Résumé

Comment compter les personnes qui accèdent à une pièce, afin de l'afficher aux personnes dans la pièce, mais aussi pour tenir des statistiques sur un plus long terme ?

Pour cela le projet a été décomposé en deux axes majeurs :

- L'affichage en temps réel pour les personnes présentes
- La sauvegarde du compteur à intervalle régulier pour faire des statistiques à plus long terme

Les deux parties font appel à un socle commun : la lecture des capteurs et l'interprétation de leurs activations.

Table des matières

Compte personnes

1 Introduction

1.1 Contexte

Dans le cadre de l'UE CMI3, nous avons dû choisir un projet à réaliser :
nous avons ainsi choisi le projet compte personne, qui correspond à trouver une solution pour connaître le nombre de personnes présentes dans la pièce. Nous souhaitons de plus pouvoir effectuer des statistiques de présence dans chaque salle afin qu'un étudiant puisse savoir en temps réel si une salle sera remplie ou non.
Un enjeu sécuritaire peut aussi être considéré. En effet, dans le cas d'une évacuation du bâtiment par exemple, un tel dispositif permettrait de savoir si tout le monde est bien sorti de chaque salle.

1.2 Problématique

Comment obtenir des statistiques en direct sur la fréquentation des salles de cours ?

2 Solutions

2.1 Solutions existantes

- Lecture du sens de passage sur une ligne sur un flux vidéo utilisé et développé par le cabinet "Occurrence". Dispositif utilisé pour compter le nombre de manifestants lors des récentes mobilisations
- Déclenchement de 2 capteurs successifs qui permettent la lecture d'un sens de passage

2.2 Solution développée

À l'aide de deux capteurs distants de quelques centimètres, et en lisant le sens d'activation, on peut déduire le sens de passage de la personne, et donc estimer le nombre de personnes dans la salle

3 Organisation

3.1 L'équipe de projet

L'intérêt de Colin pour l'électronique et la petite expérience de Quentin en développement web à permit de s'associer de la meilleure des manières. Nous avons tout de même dû faire appel à des aides humaines extérieures :

- Mobilisation de Mr.SILANUS autour de la transmission des données en bluetooth entre l'arduino et le programme JS interprétant les données.
- Mobilisation de Mr.GOZLAN autour :
    - de l'électronique : explications du fonctionnement des capteurs et du circuit

- des modules XBEE afin de transmettre les informations envoyées par les capteurs au arduino, du fonctionnement des modules et de leur transmission via la fréquence 2.4 GHz

Afin de communiquer au sein du groupe nous avons utilisé :

- Discussion Discord
- Github partagé : https://github.com/ClueXIII/compte_personne

3.2 Répartition des tâches

- Capteurs et support : partie fournie

Tâche	**Besoin**	**Elève responsable**
Envoi données brutes des capteurs via XBEE	Apprentissage de l'utilisation et de la configuration des modules XBEE à l'aide du logiciel XCTU	Colin
Interprétation des changements d'état des capteurs sur Arduino	Apprentissage du fonctionnement du système des interruptions sur le module ARDUINO UNO Utilisation d'Arduino IDE et programmation spécifique à Arduino	Colin
Affichage matériel, via matrice LED I2C	Apprentissage de l'utilisation des modules I2C et spécifiquement des matrices LED SEEED + Mise en parallèle de 2 écrans	Colin
Envoie et réception du nombre de personne du module arduino vers un script js en bluetooth	Apprentissage fonctionnement du bluetooth et du framework arduino SoftwareSerial	Quentin
Stockage dans un base de données sqlite des informations	Formation Sqlite (database Javascript)	Quentin

Création d'une api envoyant les informations de la base de donnée à partir d'une requête	Apprentissage fonctionnement api en Javascript	Quentin
Réception de ses données et élaboration de statistiques au moyen du framework chart js	Formation Chart.js	Quentin

3.3 Environnement matériel/logiciel

- Travail sur Arduino IDE/Visual Studio Code pour la partie Arduino
- Utilisation du logiciel Coolterm afin de paramétrer le module Bluetooth
- Utilisation de l'IDE Webstorm de Jetbrain afin de coder en Javascript
- Utilisation du logiciel XCTU pour configurer les modules XBEE
- Utilisation de l'outil GitHub et github desktop pour centraliser le code tout au long du projet

3.4 Planification

4 Réalisation

4.1 Travail d'équipe

Le travail s'est découpé entre nous selon les compétences de chacun, Quentin s'est orienté vers la partie WEB/Sauvegarde qu'il pratiquait déjà dans des projets personnels, alors que Colin s'est plus orienté vers la partie électronique, qu'il ne maîtrisait aucunement mais qui l'intéressait énormément. Nous avons donc travaillé chacun de notre côté puis nous nous sommes servi des séances en présentiel pour mettre en commun nos

avancements majeurs et voir les modifications que chacun devra apporter à sa partie pour conserver la possibilité de fusion.

