

Projet Multidisciplinaire 2

Lego's colour

Présenté par

AOUFIA Samir
DELAERE Théo

Année Académique 2022-2023

Table des matières

1. Introduction.....	4
2. Project's follow-up	4
2.1 Sketch/ Mind Map.....	4
2.2 Open Project.....	5
3. Raspberry pi.....	5
4. Embedded system	6
2.1 Arduino Méga.....	6
4.2 Camera Huskylens	7
4.3 Écran LCD avec module I2C intégré	10
4.4 La breadboard et le bouton.....	11
5. Docker	12
6. Database.....	12
3.1 MYSQL	12
3.2 PhpMyAdmin.....	13
7. Node Red	14
5.1 Explication des nœuds	14
8. Web	15
6.1 NodeJS.....	15
6.2 Index.....	15
6.3 Dashboard	16
6.4 History	17
6.5 Route	18
6.5.1 Route « / ».....	18
6.5.2 Route « /data ».....	18
6.5.3 Route /api/data	21
6.5.4 Route /history	21
6.5.5 Fonction retrieveData	23
9. Conclusion	23
10. Appendices	23
7.1 Datasheets.....	23
10.2 Code	23
10.2.1 Web	23

10.2.2	Arduino.....	24
10.3	GANTT	28
11.	Webographie.....	28

1. Introduction

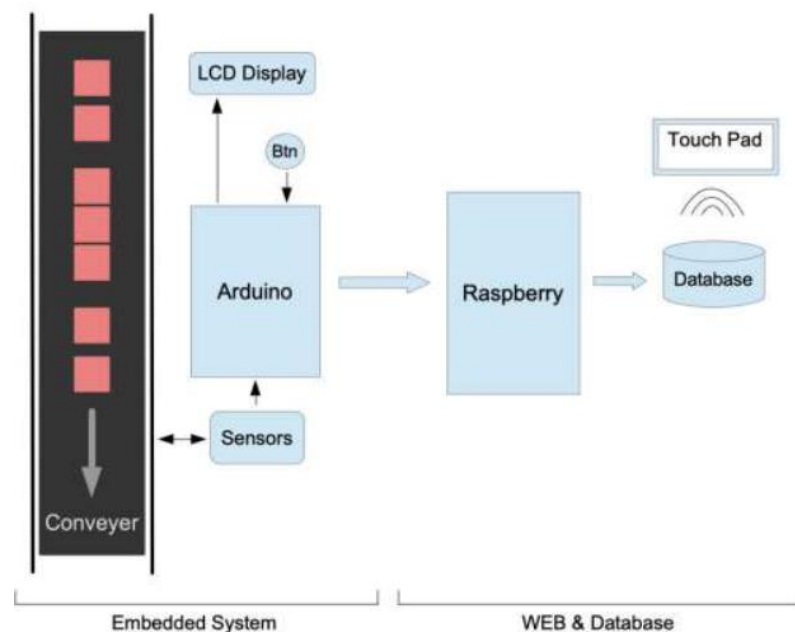
This year, we were asked to carry out a multidisciplinary project in three weeks, in pairs and with the material provided by the school.

The project involved the recognizing of different colours. Firstly, blocks of various colours pass on a conveyor belt. Then, a sensor needs to identify the colour and send it to a website where all the data will be recorded.

Important parts

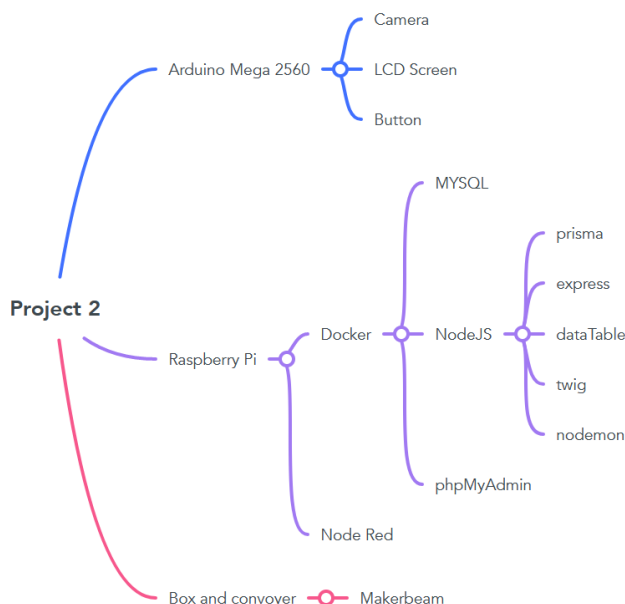
These are the major parts for the project:

