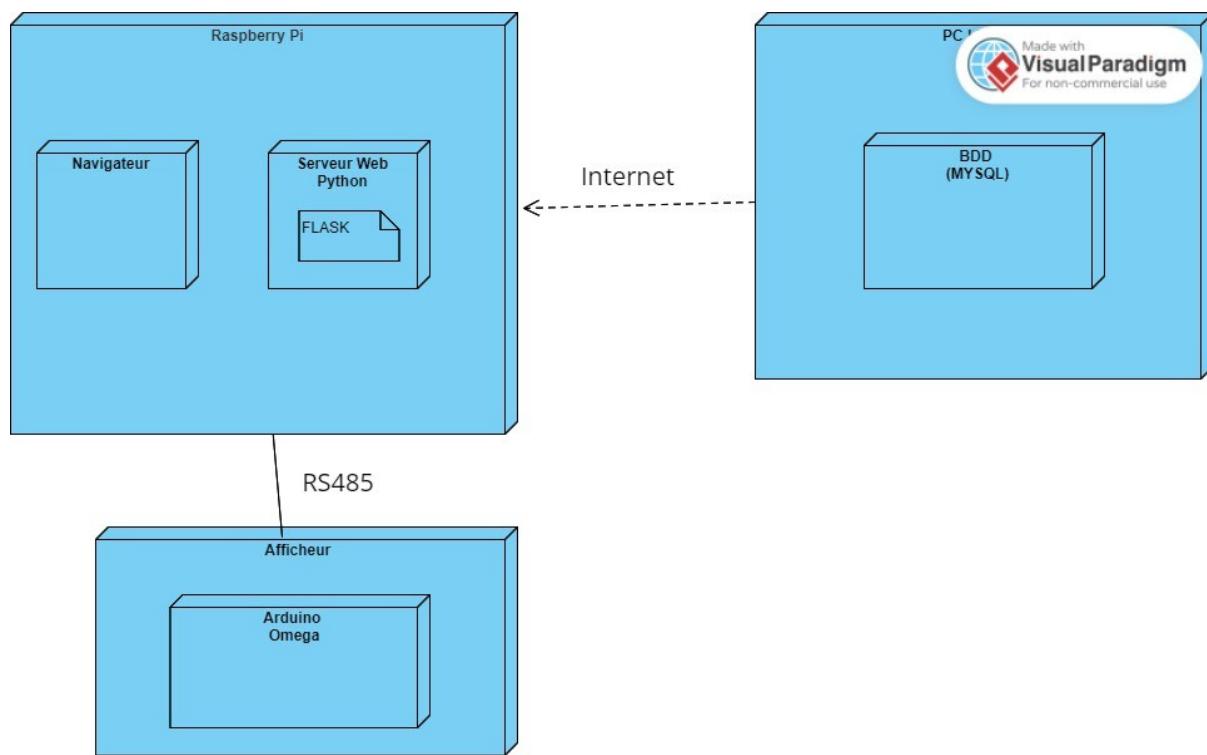
9.Changer l'emplacement du panneau

Le scénario nominal reprend au point 3.

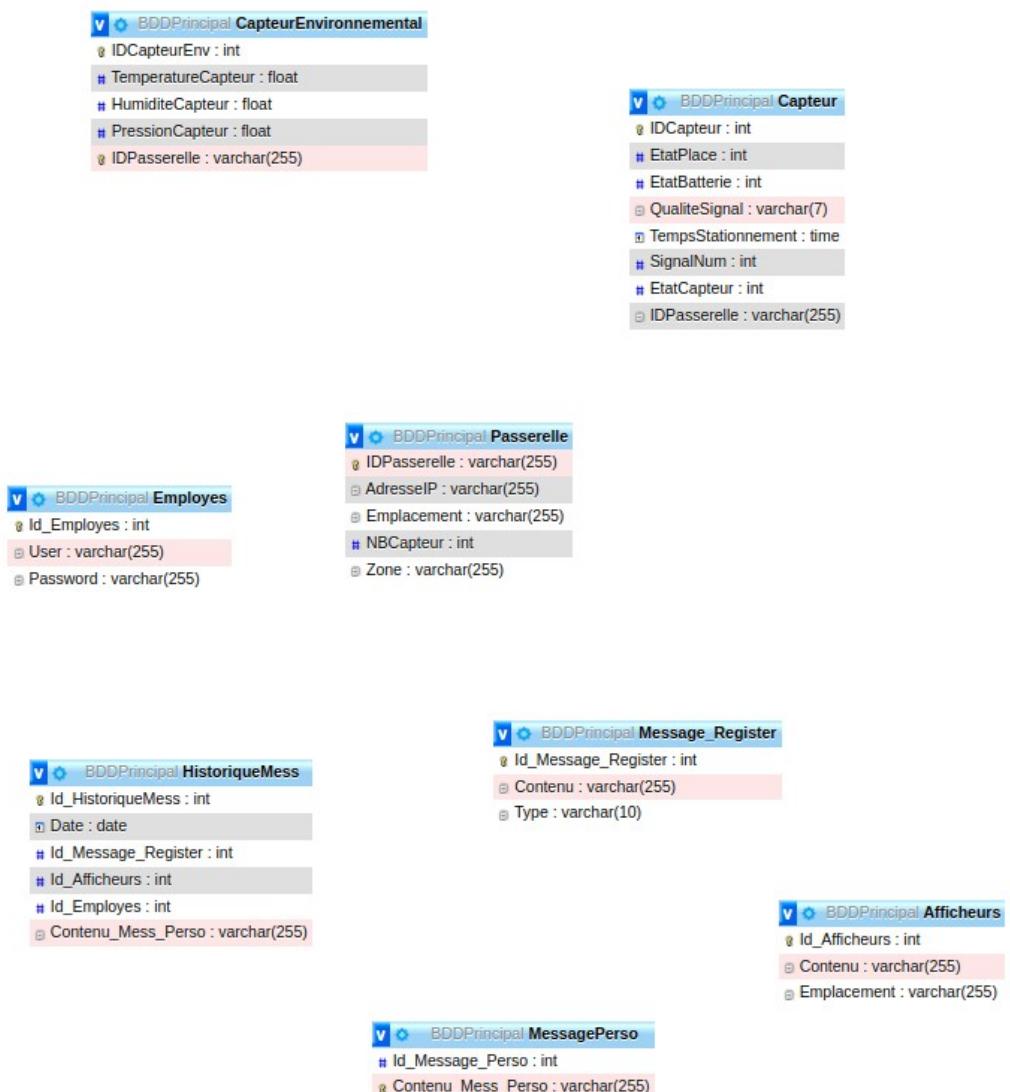
## 4. Diagramme d'exigences



## 5. Diagramme de déploiement



## **6. Base de données Centralisée.**



Contenant le mot :

Table	Action		Lignes	Type	Interclassement	Taille	Perte
Afficheurs							
Capteur							
CapteurEnvironnemental							
Employes							
HistoriqueMess							
MessagePerso							
Message_Register							
Passerelle							
8 tables	Somme					21 InnoDB utf8mb4_general_ci 224,0 kio	0,0

## Début de Projet

Au début du projet, je n'avais pas encore reçu l'afficheur. Cependant, ayant déjà l'idée du site en tête, j'ai décidé de commencer par sa conception. À ce stade, l'IHM (Interface Homme-Machine) du site n'est encore qu'une ébauche préliminaire que j'ai réalisée de mon côté :

The screenshot shows a dashboard titled "Afficheur Général". It contains two main sections: "Afficheur général numéro 1" and "Afficheur général numéro 2". Each section displays parking information and a welcome message. Below each section are buttons for "Personnaliser" and "Valider l'afficheur". To the right of the displays is a "Historique des messages" table with one entry from "Superviseur 1". At the bottom, there is a large area titled "AFFICHEURS" containing two panels: "Afficheurs générales entrée" and "Afficheurs générales sortie", both displaying placeholder text.

Et voilà 2<sup>ème</sup> version :

This screenshot shows the second iteration of the interface. It includes a top navigation bar with icons for menu, EXIT, TDB, and HOME. The main title "AFFICHEURS" is centered above a detailed display for "Afficheur général numéro 1". The display shows parking counts (70 exterior, 20 interior) and a welcome message ("Bienvenue"). Below the display are buttons for "Personnalisés" and "Add to favorites".