La mise en service du système de comptage est assez simple, il suffit de mettre le circuit imprimé sous tension, puis de brancher l'Arduino. De cette manière, le compte pourra avoir lieu, avec l'affichage sur les écrans associés à l'Arduino.

En plus de ce système, on pourra lancer le système de transmission Bluetooth afin de sauvegarder en database et d'afficher sous forme de graphique le compteur au fil du temps.

4.2 Partie de Colin

Après des premiers essais avec XBEE, le système paraissait relativement bien fonctionner, mais le manque de code derrière rendait les tests inutiles, j'ai donc décidé de remettre la partie XBEE à plus tard, pour me concentrer sur la partie après le XBEE.

La première partie du travail était de réfléchir à comment reconnaitre le sens de passage de la personne, l'idée est d'utiliser un système de flag qui permet de savoir quel capteur a été déclenché :

On passe sous le capteur, ce qui active le flag

Tant que le second capteur n'est pas déclenché, le flag reste levé

Lorsque le second capteur s'active, l'action associée est effectuée et le drapeau est désactivé

Pour compenser les risques dits de "rebonds", c'est-à-dire des activations multiples très peu espacées dans le temps, j'ai ajouté une latence sur l'activation du système, ces "rebonds" ont dû d'autant plus être compensés que pendant les tests, des boutons-poussoirs ont été utilisés, ceux-ci génèrent beaucoups de rebonds.

```
if(flag2){ // Le capteur côté intérieur de la pièce a été déclenché plus tôt
  compteur--; // La personne est bien sortie
  flag2=false; // On est plus en attente de l'activation du capteur
  captLastTrigger=millis(); // Sauvegarde du temps
}
else if(!flag1&&(millis()>(captLastTrigger+captLatency))){ // Ce capteur n'a pas encore été déclenché ET la latence est respectée
  flag1=true; // En attente de l'activation du second capteur, le premier reste en attente
  captLastTrigger=millis(); // Sauvegarde du temps
}
```

Pour la latence, une solution simple dans le module Arduino est la fonction delay(millisecondes), qui permet de mettre en arrêt le système pour ces x millisecondes. Le problème de cette solution est qu'elle est bloquante : tout le système est mis en pause, empêchant également le déclenchement des interruptions et l'actualisation des écrans. Alors j'ai opté pour un système assez simple : On sauvegarde la temporalité de la dernière exécution, puis on teste à chaque fois que le délai défini plus tôt est bien respecté.

Partie affichage Matrice LED

Après lecture de la documentation pour l'utilisation des modules Matrice LED, l'application du code est assez simple

```
Wire.begin();
waitForMatrixReady();
uint16_t VID=0, VID2=0;
VID = ecran.getDeviceVID();
if (VID != 0x2886){
  Serial.println("Can not detect led matrix !!!");
}
Serial.println("Matrix init success!!!");
```

Initialisation de la communication

```
// Affichage sur les matrices LED
 if((lastScreenUpdate+screenLatency)<millis()){
   ecran.displayNumber(compteur, screenLatency, true, DISPLAY_COLOR); // On affiche la nouvelle valeur
   lastScreenUpdate = millis(); // On sauvegarde le temps
   Serial.println(compteur);
 }
```

Affichage et actualisation de la valeur tous les x millisecondes

Un problème majeur se pose : ce modèle de matrice LED n'a qu'une seule adresse I2C non modifiable sur le module.

I2C Address	0x65

La documentation ne donne pas une gamme d'adresses possibles

Après des recherches en ligne, aucune solution n'est trouvée, car les solutions ne s'appliquent pas à ce module.