- Raspberry Pi
- Embedded system
- Database
- Node Red
- Web



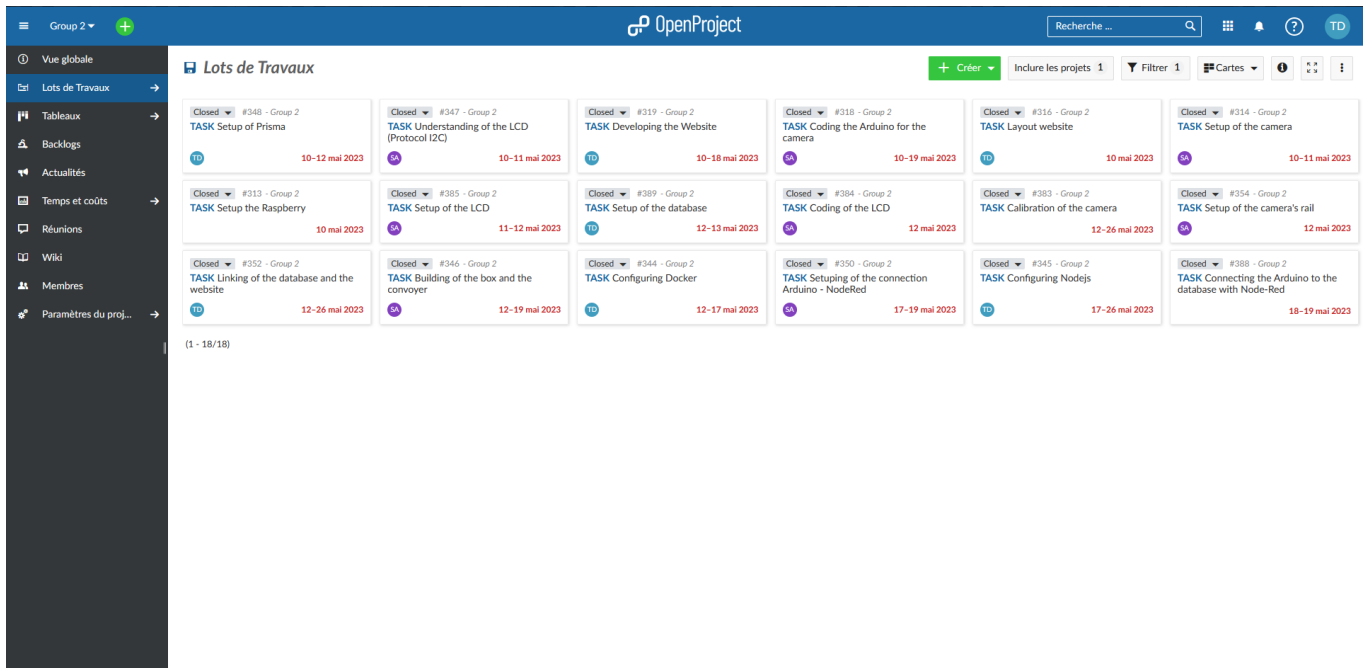
2. Project's follow-up

2.1 Sketch/ Mind Map



2.2 Open Project

Open Project is a project management software. It helped us staying organized and adhering to the deadlines we had set for the different tasks.