Le 17/03, j'ai enfin reçu mon afficheur, après un retard de deux mois par rapport à la date prévue, suite à la première revue de projet. Pour tenter de rattraper mon retard, j'ai pris l'afficheur en main et j'ai entrepris de le comprendre. Cet afficheur fonctionne avec le protocole RS485 et utilise Arduino. Il est équipé de LED et dispose d'un programme de base qui explique son fonctionnement. J'ai consacré beaucoup de temps à son étude, mais j'ai finalement réussi à le comprendre et à identifier les parties les plus importantes nécessaires à son bon fonctionnement.

Tout d'abord, permettez-moi de vous expliquer en détail le fonctionnement de l'afficheur. Ensuite, je pourrai vous donner un aperçu du projet à travers un exemple concret.

### 3. Explication Programme afficheur

La fonction "void Fct\_receptionRS485()" est une partie essentielle de l'afficheur, utilisant deux variables très importantes : "StrCommande" et "dataRecu". Grâce à cette fonction et à deux commandes, je vais vous expliquer le fonctionnement crucial de l'afficheur, ce qui est essentiel pour la suite du projet.

Tout d'abord, permettez-moi de vous montrer comment j'utilise les messages pré-enregistrés du programme initial. Au début, je les commande à l'aide du moniteur série d'Arduino, afin de réaliser les premiers tests. Le schéma est assez simple : "X/X", où le premier "X" représente "StrCommande", qui est une chaîne de caractères. Celle-ci est ensuite convertie en entier (Int) pour devenir une commande. (Le deuxième "X" représente "dataRecu".)

Il est important de noter que l'afficheur fonctionne en utilisant des commandes, plus précisément deux types de commandes : les commandes pour initialiser des variables qui seront utilisées par la suite, et l'autre type de commande est celui qui sert à l'affichage lui-même.

Voici est une introduction pour expliquer la fonction principale du programme de l'afficheur.

La boucle for dans le code parcourt chaque élément du tableau bfr pour extraire les parties pertinentes du message reçu.

Voici comment la boucle fonctionne :

```
for (int i = 0; i < tailleTrame; i++) {  
    if (((char)bfr[i] != '/') && (etape == 0)) {  
        StrCommande += bfr[i];  
    }  
    if (((char)bfr[i] != '/') && (etape == 1)) {  
        dataRecu += bfr[i];  
    }  
    if ((char)bfr[i] == '/') {  
        etape++;  
    }  
}
```

#### Explications :

La variable i est utilisé comme indice pour parcourir chaque élément du tableau bfr.

La première condition `((char)bfr[i] != '/') && (etape == 0)` vérifie si l'élément actuel n'est pas un '/' et si l'étape est égale à 0. Si c'est le cas, cela signifie que nous sommes dans la partie de la commande du message. L'élément est alors ajouté à la variable `StrCommande`.

La deuxième condition `((char)bfr[i] != '/') && (etape == 1)` vérifie si l'élément actuel n'est pas un '/' et si l'étape est égale à 1. Cela signifie que nous sommes dans la partie des données du message. L'élément est ajouté à la variable `dataRecu`.

La troisième condition `(char)bfr[i] == '/'` vérifie si l'élément actuel est un '/'. Si c'est le cas, cela signifie que nous passons de la partie de la commande à la partie des données, ou vice versa. L'étape est alors incrémentée.

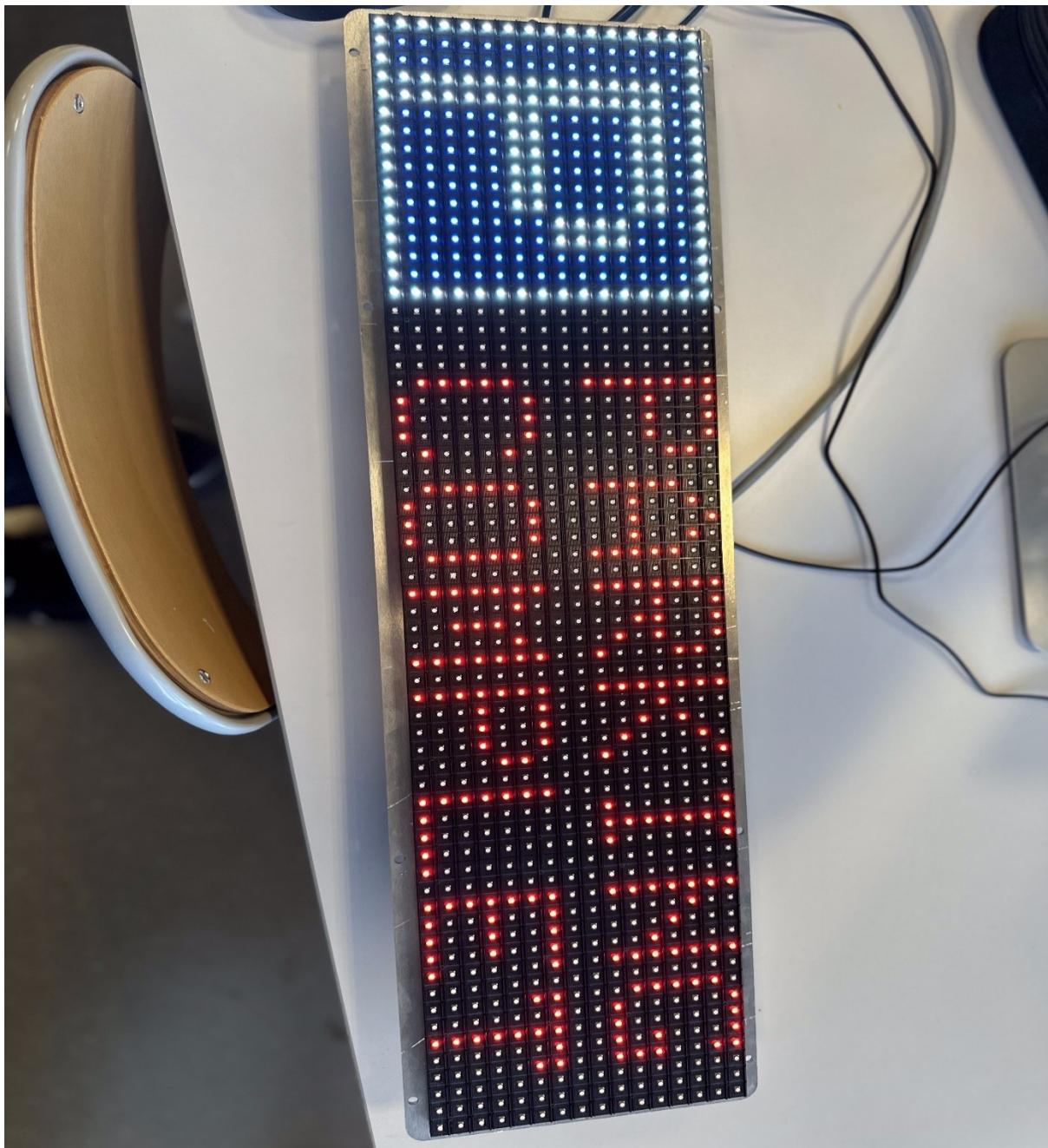
Cette boucle permet de diviser le message reçu en deux parties distinctes : la commande (`StrCommande`) et les données (`dataRecu`). Les parties sont extraites en fonction de la présence du caractère '/' dans le message.

```
commande = StrCommande.toInt();
Serial1.print("commande :");Serial1.println(commande);
Serial1.print("PlacesLibre:");Serial1.println(dataRecu);
switch (commande) {
```

Et après switch (`commande`) on a plein de case avec toutes les commandes disponibles, je vais vous présenter certaines du programme initial et d'autres que j'ai créé moi-même en m'inspirant du programme initial.

## EXEMPLES

Si je tape 12 dans le moniteur série j'obtiens ceci :



```
void Fct_complet2L(){
    matrix.fillScreen(0); //efface l ecran

    matrix.fillRect(0, 0, 16,16, pgm_read_word(&Color[3]));
    matrix.fillRect(1, 1, 14,14, pgm_read_word(&Color[1]));
    matrix.setTextSize(2);
    matrix.setTextColor(pgm_read_word(&Color[3]));
```

```

    matrix.setCursor(3, 2);
    matrix.print("P"); //F2(str)

    matrix.setTextSize(1);

    matrix.setTextColor(pgm_read_word(&Color[2]));
    matrix.setCursor(20, 0);
    matrix.print(F2(parking)); //F2(str)

    matrix.setTextColor(pgm_read_word(&Color[2]));
    matrix.setCursor(20, 9);
    matrix.print(F2(complet)); //F2(str)

    matrix.swapBuffers(false);
}

```

Cette première partie crée le carré parking que l'on voit à gauche de l'afficheur :

La table color que j'utilise :

```

static const uint16_t PROGMEM Color[5] = {

  0x0F00, // vert =0
  0x00F0, // Bleu =1
  0xF000, // Rouge =2
  0xFFFF, // Blanc =3
  0xFFA5 // Ambre =4
};

```

`matrix.fillScreen(0);` : Cette instruction efface l'écran en remplaçant tous les pixels avec la couleur noire (valeur 0), ça réinitialise l'écran.