Pour trouver de nouvelles idées, je me suis tourné vers ChatGPT, le Chatbot d'OpenAI, pour au minimum orienter mes recherches en ligne

- "Il est possible de modifier l'adresse I2C d'un module à l'aide de certains modules qui disposent d'un mécanisme"

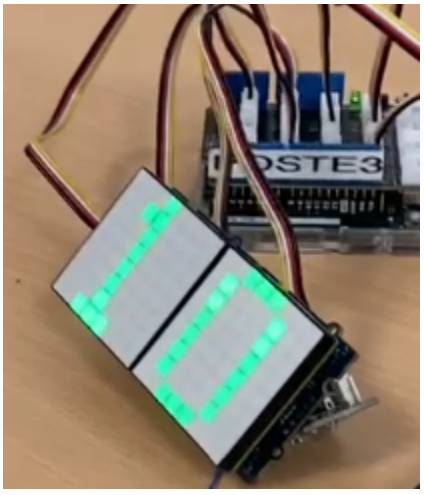
Après recherches sur internet, le module utilisé n'a pas de mécanisme de modification

Par contre cette réponse m'a donné l'idée de modifier "virtuellement" l'adresse du module, j'ai donc lu les fonctionnalités (méthodes) de la librairie grove_two_rgb_led_matrix

```
// /**********************************************************
// * Description
// *    Change i2c base address of device.
// * Parameter
// *    newAddress: 0x10-0x70, The new i2c base address of device.
// * Return
// *    Null.
// **********************************************************/
void changeDeviceBaseAddress(uint8_t newAddress);
```

Ainsi, l'adresse du module affichant l'unité est maintenant 0x66, celle de l'autre module reste 0x65. Alors en jouant sur les opérations sur les entiers en C++, les deux écrans affiche le nombre décomposé en unité et dizaine :

```
if((lastScreenUpdate+screenLatency)<lastMillis){
  screen1.displayNumber(compteur/10, screenLatency, true, DISPLAY_COLOR); // On affiche l'unité du compteur (propriété de la division des int)
  screen2.displayNumber(compteur%10, screenLatency, true, DISPLAY_COLOR); // On affiche la nouvelle valeur
  lastScreenUpdate = millis(); // On sauvegarde le temps
```

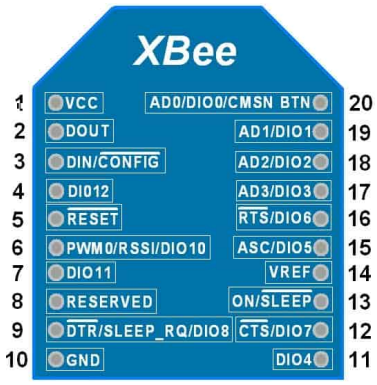


Maintenant que le code permet une bonne lecture des données reçues, il faut recevoir les données depuis le deuxième module.

Transmission via XBEE

Après analyse du circuit imprimé avec la disposition des pins fournis, les pins utilisés par les capteurs sont les pins DIO1 et DIO2, ainsi du côté récepteur et dans la configuration, j'ai donc activé la transmission des pins 1 et 2

Le système semble fonctionner correctement, malgré quelques imperfections

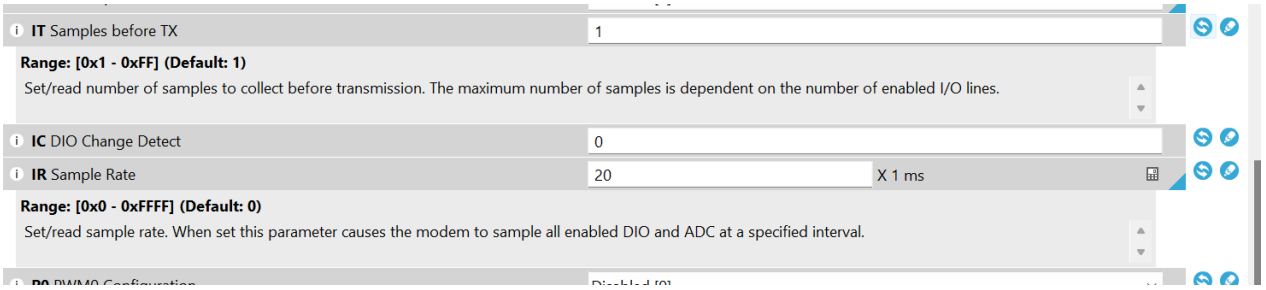


Lors des tests de la totalité du système (de la captation à l'interface web), j'ai remarqué que les détections semblent mal fonctionner, alors j'ai ajouté deux LED sur l'Arduino, retranscrivant les valeurs du capteurs, que j'ai pu comparer avec les deux LED présentes sur le circuit imprimé

```
digitalWrite(FLAG1, digitalRead(CAPT1));
digitalWrite(FLAG2, digitalRead(CAPT2));
```

J'ai ainsi remarqué que certains passages rapides n'étaient pas retranscrits sur l'Arduino, ou d'autres sont retransmis avec de la latence, créant ainsi des passages lu comme dans le sens inverse que celui qui a réellement eu lieu.