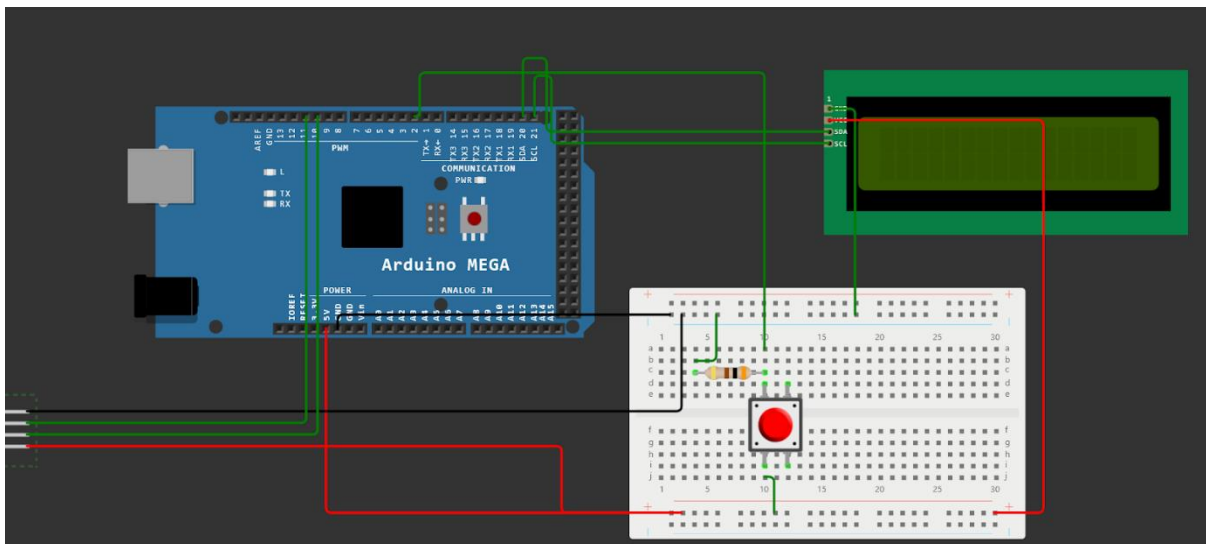
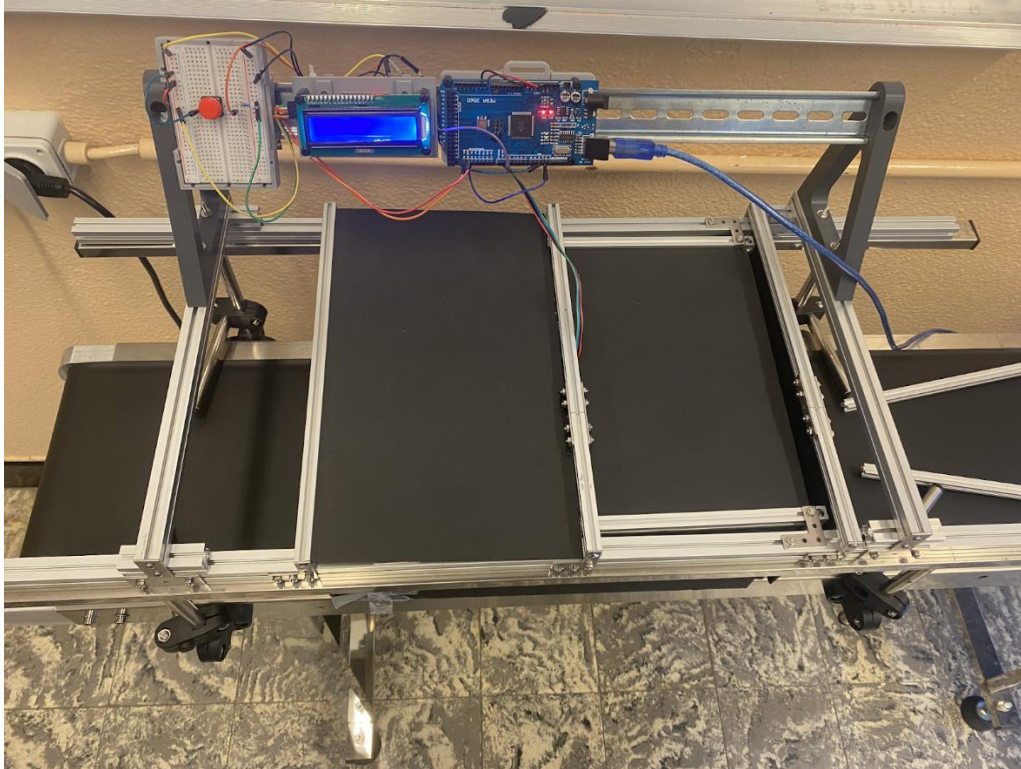
We mainly used the "Lots de travaux" page to complete the project successfully. All tasks are already in the "Appendices", at the end of this paper.