L'icone « P » :

`matrix.fillRect(0, 0, 16, 16, pgm_read_word(&Color[3]));` : Cette instruction dessine un rectangle rempli aux coordonnées (0, 0) avec une largeur de 16 pixels et une hauteur de 16 pixels. La couleur utilisée pour remplir le rectangle est récupérée à partir de la mémoire flash en utilisant la fonction `pgm_read_word()` et l'indice 3 de la table Color.

`matrix.fillRect(1, 1, 14, 14, pgm_read_word(&Color[1]));` : Cette instruction dessine un rectangle rempli légèrement plus petit que le précédent. Il est positionné aux coordonnées (1, 1) avec une largeur de 14 pixels et une hauteur de 14 pixels. La couleur utilisée est récupérée à partir de la mémoire flash en utilisant `pgm_read_word()` et l'indice 1 de la table Color.

`matrix.setTextSize(2);` : Cette instruction définit la taille du texte à 2. Cela signifie que le texte suivant sera affiché avec une taille de caractère plus grande.

Le message :

`matrix.setTextColor(pgm_read_word(&Color[3]));` : Cette instruction définit la couleur du texte en utilisant la valeur récupérée de la mémoire flash à partir de l'indice 3 de la table Color.

`matrix.setCursor(3, 2);` : Cette instruction positionne le curseur du texte aux coordonnées (3, 2) sur l'écran.

`matrix.print("P");` : Cette instruction affiche la lettre "P" à la position du curseur précédemment définie. Il s'agit probablement d'une abréviation ou d'un symbole spécifique.

Et la seconde partie, pour le message « Le parking complet » sur deux lignes :

`matrix.setTextSize(1);` : Cette instruction rétablit la taille du texte à 1.

`matrix.setTextColor(pgm_read_word(&Color[2]));` : Cette instruction définit la couleur du texte en utilisant la valeur récupérée de la mémoire flash à partir de l'indice 2 de la table Color.

`matrix.setCursor(20, 0);` : Cette instruction positionne le curseur du texte aux coordonnées (20, 0) sur l'écran.

`matrix.print(F2(parking));` : Cette instruction affiche le contenu de la variable parking en utilisant la fonction F2() pour la conversion en une chaîne de caractères. Le texte est affiché à la position du curseur précédemment définie.

`matrix.setTextColor(pgm_read_word(&Color[2]));` : Cette instruction définit à nouveau la couleur du texte en utilisant la valeur récupérée de la mémoire flash à partir de l'indice 2 de la table Color.

`matrix.setCursor(20, 9);` : Cette instruction positionne le curseur du texte aux coordonnées (20, 9) sur l'écran.

`matrix.print(F2(complet));` : Cette instruction affiche le contenu de la variable complet en utilisant la fonction F2() pour la conversion en une chaîne de caractères. Le texte est affiché à la position du curseur précédemment définie.

`matrix.swapBuffers(false);` : Cette instruction échange les tampons d'affichage et met à jour l'écran avec les modifications apportées.

Pour continuer, je vous propose une autre commande, cette fois ci sur une ligne et je tape la commande 4 :



```
case 4:  
    matrix.fillScreen(0);  
    matrix.setTextColor(matrix.Color333(7, 7, 7));  
    matrix.setCursor(0, 0);  
    matrix.print("Travaux");  
    matrix.swapBuffers(false);  
    Fct_receptionRS485();
```

Il est essentiel que la fonction "Fct\_receptionRS485()" se trouve à la fin du programme afin de pouvoir continuer à utiliser d'autres fonctions. Cette fonction est responsable de l'affichage statique sur l'afficheur. Si nécessaire, nous avons la possibilité de modifier ou d'ajouter des fonctionnalités à cet affichage, notamment en ce qui concerne sa taille, pour répondre aux besoins spécifiques du projet.

`matrix.setTextSize(1);` pour la taille qui est à 2 sur l'image.

```
// Move text left (w/wrap), increase hue  
if(--textX < textMin){ textX = matrix.width(); i++;}
```

Cette ligne de code est une condition qui déplace le texte vers la gauche avec un défilement du texte affiché. Voici comment cela fonctionne :

1. La variable `textX` est décrémentée de 1 (`--textX`), ce qui signifie que la position horizontale du texte est déplacée d'un pixel vers la gauche.
2. Ensuite, la condition `textX < textMin` est vérifiée. `TextMin` représente la limite de déplacement vers la gauche du texte. Si `textX` devient inférieur à `textMin`, cela signifie que le texte a atteint cette limite et doit être ramené à la position de départ pour le défilement continu.
3. Si la condition est vraie, `textX` est réinitialisé à `matrix.width()`, ce qui signifie que le texte revient à la position initiale tout à gauche de l'écran. En même temps, la variable `i` est incrémentée (`i++`), ce qui permet d'augmenter la teinte (couleur) du texte affiché.

En résumé, cette partie du code permet de déplacer le texte horizontalement de manière continue vers la gauche avec un effet de défilement de droite à gauche.

J'ai utilisé cet affichage précis, car il sert de maquette pour le système du projet. Maintenant que vous avez compris le fonctionnement de l'afficheur et que vous devez permettre au superviseur de le commander facilement, vous avez été conseillé par votre professeur d'utiliser un Raspberry Pi 3B+ comme une passerelle entre le site web, la base de données (BDD) et l'afficheur. L'utilisation du terme "passerelle" ici fait référence à un pont qui facilite la communication entre ces différents éléments du projet, plutôt qu'au rôle global de "passerelle" dans le projet lui-même.

Je peux vous expliquer quel doit être le rôle de chacun :

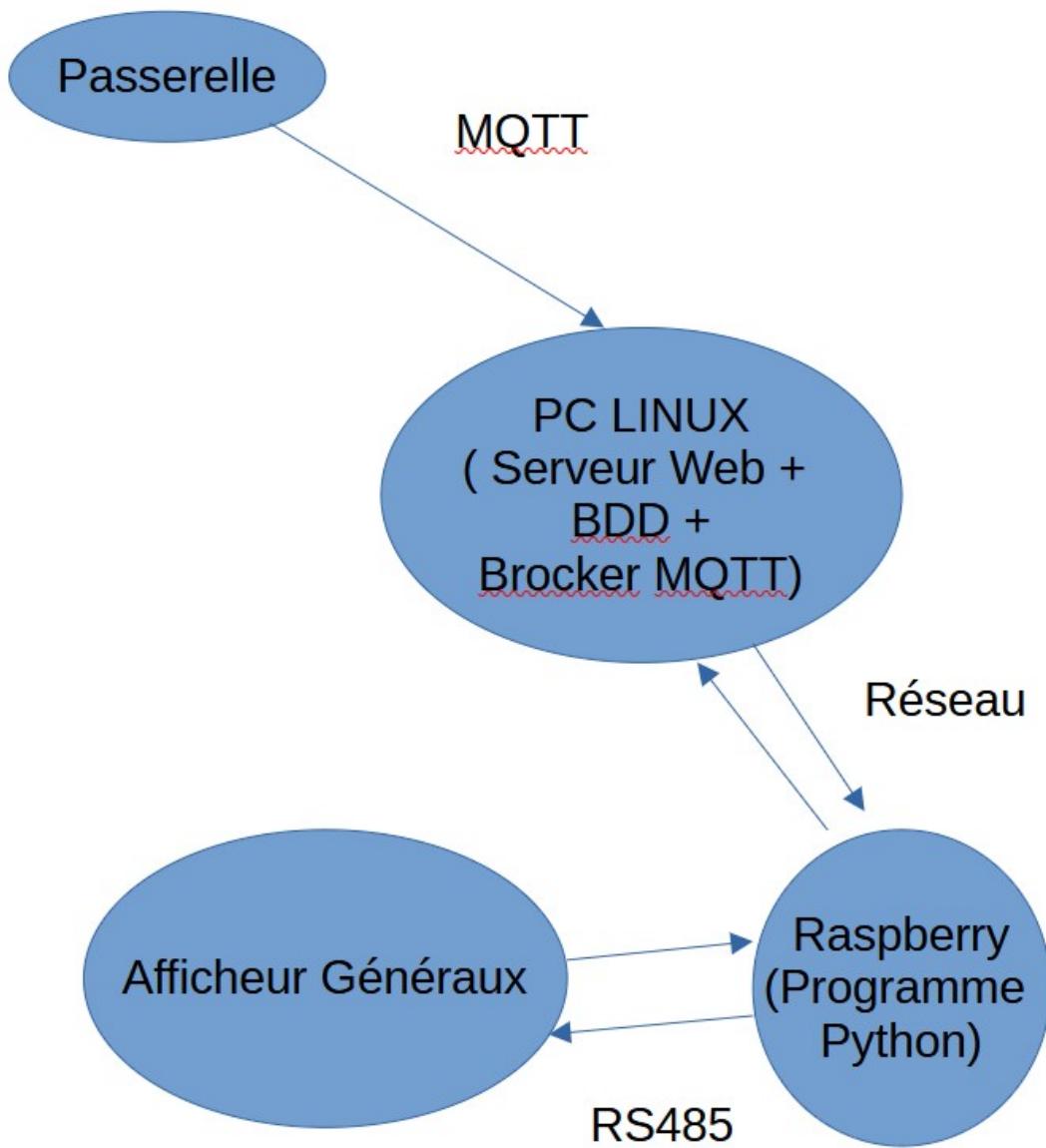
-Le Raspberry Pi joue un rôle central dans le système. Il exécute en continu un programme Python qui permet de réaliser plusieurs tâches. Tout d'abord, il doit récupérer les données de la base de données (BDD), ce qui lui permet d'obtenir les messages pré-enregistrés et autres informations pertinentes. Ensuite, il doit recevoir les données provenant du site web "Afficheur Général" par le biais de requêtes HTTP. Ces données peuvent contenir les choix du superviseur pour l'affichage sur l'afficheur. Ensuite, le Raspberry Pi doit communiquer avec l'afficheur en utilisant le protocole RS485, et cela est rendu possible grâce à un module spécifique. Enfin, le Raspberry Pi doit également être capable de recevoir les données de la BDD par le réseau, afin de maintenir ses informations à jour et de pouvoir fournir des mises à jour pertinentes à l'afficheur.