N'ayant aucune idée de comment régler ce problème, et la recherche en ligne n'ayant pas vraiment été fructueuse, avant d'avoir recours à ChatGPT, j'ai essayé de lire l'ensemble des options proposée à la configuration dans le logiciel XCTU :



Les valeurs de Sample Rate et Sample before TX étaient élevées, respectivement 14 et 5. Pour résoudre ces latences, j'ai donc mis les valeurs au minimum possibles (0 et 1). Ceci a permis de résoudre le problème.

4.3 Partie de Quentin

Il est important de notifier que le code étant très complexe dû aux spécifications des frameworks et aux diverses exceptions que le code doit gérer, l'entièreté du code ne peut pas être présentée ici. J'ai ainsi mis de nombreux commentaires sur les screens du code afin de faciliter la compréhension.

Etape 1 : Mettre en place le bluetooth

- Une fois le nombre de personnes dans la salle défini par le arduino, il faut envoyer les données en bluetooth. Pour cela j'utilise le frameWork arduino “SoftwareSerial”. Voici la partie bluetooth isolée du reste du programme arduino.
A noter la présence de la fonction delay dans ce programme qui va être remplacée par une méthode qui ne bloque pas l’exécution des autres fonctionnalités, comme pour l’actualisation des matrices LED I2C (voir “Affichage et actualisation de la valeur tous les x millisecondes” dans la partie 4.2

```
#include <SoftwareSerial.h>

// Pin utilisée pour la communication Bluetooth
#define RX_PIN 4
#define TX_PIN 5

// Initialise le port série Bluetooth
SoftwareSerial bluetooth(RX_PIN, TX_PIN);

// variable contenant le nombre de personnes
byte personnes = 10;

void setup() {
    // Initialise la communication série à 9600 bauds
    Serial.begin(38400);
    // Initialise la communication Bluetooth à 38400 bauds
    bluetooth.begin(38400);
}

void loop() {
    // Envoie le nombre de personnes en Bluetooth
    bluetooth.print(personnes);
    delay(10000);
}
```

- Il faut ensuite réceptionner ces données avec le programme javascript qui, lui, utilise le framework “bluetooth-serial-port” Ce dernier fonctionne avec Node.js

```
const btSerial = new (require('bluetooth-serial-port')).BluetoothSerialPort();

// Adresse MAC de l'Arduino Bluetooth
const address = '20:13:08:28:12:42'; // AT+ADDR? => 2013:8:281242

// Connecte à l'Arduino Bluetooth
btSerial.connect(address, channel: 1, successCallback: async function() {
  console.log('Connected to Arduino Bluetooth.');

  // Récupère la valeur envoyée par l'Arduino
  btSerial.on( eventName: 'data', listener: async function(buffer) {
    // Ferme la connexion Bluetooth
    btSerial.close();
    const data = await parseInt(buffer, radix: 10); // Convertit le buffer en chaîne de caractères
    console.log('Received data: ' + data); // Affiche la valeur reçue

  });
}, errorCallback: function() {
  console.log('Failed to connect to Arduino Bluetooth.');
  // Ferme la connexion Bluetooth
  btSerial.close();
});
```

Difficultées rencontrées :

La principale problématique rencontrée lors de la mise en place du bluetooth à été le paramétrage du module bluetooth au moyen du logiciel Cooltherm.

Dans un premier temps j'ai travaillé sur le "Module Grove Bluetooth BLE 113020007" qui est très peu documenté et que je n'ai ainsi pas réussi à faire fonctionner. Cependant cela n'a pas été du temps perdu étant donné que cela m'a permis de me familiariser avec le fonctionnement du bluetooth.

J'ai par la suite travaillé sur le "TEL0026 DF-BluetoothV3 Bluetooth module" qui, lui, était beaucoup mieux documenté. J'ai pu ainsi le connecter facilement à mon pc et le paramétrer au moyen de commandes entrée dans l'invite de commande de coolterm telles que :

- at+name=CP (permet de définir le nom du module bluetooth)
- at+uart=38400,0,0 (permet de définir le baudrate sur 38400 (bits/s), le nombre de stop-bits à 1 bit et le parity bit à None)

Le baudrate est très important car il définit le nombre de bit par seconde de la transmission. Ainsi si le baudrate de l'émetteur et du récepteur ne sont pas accordées les données reçues ne seront pas celles souhaitées. Le baudrate choisi a été 38400

Étape 2 : Stockage en database Sqlite:

Sqlite est un framework javascript qui stocke ses données dans un fichier .sqlite dans le projet. Le choix d'utiliser sqlite est né de la volonté de faire l'entièreté du projet en js. Ce framework permet de gérer seulement une faible quantité de données de manière optimisée ce qui nous convient énormément car nous n'avons pas énormément de données à stocker.