3. Raspberry pi

L'école nous a mis à disposition un Raspberry Pi. Il s'agit d'un nano-ordinateur à processeur ARM de la taille d'une carte de banque. Il nous a permis de développer et d'héberger la base de données, le site internet et le Node Red.



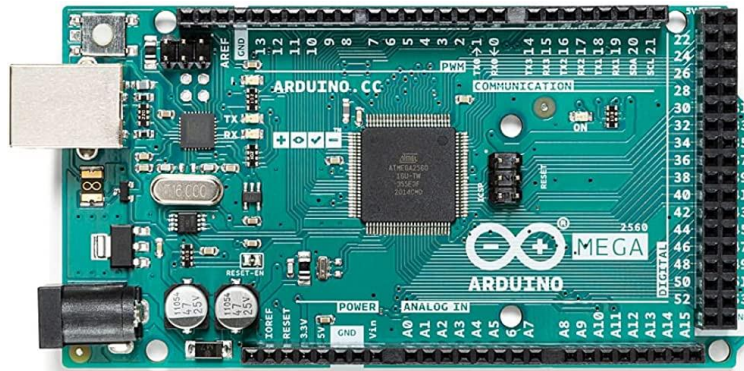
4. Embedded system



2.1 Arduino Méga

Nous avons utilisé un Arduino méga 2560 qui est assez similaire à l'Arduino uno que nous avons utilisé pour le cours de *Systèmes embarqués*. L'Arduino méga 2560 possède un microcontrôleur ATmega2560 avec une mémoire flash de 256 KB, ainsi que 54 entrées/sorties numériques, dont 14 sorties digitales (PWM), 16 entrées analogiques et 1

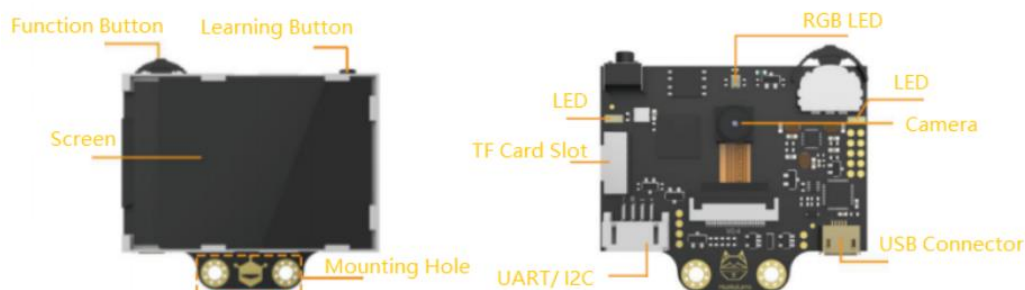
bouton reset. L'Arduino nous permet d'exécuter un code en boucle pour récupérer les informations de la caméra.



4.2 Camera Huskylens

Nous avons décidé de choisir la caméra Huskylens car c'est une caméra intelligente et cela nous a facilité la tâche. Nous aurions aussi pu choisir le capteur gy-31 tcs3200 cependant, il nous a été renseigné que le capteur était peu performant.

La caméra dispose de deux boutons, le premier est le learning bouton qui sert à apprendre les couleurs. Nous devons simplement placer une brique de lego d'une certaine couleur devant l'objectif de la caméra, vérifier qu'elle capte correctement la couleur souhaitée et ensuite, appuyer sur le learning bouton. Après cela, un décompte se lance. À la fin de ce décompte, la caméra commence la détection. Pour apprendre d'autres couleurs, nous devons réappuyer sur ce même bouton avant la fin du décompte.



La caméra détecte les couleurs grâce à des id préalablement enregistrer manuellement dans la caméra.

Pour l'utilisation nous avons dû inclure 2 bibliothèques : HUSKYLENS.h contient les fonctions pour communiquer avec la caméra, SoftwareSerial.h est utilisée pour établir une communication logicielle.

```
#include "HUSKYLENS.h"
#include "SoftwareSerial.h"
```


La première ligne crée une instance de classe HUSKYLENS nommée huskylens.

La deuxième ligne crée une instance de classe SoftwareSerial nommée mySerial avec les broches 10 et 11 utilisées pour la communication.

```
HUSKYLENS huskylens;
SoftwareSerial mySerial(10, 11);
```

Dans la partie setup, qui est exécutée seulement au démarrage ou au reset de l'arduino :

On démarre la communication sur le port matériel avec une vitesse de 115200 bauds et une communication logicielle avec une vitesse de 9600 bauds sur les broches 10 et 11.