-**La base de données (BDD) centralisée, gérée par Mr Megret**, joue un rôle essentiel dans le système. Elle doit être capable de stocker les messages pré-enregistrés, qui seront utilisés par l'afficheur. De plus, elle doit permettre d'insérer les messages personnalisés saisis par le superviseur. Pour cela, elle doit avoir une table spécifique appelée "HistoriqueMessage" qui enregistre les messages avec des informations telles que le contenu, la date/heure et d'autres détails pertinents. Cette table d'historique des messages est cruciale pour pouvoir afficher l'historique sur le site web, permettant ainsi au superviseur de consulter les messages précédemment affichés.

-**L'afficheur** joue un rôle clé dans le système, qui est d'afficher le choix sélectionné par le superviseur à partir du site web. Une fois que le superviseur a fait son choix sur le site, le Raspberry Pi, en tant que passerelle, communique avec l'afficheur en utilisant le protocole RS485. L'afficheur reçoit les instructions appropriées pour afficher le message choisi sur son écran. Il est donc responsable de l'affichage physique des messages sur le panneau, permettant ainsi au superviseur et à d'autres personnes de visualiser les informations souhaitées en temps réel.

-**Le site web "Afficheur Général"** joue un rôle crucial dans le système en permettant au superviseur de faire des choix et d'afficher des messages sur l'afficheur. Le site offre une interface conviviale où

le superviseur peut choisir entre afficher un message personnalisé ou sélectionner un message pré-enregistré provenant de la base de données. Une fois que le superviseur a fait son choix, le site envoie les données correspondantes au programme Python s'exécutant sur le Raspberry Pi. Ces données comprennent le contenu du message à afficher ainsi que d'autres informations pertinentes. Le programme Python du Raspberry Pi reçoit ensuite ces données et les utilise pour communiquer avec l'afficheur et afficher le message sélectionné. Ainsi, le site "Afficheur Général" agit comme une interface permettant au superviseur de contrôler facilement l'affichage sur l'afficheur.



Dans le système, j'ai mis en place une table appelée "MessageRegister" qui sert à stocker les messages personnalisés. Chaque message personnalisé est associé à un numéro d'identification unique appelé "Id\_Message\_Register". Ce numéro est important car il doit être transmis au

programme Python qui se charge de l'envoyer au programme Arduino de l'afficheur. Le programme Arduino, à son tour, utilise ce numéro pour traduire la commande correspondante qui affiche le message spécifique sur l'afficheur.

Pour me connecter en ligne au début du projet, j'utilise la commande suivante dans le terminal (cmd) :

```
mysql -h 192.168.1.72 -P 3306 -u AfficheurUser -D bddprincipal -p
```

Cette commande me permet de me connecter à distance à la base de données (BDD) à l'adresse IP 192.168.1.72, sur le port 3306, en utilisant le nom d'utilisateur "AfficheurUser" et la base de données "bddprincipal". Je dois également fournir un mot de passe pour accéder à la BDD. Cela me permet d'interagir avec la BDD pour récupérer et gérer les messages enregistrés dans la table "MessageRegister".

```
mysql> select * from message_register;
+-----+-----+-----+
| Id_Message_Register | Contenu | Type   |
+-----+-----+-----+
|           4 | Travaux | Register |
|           5 | PUB      | Register |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

J'ai également développé une partie du site qui est en lien avec la table "Afficheurs". Bien que j'aie effectué des tests avec cette table, la logique reste la même pour la table "MessageRegister" que j'ai mentionnée précédemment.

Sur le site, j'ai mis en place des fonctionnalités qui permettent de manipuler les données de la table "Afficheurs". Cela inclut la possibilité de visualiser les informations des afficheurs, de les modifier et d'effectuer d'autres opérations associées.

La logique de fonctionnement du site est similaire à celle que j'ai décrite précédemment. L'utilisateur du site, qui est le superviseur, peut sélectionner un afficheur spécifique, choisir un message personnalisé ou pré-enregistré à afficher, et envoyer cette sélection vers le programme Python. Le programme Python se chargera ensuite de transmettre les instructions appropriées à l'afficheur concerné, en utilisant le protocole de communication adapté.

Cela permet au superviseur de contrôler et de gérer les afficheurs à partir du site de manière conviviale et pratique.

```
mysql> select * from afficheurs;
+-----+-----+-----+
| Id_Afficheurs | Contenu          | Emplacement |
+-----+-----+-----+
|           4 | L'étage en plein air est en travaux aujourd'hui | SORTIE    |
|           5 | Voici un message de PUB                         | ENTREE    |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

## Afficheur

<input type="checkbox"/> Case à cocher avec texte personnalisé	<input checked="" type="checkbox"/> Case à cocher pour le message Pré-enrgistré
Personnalisé <input type="button" value="▼"/>	4 - L'étage en plein air est en travaux aujourd'hui - SORTIE 4 - L'étage en plein air est en travaux aujourd'hui - SORTIE 5 - Voici un message de PUB - ENTREE
<input type="button" value="Envoyer"/>	

```
Le message a changé
Numéro de l'afficheur 4 Message : L'étage en plein air est en travaux aujourd'hui
Le message a changé
127.0.0.1 - - [22/May/2023 23:08:12] "POST / HTTP/1.1" 200 -
Le message a changé
Le message a changé
Le message a changé
Le message a changé
Numéro de l'afficheur 5 Message : Voici un message de PUB
Le message n'a pas changé
127.0.0.1 - - [22/May/2023 23:08:17] "POST / HTTP/1.1" 200 -
Le message a changé
Le message a changé
```

Dans le cas des messages personnalisés, le système fonctionne de manière similaire. Cependant, il y a une différence dans la manière dont les messages sont gérés. Tous les messages personnalisés sont attribués à la même valeur, qui est 6 dans ce cas, car cela correspond à la commande qui prend un message en paramètre. Ensuite, la valeur 7 est utilisée pour afficher le message sur l'afficheur.

Cela signifie que dans le programme, il sera nécessaire d'effectuer une adaptation supplémentaire pour gérer cette particularité des messages personnalisés. Il peut s'agir d'ajouter une condition spécifique pour les messages ayant la valeur 6 et de définir les actions correspondantes, telles que l'envoi du message à l'afficheur.