Voici l'organisation de notre database :

- une table comprenant les salles enregistrées et les caractéristiques (nom, adresse mac de leur module bluetooth, couleur sur le graphique et capacité maximale de la salle).

id	salle	adrs_mac	couleur	capacite
1	C0_21	20:13:08:28:12:42	#4349E2	30

Voici le code permettant de créer cette table :

```
db.run( sql 'CREATE TABLE IF NOT EXISTS salle_api(id INTEGER PRIMARY KEY AUTOINCREMENT,salle VARCHAR, adrs_mac VARCHAR, couleur VARCHAR, capacite INTEGER)')
db.all( sql 'SELECT * FROM salle_api', callback async (err :Error|null , data :unknown[] ) => { // on récupère les données de la table
```

A noter qu'il est possible d'ajouter n'importe quelle salle à partir de la fonction add_salle_api (ici nous travaillons avec une seule salle étant donné que nous n'avons qu'un seul support). Voici un exemple d'exécution :

```
add_salle_api( salle: "C0_21", adrs_mac: "20:13:08:28:12:42", couleur: "#4349E2", capacite: 30)
```

- D'autres sous tables de la première pour chaque salle qui enregistrent toutes les 10 secondes le nombre actuel de personnes envoyé par le module bluetooth :

id	hour	nbr_personnes
1	20:31	16
2	20:31	16
3	20:31	16
4	20:31	16

Voici le code de la création, de la table :

```
db.run( sql 'INSERT INTO salle_api(salle,adrs_mac,couleur,capacite) VALUES(?,?,?,?)', params: [salle,adrs_mac,couleur,capacite]); /
db.run( sql 'CREATE TABLE ' + salle + '_nbr_personnes(id INTEGER PRIMARY KEY AUTOINCREMENT,hour VARCHAR, nbr_personnes INTEGER)')
db.all( sql 'SELECT * FROM salle_api', callback async (err :Error|null , data :unknown[] ) => {
```

Et le code d'ajout de la données reçue en bluetooth (contenu dans bldata) :

```
db.run( sql 'INSERT INTO ' + salle + '_nbr_personnes(hour,nbr_personnes) VALUES(?,?)', params: [getDateFormatted(), bldata]);
db.all( sql 'SELECT * FROM ' + salle + '_nbr_personnes', callback (err :Error|null , data :unknown[] ) => { // on récupère les donné
```

Difficultées rencontrées :

Maîtrisant déjà le framework sqlite, le problème se trouvait principalement au nouveau des données reçues en bluetooth. En effet, par exemple si j'envoie 9 et que je laisse le programme tourner sans réceptionner les données, je vais recevoir "99999…." Ainsi, c'est ce genre d'exception que j'ai dû gérer et qui m'a posé problème.

Etape 3 : Création de l'api :

Afin d'obtenir les graphiques il faut récupérer les données de la database, ce qui n'est pas possible dans un programme chart.js. Ainsi l'on souhaite avec une simple requête pouvoir obtenir les données souhaitées. L'utilisation d'une api dans notre projet est donc incontournable. En effet, cette dernière permet à deux programmes informatiques différents de communiquer entre eux, en échangeant des données en utilisant le format json. J'ai donc dû mettre en place cette dernière (en local). Deux frameworks différents sont nécessaires : express et cors.