Ensuite, on teste la communication avec la caméra avec une boucle while et s'il y a un problème, on affiche un message d'erreur.

```
Serial.begin(115200);
mySerial.begin(9600);

while (!huskylens.begin(mySerial))
{
    Serial.println(F("Begin failed!"));
    Serial.println(F("1.Please recheck the \"Protocol Type\" in
HUSKYLENS (General Settings>>Protocol Type>>Serial 9600)"));
    Serial.println(F("2.Please recheck the connection."));
    delay(100);
}
```

Dans la partie loop qui est exécutée en boucle :

On fait un test avec un IF si la caméra envoie bien des informations. Si ce n'est pas le cas, on affiche un message d'erreur, sinon, on continue le test avec un ELSE IF pour voir si ce qu'elle envoie a été appris. Si ce n'est pas le cas, on affiche un message d'erreur, sinon on rentre dans une boucle while qui lit les résultats envoyés par la caméra et pour chaque résultat on appelle la fonction printResult

```
if (!huskylens.request()) Serial.println(F("Fail to request data from
HUSKYLENS, recheck the connection!"));
else if (!huskylens.isLearned()) Serial.println(F("Nothing learned,
press learn button on HUSKYLENS to learn one!"));
else
{
    while (huskylens.available())
    {
        HUSKYLENSResult result = huskylens.read();
        printResult(result);
    }
}
```

La fonction printResult sert à afficher les résultats sur l'écran lcd et sur le Serial Moniteur :

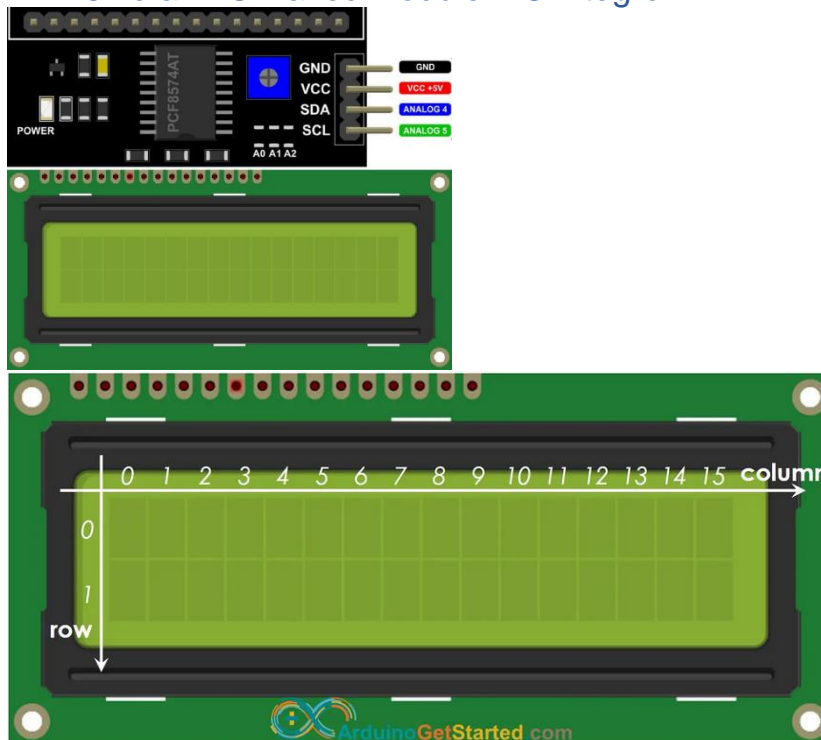
On commence par faire un test IF pour être sûr qu'il s'agit bien d'un bloc. Ensuite on fait un SWITCH pour qu'en fonction de l'ID du bloc, elle affiche la bonne couleur sur l'écran et le Serial Moniteur. Pour finir, on attend 2 secondes puis on efface l'écran.

```
void printResult(HUSKYLENSResult result){
    if (result.command == COMMAND_RETURN_BLOCK){
```



```
switch(result.ID) {  
  
    case 1:  
        lcd.clear();  
        lcd.print("Yellow");  
        Serial.println("Yellow");  
        delay(2000);  
        lcd.clear();  
        break;  
  
    case 2:  
        lcd.clear();  
        lcd.print("Blue");  
        Serial.println("Blue");  
        delay(2000);  
        lcd.clear();  
        break;  
  
    case 3:  
        lcd.clear();  
        lcd.print("Red");  
        Serial.println("Red");  
        delay(2000);  
        lcd.clear();  
        break;  
  
    case 4:  
        lcd.clear();  
        lcd.print("Orange");  
        Serial.println("Orange");  
        delay(2000);  
        lcd.clear();  
        break;  
  
    case 5:  
        lcd.clear();  
        lcd.print("Green");  
        Serial.println("Green");  
        delay(2000);  
        lcd.clear();  
        break;  
  
}
```