Il est important de prendre en compte cette différence lors du développement du programme afin de s'assurer que les messages personnalisés sont correctement traités et affichés sur l'afficheur.

```
case 6:
    afficherMessage(Mess);
break;
case 7:
    Mess=dataRecu;
Break;
```

## Afficheur

<input checked="" type="checkbox"/>	Case à cocher avec texte personnalisé
<input type="text" value="yes"/>	
<input type="button" value="Personnalisé"/>	
<input type="button" value="Envoyer"/>	

Je vous ai donné les résultats mais voici les explications du site « afficheur général » et le programme python :

## 4. SITE AFFICHEUR

On commence par le site « Afficheur général » : le site je le divise en deux avec le message personnalisé à gauche et à droite les messages personnalisés voici un aperçu complet :

The screenshot shows a dark-themed application window titled 'Afficheur'. It contains two separate sections for personalizing messages. The left section has a checked checkbox labeled 'Case à cocher avec texte personnalisé', a dropdown menu set to 'Personnalisé' with an option 'yes' highlighted, and a 'Envoyer' button. The right section has a checked checkbox labeled 'Case à cocher pour le message Pré-enregistré', a dropdown menu set to '4 - L'étage en plein air est en travaux aujourd'hui - SORTIE', and a 'Envoyer' button.

-Pour la partie personnalisée, en résumé, ce code génère une section contenant une case à cocher, un texte personnalisé, une zone de liste déroulante avec une option, une zone de texte et un bouton pour envoyer un message personnalisé.

```
<div class="section">
    <input type="checkbox" id="customCheckbox">
    <label for="customCheckbox" class="checkbox-label">Case à cocher avec texte personnalisé</label>
    <div class="dropdown" id="customDropdown">
        <select>
            <option value="Personnalisé">Personnalisé</option>
        </select>
        <textarea id="messageTextarea"></textarea>
    </div>
    <button class="submit-button"
    onclick="envoyerMessagePerso()">Envoyer le message personnalisé</button>
</div>
```

```
function envoyerMessagePerso() {
    // Récupérer la valeur du champ de texte personnalisé
    var customMessage =
document.getElementById('messageTextarea').value;

    // Envoyer le message personnalisé via une requête AJAX
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "http://127.0.0.1:5000/", true); // Modifier l'URL si nécessaire
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.onreadystatechange = function () {
        if (xhr.readyState === XMLHttpRequest.DONE && xhr.status ===
200) {
            // Traitement de la réponse
            console.log(xhr.responseText);
        }
    };

    // Concaténer le message personnalisé dans le corps de la requête
}
```

```

        var data = "customMessage=" + encodeURIComponent(customMessage);

        xhr.send(data);
    }

```

La fonction envoyerMessagePerso() est une fonction JavaScript qui est appelée lorsqu'on clique sur le bouton "Envoyer le message personnalisé". Voici ce que fait cette fonction :

var customMessage = document.getElementById('messageTextarea').value; : Cette ligne de code récupère la valeur saisie dans la zone de texte avec l'identifiant "messageTextarea" et la stocke dans la variable customMessage. Cela permet de récupérer le message personnalisé saisi par l'utilisateur.

var xhr = new XMLHttpRequest(); : Cela crée un nouvel objet XMLHttpRequest, qui est utilisé pour effectuer des requêtes HTTP asynchrones.

xhr.open("POST", "http://127.0.0.1:5000/", true);: Cette ligne spécifie les détails de la requête. Elle indique que la requête sera de type POST, et l'URL cible est "http://127.0.0.1:5000/". Vous pouvez modifier cette URL en fonction de vos besoins. Le dernier argument true spécifie que la requête sera asynchrone.

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); : Cette ligne définit l'en-tête de la requête HTTP. Elle spécifie le type de contenu de la requête comme "application/x-www-form-urlencoded", ce qui est couramment utilisé pour envoyer des données de formulaire.

xhr.onreadystatechange = function () { ... } : Cela définit une fonction qui sera appelée chaque fois que l'état de la requête change. À chaque changement d'état, la fonction vérifie si la requête est terminée (readyState === XMLHttpRequest.DONE) et si le statut de la réponse est 200 (ce qui signifie que la requête a réussi). Si ces conditions sont remplies, la fonction effectue un traitement supplémentaire. Dans cet exemple, la réponse de la requête est affichée dans la console à l'aide de console.log(xhr.responseText);.

var data = "customMessage=" + encodeURIComponent(customMessage); : Cette ligne crée une chaîne de requête avec le message personnalisé encodé à l'aide de encodeURIComponent(). La chaîne de requête est utilisée pour envoyer les données dans le corps de la requête HTTP.

xhr.send(data); : Cette ligne envoie la requête HTTP avec les données spécifiées dans data en tant que corps de la requête.

-Pour la partie messages pré-enregistrés, on a :

```

<div class="section">
    <input type="checkbox" id="definitCheckbox">
    <label for="definitCheckbox" class="checkbox-label">Case à cocher
pour le message Pré-enrgistré</label>
    <div class="dropdown" id="definitDropdown">
        <?php
        // Informations de connexion à la base de données
        $servername = '10.198.214.55';
        $port = 3306;
        $username = 'AfficheurUser';
        $password = 'Afficheur';
        $dbname = 'BDDPrincipal';

```

```

        // Connexion à la base de données
        $conn = new mysqli($servername, $username, $password,
$dbname);

        // Vérification de la connexion
        if ($conn->connect_error) {
            die("Connexion échouée : " . $conn->connect_error);
        }

        // Requête SQL pour récupérer les données de la table
"Message_Register"
        $sql = "SELECT Id_Message_Register, Contenu, Type FROM
Message_Register";
        $result = $conn->query($sql);

        // Vérification des résultats de la requête
        if ($result->num_rows > 0) {
            echo '<select name="messages" id="messages">';
            while ($row = $result->fetch_assoc()) {
                $id = $row['Id_Message_Register'];
                $Contenu = $row['Contenu'];
                $Type = $row['Type'];
                echo '<option value="' . $id . '">' . $id . ' - ' .
$Contenu . ' - ' . $Type . '</option>';
            }
            echo '</select>';
        } else {
            echo 'Aucun résultat trouvé.';
        }

        // Fermeture de la connexion à la base de données
        $conn->close();
    ?>
</div>
<button class="submit-button" onclick="envoyerDonnees()">Envoyer
un Message Pré-enregistrés</button>
</div>

```

En résumé, ce code crée une section dans laquelle vous trouverez une case à cocher, une étiquette, une zone de liste déroulante préremplie avec des données provenant d'une base de données, et un bouton permettant d'envoyer un message pré-enregistré. Le code PHP est utilisé pour récupérer les données de la base de données et les afficher dans la liste déroulante. Ainsi, le superviseur pourra facilement sélectionner l'un des messages pré-enregistrés disponibles et l'envoyer en utilisant le bouton correspondant.

```

function envoyerDonnees() {
    // Récupérer les valeurs sélectionnées

```

```

        var selectElement = document.getElementById("messages");
        var Id_Message_Register = selectElement.value;
        var Contenu =
selectElement.options[selectElement.selectedIndex].text.split(" - ")[1];
        var Type =
selectElement.options[selectElement.selectedIndex].text.split(" - ")[2];

        // Envoyer les données via une requête AJAX
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "http://127.0.0.1:5000/", true); // Modifier
l'URL si nécessaire
        xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
        xhr.onreadystatechange = function () {
            if (xhr.readyState === XMLHttpRequest.DONE && xhr.status ===
200) {
                // Traitement de la réponse
                console.log(xhr.responseText);
            }
        };
        // Concaténer les données à envoyer dans le corps de la requête
        var data = "Id_Message_Register=" + Id_Message_Register +
"&Contenu=" + encodeURIComponent(Contenu);

        xhr.send(data);
    }
}

```

En résumé, cette fonction permet de récupérer les valeurs sélectionnées dans la liste déroulante, telles que l'ID du message pré-enregistré, son contenu et son type. Ensuite, ces données sont envoyées à un serveur à l'aide d'une requête AJAX de type POST. Cela permet de transmettre les informations nécessaires au traitement ultérieur du message sélectionné.

## 5. Programme Arduino

Enfin on va pouvoir parler du programme python , très important dans notre système.

```

import mysql.connector
import time
# import serial
from flask import Flask, request
import threading

# Informations de connexion
config = {
    'user': 'AfficheurUser',
    'password': 'Afficheur',
    'host': '10.198.214.55',
    'database': 'BDDPrincipal'
}

```

```
app = Flask(__name__)
```

Les bibliothèques utilisées dans le code ont des rôles spécifiques pour le traitement des données reçues du site et la communication avec une base de données. Voici une explication de l'utilité de chaque bibliothèque par rapport au site qui envoie les données :

`mysql.connector` : Cette bibliothèque est utilisée pour établir une connexion avec une base de données MySQL. Dans ce cas, elle est utilisée pour se connecter à la base de données qui stocke les données du site. L'utilisation de `mysql.connector` permet d'effectuer des requêtes SQL pour récupérer, insérer ou mettre à jour les données de la base de données.