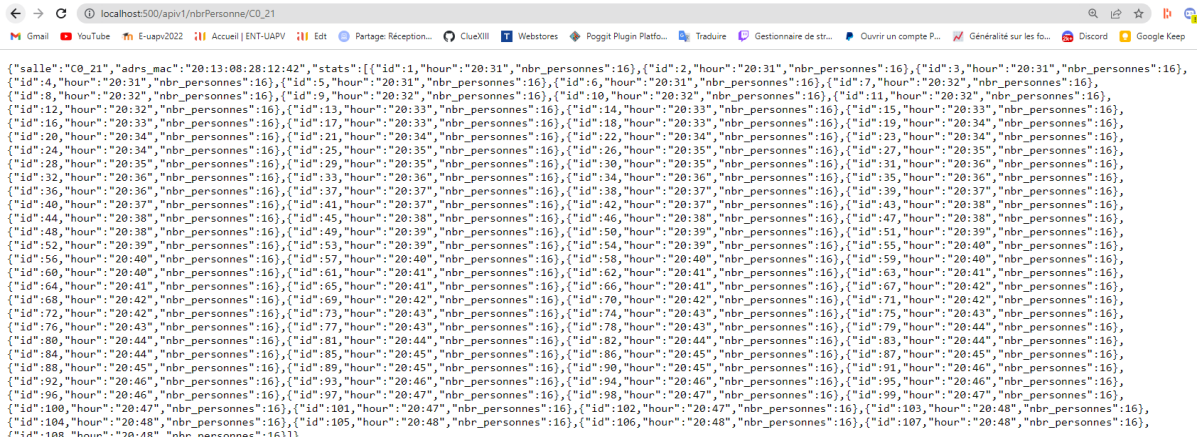
Voici la fonction set_api :

```
async function set_api(){
  app.use(cors()) // permet de faire des requetes depuis n'importe quel site
  app.get(`/apiv1/nbrPersonne/:salle`,async (req : … ,res : Response<ResBody, LocalsObj> ) =>{ // on récupère les données de la salle mise en paramètre
    db.all( sql: 'SELECT * FROM salle_api', callback: async (err : Error | null , data : unknown[] ) => { // on récupère les données de la table => les salles
      validator = 0 // on initialise le validateur à 0
      for (let i = 0; i < data.length; i++) { // on parcourt les salles
        if (data[i].salle === req.params.salle) { // on compare les salles avec la salle demandée
          validator = 1 // si la salle existe on passe le validateur à 1
          db.all( sql: 'SELECT * FROM ' + req.params.salle + '_nbr_personnes', callback: (err : Error | null , data_ : unknown[] ) => { // on récupère les données de la salle
            res.json( body: { // on renvoie les données
              salle: req.params.salle, // on renvoie le nom de la salle
              adrs_mac: data[i].adrs_mac, //  on renvoie l'adresse mac du module bluetooth
              stats: data_ // on renvoie les données de la salle
            })
          })
        }
      }
      if (validator==0){
        res.json( body: {
          message: "Cette salle n'est pas encore en place dans notre api."
        })
      }
    })
  })
```

Dans cette dernière on définit les requêtes possibles et les données envoyées lorsque ces dernières sont effectuées.

Par exemple, une fois l'api lancée si je marque dans mon navigateur : "http://localhost:500/apiv1/nbrPersonne/C0_21"

Voici ce que j'obtiens :



Le programme chart.js peut ainsi totalement librement utiliser les données obtenues pour faire des statistiques sous forme de graphique.

Etape 4 : Création de graphiques

Le framework Chart js utilisant lui aussi node.js est l'outil de création de graphiques sur un site web le plus simple à prendre en main.

Dans un premier temps il faut créer dans le fichier html une balise canva avec un id spécifique qui permettra de récupérer ce dernier en js (en ayant bien sur importé précédemment le cdn de chart js et le lien du script qui crée le graphique dans le fichier html) :

```
<div><canvas class="graph" id="hours_affluence" aria-label="chart" role="img" style="..."></canvas></div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.7.1/chart.min.js"></script>
<script src="graph.js"></script>
```

Dans le fichier graph.js la fonction getdata permet d'effectuer une requête serveur afin de récupérer les données de l'api :

```
async function get_Data() {
  try {
    const response = await fetch( // on récupère les salles en utilisant l'api
      input: `http://localhost:500/apiv1/salle`
    );
    responses = await response.json()
    salle = responses.salle
    let result = []; // on initialise le tableau qui contiendra les données
    for (let i = 0; i < salle.length; i++) { // on parcourt les salles
      let salleinfo = await fetch(
        input: `http://localhost:500/apiv1/nbrPersonne/` + salle[i].salle // on récupère les données de la salle
      );
      data = await salleinfo.json()
      result.push({ // on ajoute les données de la salle dans le tableau
        label: salle[i].salle, // on ajoute le nom de la salle
        data: await data.stats.map(row => row.nbr_personnes), // on ajoute les données de la salle
        borderColor :'rgb(255,255,255)', // on ajoute la couleur de la courbe
        tension: 0.1, // on ajoute la tension de la courbe
      })
    }
    return result
  } catch (error) {
    console.error(error);
  }
}
```