`time` : La bibliothèque `time` est utilisée pour gérer les délais d'attente ou les temporisations dans le code. Dans le contexte du site qui envoie les données, elle peut être utilisée pour introduire des délais entre les opérations, comme attendre un certain temps avant de récupérer ou d'envoyer des données.

`flask` : Flask est un framework web léger pour Python qui facilite la création d'applications web. Dans le contexte du site qui envoie les données, Flask est utilisé pour créer un serveur web qui écoute les requêtes HTTP entrantes et y répond. Il permet de définir des routes pour différentes URL et de définir des fonctions qui sont exécutées lorsque ces routes sont accédées. Ainsi, le code Flask peut gérer les requêtes provenant du site et effectuer des actions spécifiques en réponse à ces requêtes.

`threading` : La bibliothèque `threading` permet de gérer les threads en Python. Dans le contexte du site qui envoie les données, `threading` peut être utilisé pour exécuter certaines tâches de manière asynchrone et simultanée. Par exemple, il est possible d'utiliser des threads pour gérer des opérations longues ou bloquantes sans bloquer l'exécution du reste du programme. Cela permet de maintenir la réactivité de l'application en continuant à traiter d'autres requêtes tout en effectuant des opérations intensives en arrière-plan.

En résumé, les bibliothèques utilisées dans le code sont sélectionnées pour leurs fonctionnalités spécifiques qui facilitent la communication avec une base de données, la gestion des délais, la création d'un serveur web et la gestion des tâches simultanées. Elles permettent au site qui envoie les données d'interagir avec la base de données, de répondre aux requêtes HTTP et d'exécuter des tâches de manière efficace et réactive.

```
@app.route('/', methods=['POST'])
```

La ligne `@app.route('/', methods=['POST'])` est un décorateur utilisé dans Flask pour définir une route au sein de l'application web. Voici une explication de son fonctionnement :

- `@app.route('/')` : Le décorateur `@app.route()` est utilisé pour définir une route dans Flask. Entre les parenthèses, on spécifie le chemin de l'URL associé à la route. Dans ce cas, le chemin est `'/'`, ce qui signifie que la route correspond à la racine de l'application, c'est-à-dire la page d'accueil.

- `methods=['POST']` : L'argument `methods` du décorateur `@app.route()` spécifie les méthodes HTTP autorisées pour cette route. Dans ce cas, seule la méthode HTTP POST est autorisée. Cela signifie que cette route ne répondra qu'aux requêtes POST et ignorerá les autres méthodes telles que GET, PUT, DELETE, etc.

En résumé, la ligne `@app.route('/', methods=['POST'])` indique que la fonction qui suit ce décorateur sera exécutée lorsque la route de la page d'accueil est accédée par une requête HTTP POST. Cela permet de définir une action spécifique à exécuter lorsque des données sont envoyées depuis le site vers cette route.

Maintenant je vous donne toutes les fonctions sachant qu'il y en a , elle sont pas encore utilisés car au moment où j'écris ceci , le panneau n'est pas encore connecté au raspberry :

```
def recevoir_donnees():
```

```
    Id_Message_Register = request.form.get('Id_Message_Register')
```

```

contenu = request.form.get('Contenu')

# Récupérer le contenu du message personnalisé
custom_message = request.form.get('customMessage')

# Insérer le message personnalisé dans la table MessagePerso
mettre_a_jour_message_personnalise(custom_message)

# Mettre à jour les données dans votre base de données
update_data(Id_Message_Register, contenu)

# Récupérer le choix sélectionné
choix = retrieve_choice(Id_Message_Register)

# Envoyer le choix à l'Arduino
if choix is not None:
    print("Numéro de message Pré-éregistrés {} Message :"
{}".format(Id_Message_Register, choix))
    # send_to_arduino(choix)

# Après l'insertion du message personnalisé dans la table MessagePerso
if custom_message:
    print("Message personnalisé : {}".format(custom_message))

# Comparer le choix avec le dernier message affiché
global dernier_message
if choix != dernier_message:
    print("Le message a changé")
    dernier_message = choix
else:
    print("Le message n'a pas changé")

return 'Données reçues avec succès !'

```

1. `recevoir_donnees()`: Cette fonction est appelée lorsque des données sont envoyées via une requête POST à la racine de l'application. Elle récupère les données envoyées, met à jour le message personnalisé dans la base de données, met à jour les données dans la base de données principale, récupère le choix sélectionné. (Et plus tard enverra ce choix à un dispositif Arduino connecté.)

```

def mettre_a_jour_message_personnalise(message):
    try:
        # Informations de connexion à la base de données
        host = '10.198.214.55'
        user = 'AfficheurUser'
        password = 'Afficheur'
        database = 'BDDPrincipal'

        # Établir la connexion à la base de données

```

```

conn = mysql.connector.connect(host=host, user=user,
password=password, database=database)

# Créer un curseur pour exécuter les requêtes SQL
cursor = conn.cursor()

# Requête SQL pour mettre à jour le message personnalisé dans la table
MessagePerso
sql = "UPDATE MessagePerso SET Contenu_Mess_Perso = %s WHERE
Id_Message_Perso = %s"
values = (message, 6) # Utiliser l'ID correspondant à votre message
personnalisé

# Exécuter la requête SQL
cursor.execute(sql, values)

# Valider les modifications dans la base de données
conn.commit()

# Fermer le curseur et la connexion
cursor.close()
conn.close()

print("Message personnalisé mis à jour dans la table MessagePerso avec
succès.")

except mysql.connector.Error as error:
    print("Erreur lors de la mise à jour du message personnalisé dans la
table MessagePerso :", error)

```

2. mettre\_a\_jour\_message\_personnalise(message): Cette fonction met à jour le message personnalisé dans la table MessagePerso de la base de données. Elle se connecte à la base de données, exécute une requête SQL pour mettre à jour le message personnalisé, valide les modifications et ferme la connexion.

```

def update_data(Id_Message_Register, contenu):
    # Se connecter à la base de données
    cnx = mysql.connector.connect(**config)
    cursor = cnx.cursor()

    # Exécuter une requête SQL pour mettre à jour les données
    query = "UPDATE Message_Register SET Contenu = %s WHERE
Id_Message_Register = %s"
    values = (contenu, Id_Message_Register)
    cursor.execute(query, values)

    # Valider la transaction

```

```
cnx.commit()

# Fermer la connexion
cursor.close()
cnx.close()

def retrieve_choice(Id_Message_Register):
    # Se connecter à la base de données
    cnx = mysql.connector.connect(**config)
    cursor = cnx.cursor()

    # Exécuter une requête SQL
    query = "SELECT Contenu FROM Message_Register WHERE Id_Message_Register = %s"
    values = (Id_Message_Register,)
    cursor.execute(query, values)

    # Récupérer le résultat
    result = cursor.fetchone()

    # Fermer la connexion
    cursor.close()
    cnx.close()

    if result is not None:
        return result[0]
    else:
        return None

def retrieve_all_data():
    # Se connecter à la base de données
    cnx = mysql.connector.connect(**config)
    cursor = cnx.cursor()

    # Exécuter une requête SQL pour récupérer toutes les données
    query = "SELECT Id_Message_Register, Contenu, Type FROM Message_Register"
    cursor.execute(query)

    # Récupérer tous les résultats
    data = cursor.fetchall()

    # Fermer la connexion
    cursor.close()
    cnx.close()

    return data
```

Voici un résumé de chaque fonction :

-update\_data(Id\_Message\_Register, contenu): Cette fonction met à jour les données dans la table Message\_Register de la base de données principale. Elle se connecte à la base de données, exécute une requête SQL pour mettre à jour les données, valide la transaction et ferme la connexion.

-retrieve\_choice(Id\_Message\_Register): Cette fonction récupère le choix sélectionné dans la table Message\_Register de la base de données principale. Elle se connecte à la base de données, exécute une requête SQL pour récupérer le contenu du choix, récupère le résultat et ferme la connexion.

-retrieve\_all\_data(): Cette fonction récupère toutes les données de la table Message\_Register de la base de données principale. Elle se connecte à la base de données, exécute une requête SQL pour récupérer toutes les données, récupère tous les résultats et ferme la connexion.

Dans l'ensemble, ces fonctions sont utilisées pour recevoir des données, les traiter, les mettre à jour dans la base de données, récupérer des informations de la base de données, surveiller les changements de messages, et communiquer avec un dispositif Arduino. Et plus tard , il enverra dans le programme de l'afficheur, c'est l'objectif.

## II. Etude Physique

Je vous remets les missions qui m'ont été confiées.

Etudiants	Missions	Matériels ou logiciels SPÉCIFIQUES
Étudiant 2 : Informatique et réseau	<b>Vous êtes chargé de :</b> <ul style="list-style-type: none"><li>- De gérer l'ensemble des afficheurs du parking<ul style="list-style-type: none"><li>o Possibilité de choisir un afficheur dans le réseau et de diffuser des messages d'information ou publicitaire.</li><li>o Possibilité de consulter un historique des messages.</li></ul></li></ul>	<ul style="list-style-type: none"><li>- Raspberry PI</li><li>- Docker MQTT</li></ul>
Étudiant 1 : Physique appliquée	<b>Vous êtes chargé de :</b> <ul style="list-style-type: none"><li>- D'étudier la consommation électrique du système en plein air</li><li>- De faire des expériences et mesures</li></ul>	<ul style="list-style-type: none"><li>-Ensemble des afficheurs</li><li>- Plot LORA Bosch TPS110EU</li><li>- Matériel de mesures</li></ul>

Pour la Partie Physique, je me suis renseigné auprès de mes collègues qui travaillent sur le système en plein air du parking, donc j'ai des informations mais pas encore fait d'expériences. Je tiens juste à préciser que de leur côté aussi, le système en plein air, ça a beaucoup changé de matériels mais je vais donner tout ce que je sais.

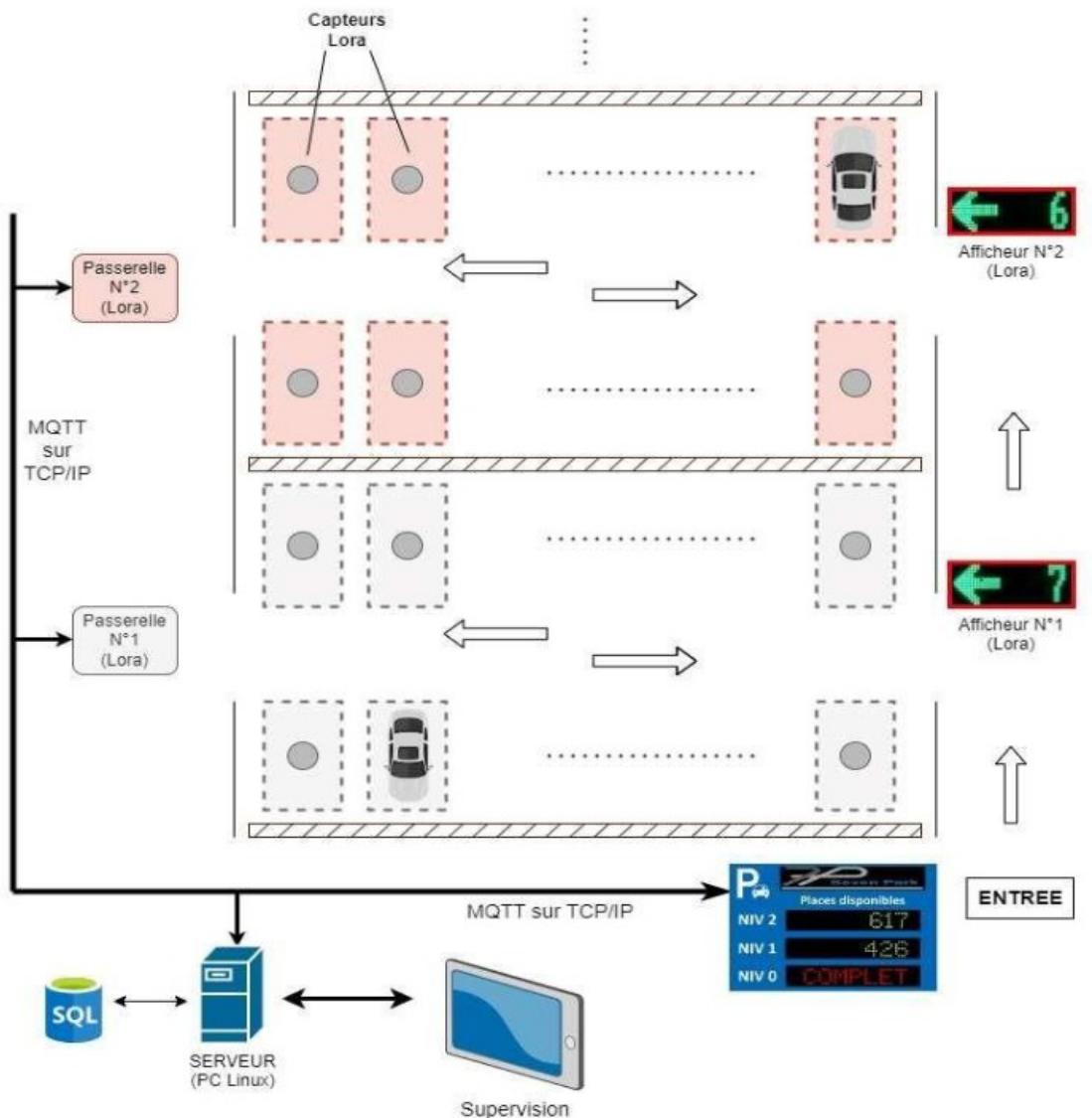
Voici le système expliqué par une image :

Académie de Nice

Comptage à la place

Projet technique U62

##### **5. Schéma général du système pour parking extérieur**



Voici les informations complètes concernant le courant, la tension et la puissance de chaque appareil, ainsi que le calcul de la consommation électrique (les informations sortent d'internet):

###### 1. Capteur BME280 :

- Tension d'alimentation VDDIO : 1,2 ... 3,6 V
- Tension d'alimentation VDD : 1,71 ... 3,6 V
- Courant moyen de consommation (typ.) (Taux de rafraîchissement des données à 1 Hz) :
  - 1,8 µA @ 1 Hz (H, T)
  - 2,8 µA @ 1 Hz (P, T)
  - 3,6 µA @ 1 Hz (H, P, T)

- Courant moyen de consommation en mode veille : 0,1 µA

## 2. Afficheur G-Energy N200V5-A 5V40A 200W LED Display Power Supply:

- Puissance de sortie : 200 W
- Tension d'entrée : 200-240 V CA, 47-63 Hz
- Courant de fuite : 0,30 mA @ 230 V CA

## 3. Arduino Omega :

- Tension de fonctionnement : 5 V

## 4. Raspberry Pi 3B+ (x2):

- Processeur: Cortex-A53 1,4 GHz (Broadcom BCM2837)
- Quad Core
- Alimentation : Externe, via port micro USB 5V (2A)

## 5. Puces RFM96/RFM98 :

- Modem Lora
- Budget de liaison maximal : 168 dB
- Sortie RF constante de +20 dBm -100 mW par rapport à Vsupply
- PA haute efficacité de +14 dBm
- Courant de réception faible de 10,3 mA
- Rétention du registre : 200 nA

## 6. Arduino Nano :

- Alimentation :
  - Via port USB
  - 5 Vcc régulés sur la broche 27
  - 6 à 20 V non régulés sur la broche 30
- Microprocesseur : ATMega328
- Mémoire flash : 32 kB
- Mémoire SRAM : 2 kB
- Mémoire EEPROM : 1 kB
- Interfaces :
  - 14 broches d'E/S dont 6 PWM
  - 8 entrées analogiques 10 bits
  - Bus série, I2C et SPI
- Intensité par E/S : 40 mA
- Cadencement : 16 MHz

Maintenant, je vais calculer la consommation électrique de chaque appareil en watts-heure (Wh) pour une durée d'utilisation de 11 heures par jour. Veuillez noter que les puces RFM96/RFM98 et les Raspberry Pi 3B+ nécessitent des informations supplémentaires pour effectuer ces calculs.