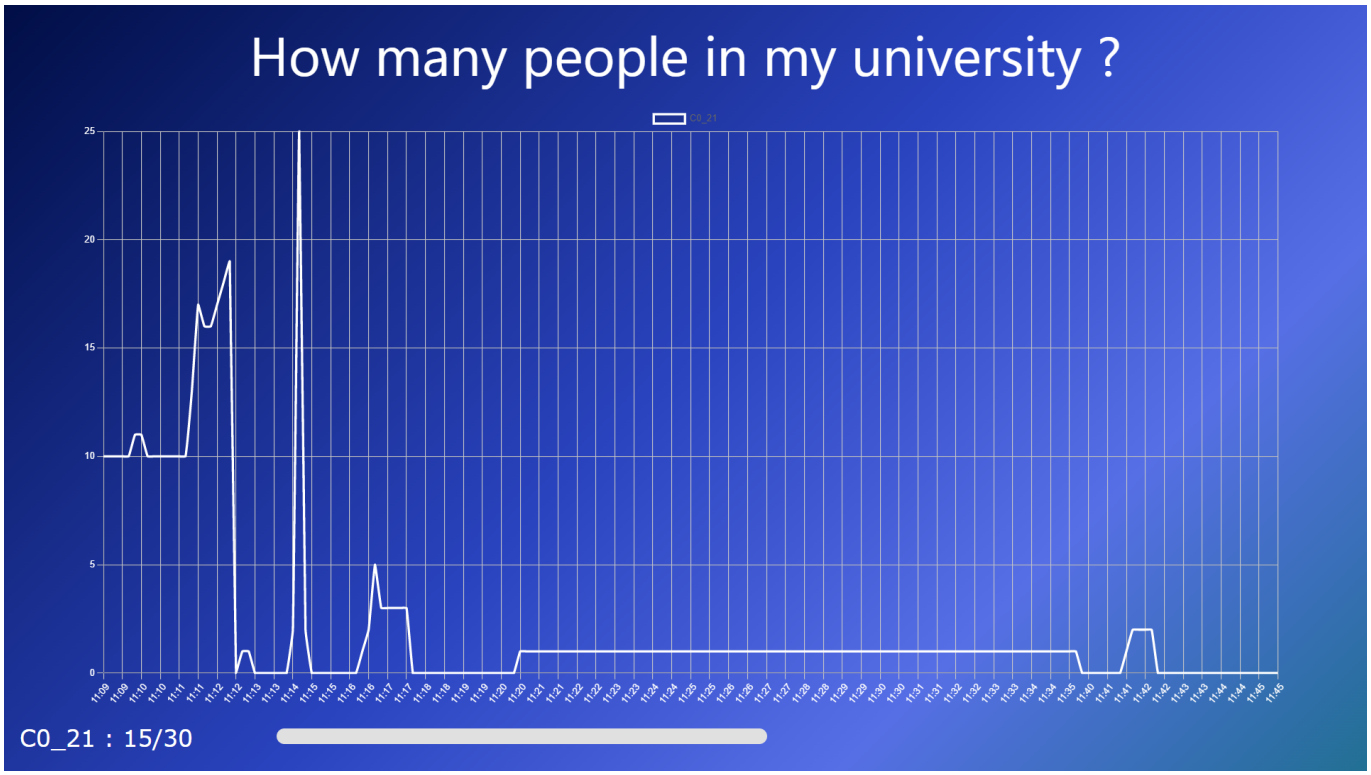
On classe toutes les données dans le tableau résultat sous le format chart.js et cela pour chaque salle transmise par l'api.

On utilise ensuite la fonction build chart qui va permettre en utilisant la fonction get_Data() de construire le graphique. Beaucoup de choses sont paramétrable dans ce dernier en passant même par la couleur du grillage de chaque axe :

```
scales: {
  y: {
    ticks: { color: 'rgb(255,255,255)'},
    beginAtZero: true,
    grid : {
      color : 'rgb(187,187,187)'
    }
  },
  x: {
    ticks: { color: 'rgb(255,255,255)'},
    grid : {
      color : 'rgb(199,199,199)',
```

Il est aussi possible de choisir un type de graphique différent en changeant l'attribut "type : line" en "type : bar" pour avoir un graphique en barres par exemple. Cependant le graphique de ligne est beaucoup plus parlant et lisible avec beaucoup de données qu'un graphique en barres.

Etape 5 : Mise en page du site



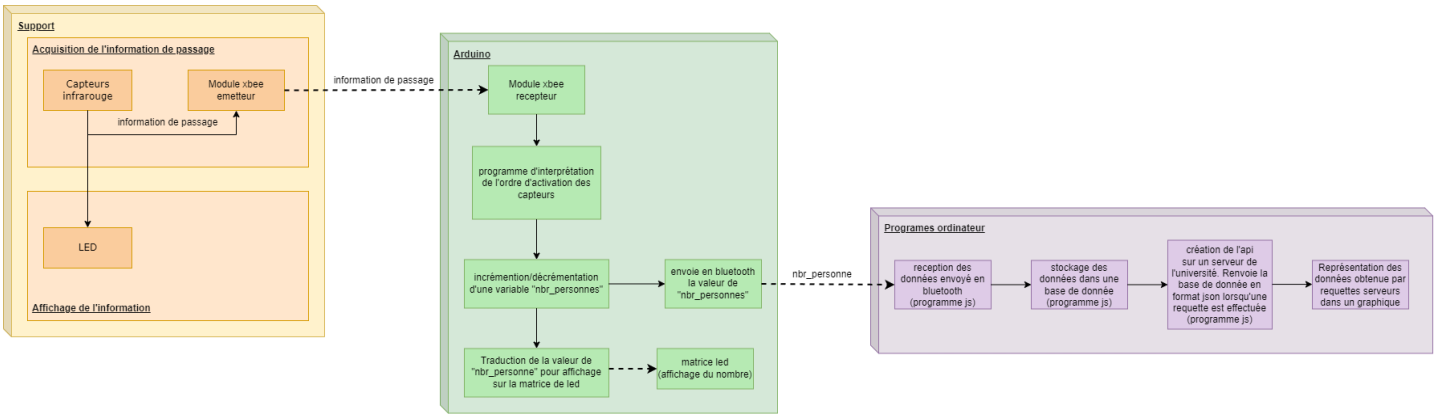
La mise en page n'étant pas complexe, quelles lignes de css sont suffisantes.
La petite spécificitée que je peux souligner et la barre de progression en dessous du graphique :
Cette dernière fonctionne entièrement en js et s'adapte au nombre de personnes dans chaque salle. En effet, ici nous n'avons qu'une seule salle, mais si il y en à plusieurs, une autre ligne apparaîtra en dessous avec sa propre capacité, son propre nombre de personnes et sa propre barre de progression. Les données sont récupérées de la même manière que pour le graphique.
Le développement de cette barre de progression, quand à lui, est plutôt classique :

```
const progressBar = document.createElement( tagName: "div"); // créer une div pour la barre de progression
progressBar.className = "progressBar";
if (lastNbrPersonnes >= capa) { // si la salle est pleine
  progressBar.style.width = "100%"; // remplir la barre
} else {
  progressBar.style.width = (lastNbrPersonnes / capa) * 100 + "%"; // remplir la barre proportionnellement au nombre de personnes présentes
}
```

4.4 Intégration

L'intégration de nos deux parties s'est faite simplement au moyen du bluetooth entre l'arduino et les programme de l'ordinateur.
Lors d'une des dernières séances, Colin s'est rendu compte que les pins utilisés par les deux parties étaient les mêmes, après avoir fait des recherches, le module Arduino ne permet l'utilisation des interruptions que sur les pins 2 et 3, alors, Quentin a déplacé son travail sur les pins 4 et 5.

Voici le schéma récapitulatif de nos axes de travail.



5 Conclusion

Grâce à notre système de compte personnes, l'on peut ajouter autant de salles que l'on souhaite à la base de données à partir du moment où cette dernière est dotée d'un support et d'un arduino qui réceptionne et envoie les données en bluetooth (à noter que le module Arduino UNO ne permet pas d'avoir plus de deux interruptions, limitant ce modèle à une unique salle).

Par contre, l'ajout de multiples salles ne permettra pas d'avoir un affichage dans chacune des salles, à moins de revoir le système, car il faudrait alors faire un système qui permettrait de faire fonctionner le système I2C via XBEE, et faire l'affichage dans une zone proche des capteurs.

Ainsi, nous avons là une source de données incroyable qui peuvent être interprétées et utilisées de nombreuses manières. Le site que nous avons développé n'est qu'une simple démonstration des nombreuses possibilités que de telles données offrent. Si l'on étend la mise en place a un un établissement entier, l'on pourrait faire des prévision d'affluences ou encore dans le cas d'une évacuation, vérifier que plus personne n'est dans le bâtiment.

La découverte tant d'un point de vue global que très spécifique à certains outils nous a permis d'avoir un premier regard très intéressant sur le monde de l'ingénierie, qui reste encore tellement vaste que nous aimerions continuer de découvrir à travers d'autres projets.