1. Capteur BME280 :

- Consommation moyenne à 1 Hz (H, T) : 1,8 µA @ 1 Hz

$$\text{Puissance} = (1,8 \mu\text{A}) \times (3,6 \text{ V}) = 6,48 \mu\text{W}$$

$$\text{Consommation électrique par jour} = (6,48 \mu\text{W}) \times (11 \text{ h}) = 71,28 \mu\text{Wh}$$

$$\text{Consommation électrique par an} = (71,28 \mu\text{Wh}) \times (365 \text{ jours}) = 26,02 \text{ mWh}$$

2. Afficheur G-Energy N200V5-A 5V40A 200W LED Display Power Supply:

- Puissance de sortie : 200 W

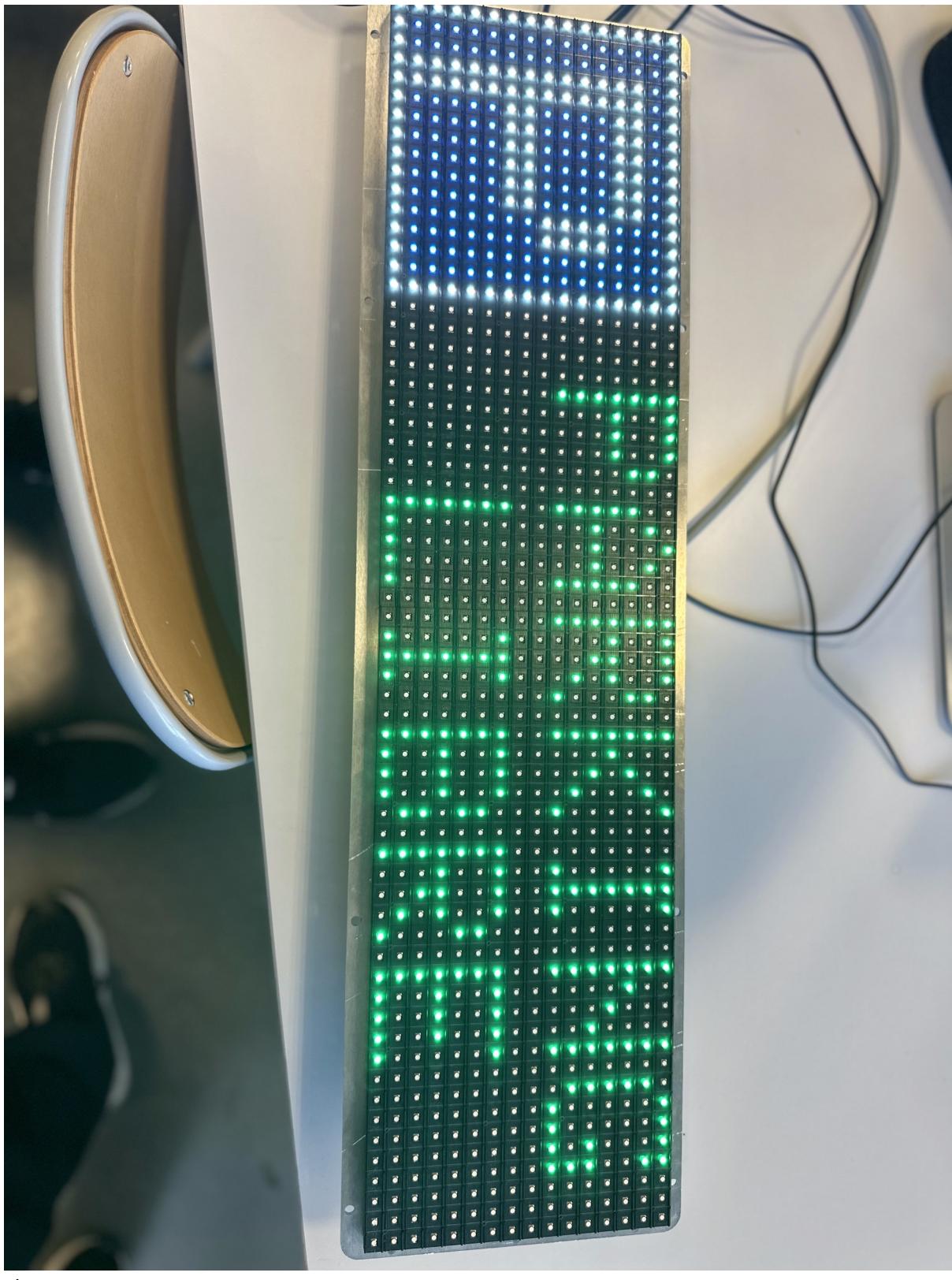
$$\text{Consommation électrique par jour} = (200 \text{ W}) \times (11 \text{ h}) = 2200 \text{ Wh} = 2,2 \text{ kWh}$$

$$\text{Consommation électrique par an} = (2,2 \text{ kWh}) \times (365 \text{ jours}) = 803 \text{ kWh}$$

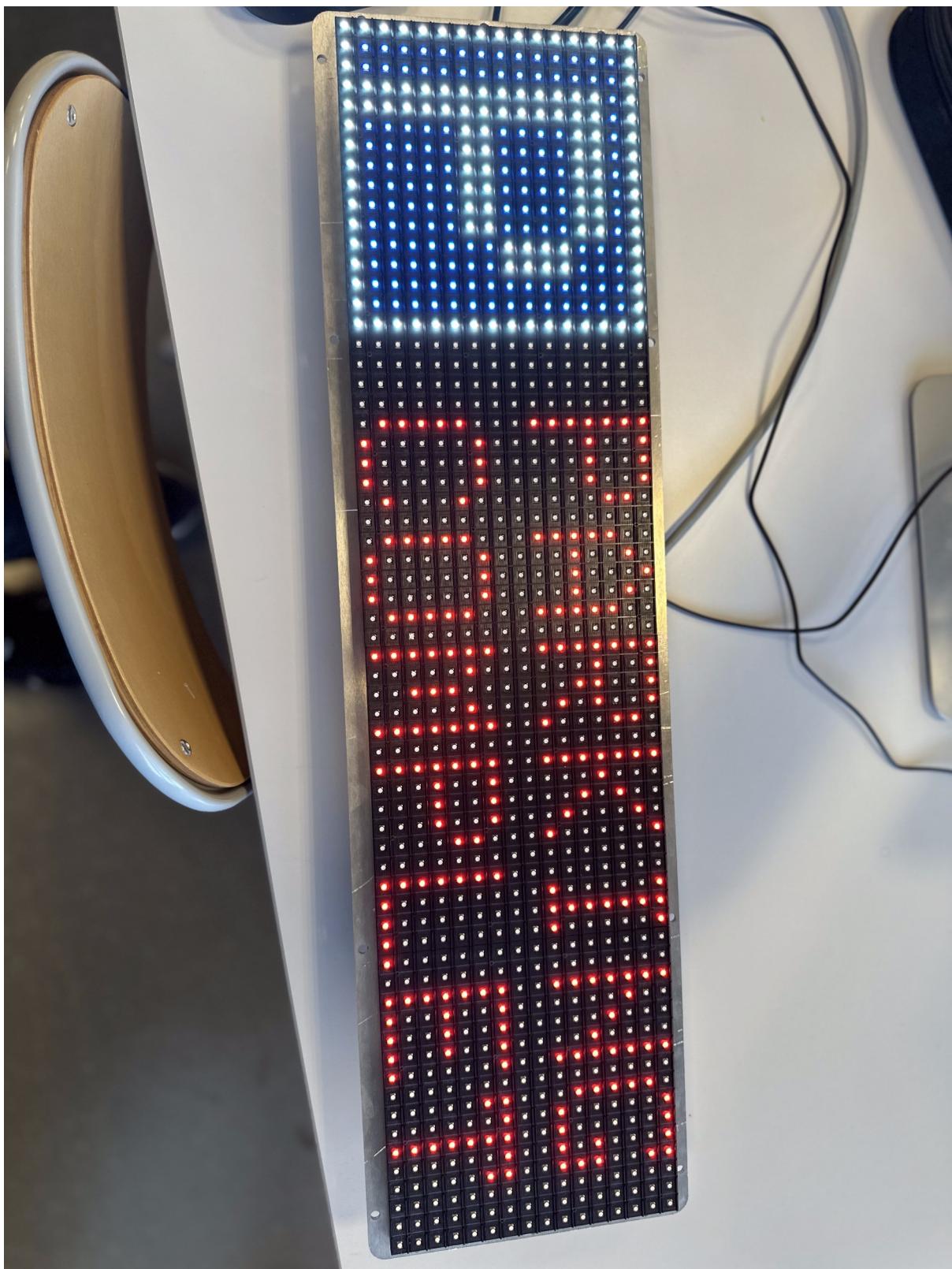
Il me manque la partie Expériences et mesures , j'espère pouvoir le proposer pour la prestation devant le jury .

Annexe :





uiy



ef



d