4.3 Écran LCD avec module I2C intégré



Nous avons utilisé un écran LCD (16 colonnes, 2 lignes) avec un module I2C. Le module I2C nous sert à simplifier le câblage à seulement 4 fils, 2 pour l'alimentation et 2 pour la communication I2C, SDA (data signal) et SCL (clock signal)

Nous utilisons le LCD pour afficher la couleur du bloc qui vient de passer pendant 2 secondes. Après cela l'écran s'efface tout seul.

Pour l'utilisation nous avons dû inclure 2 bibliothèques :
 LiquidCrystal_I2C.h est utilisé pour contrôler l'écran LCD, Wire.h permet la communication I2C.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Cette ligne crée une instance de la classe LiquidCrystal_I2C appelée LCD avec l'adresse 0x27, 16 colonnes et 4 lignes

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

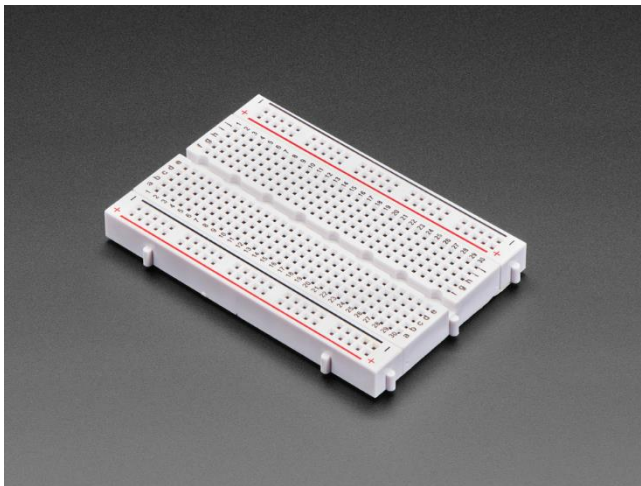
Dans la partie setup, on initialise le LCD, et on active le rétroéclairage

```
lcd.init();
lcd.backlight();
```

Voici les lignes que nous avons utilisées pendant notre programme. La première ligne sert à effacer l'écran LCD tandis que la seconde sert à afficher les éléments souhaités sur l'écran LCD.

```
lcd.clear();  
lcd.print("Blue");
```

4.4 La breadboard et le bouton



Nous avons utilisé une breadboard pour réaliser notre circuit électronique. Nous y avons branché un bouton qui est utilisé pour effacer l'écran du LCD. La breadboard nous a également servi d'extension d'alimentation avec le +5V et le GND.

Pour l'utilisation du bouton, nous avons commencé par déclarer une variable de type byte pour économiser de la place sur l'arduino et nous l'avons appelée btn. Dans btn, nous avons donné le numéro de la broche.

```
byte btn = 2;
```

On configure le mode de la pin btn en INPUT_PULLUP. Cela active une résistance de tirage interne à la carte arduino, ce qui signifie que lorsque le bouton n'est pas enfoncé, la broche est maintenue à un niveau logique haut grâce à la résistance interne.

```
pinMode(btn, INPUT_PULLUP);
```

La fonction attachInterrupt permet de faire une interruption dans le code à n'importe quel moment. L'option FALLING permet de faire l'interruption quand la broche passe de haut à bas. Citons enfin la fonction clear qui est la fonction à exécuter au moment de l'interruption.

```
attachInterrupt(digitalPinToInterrupt(btn), clear, FALLING);
```

Dans notre cas l'interruption fait effacer l'écran LCD.

```
void clear() {  
  lcd.clear();  
}
```

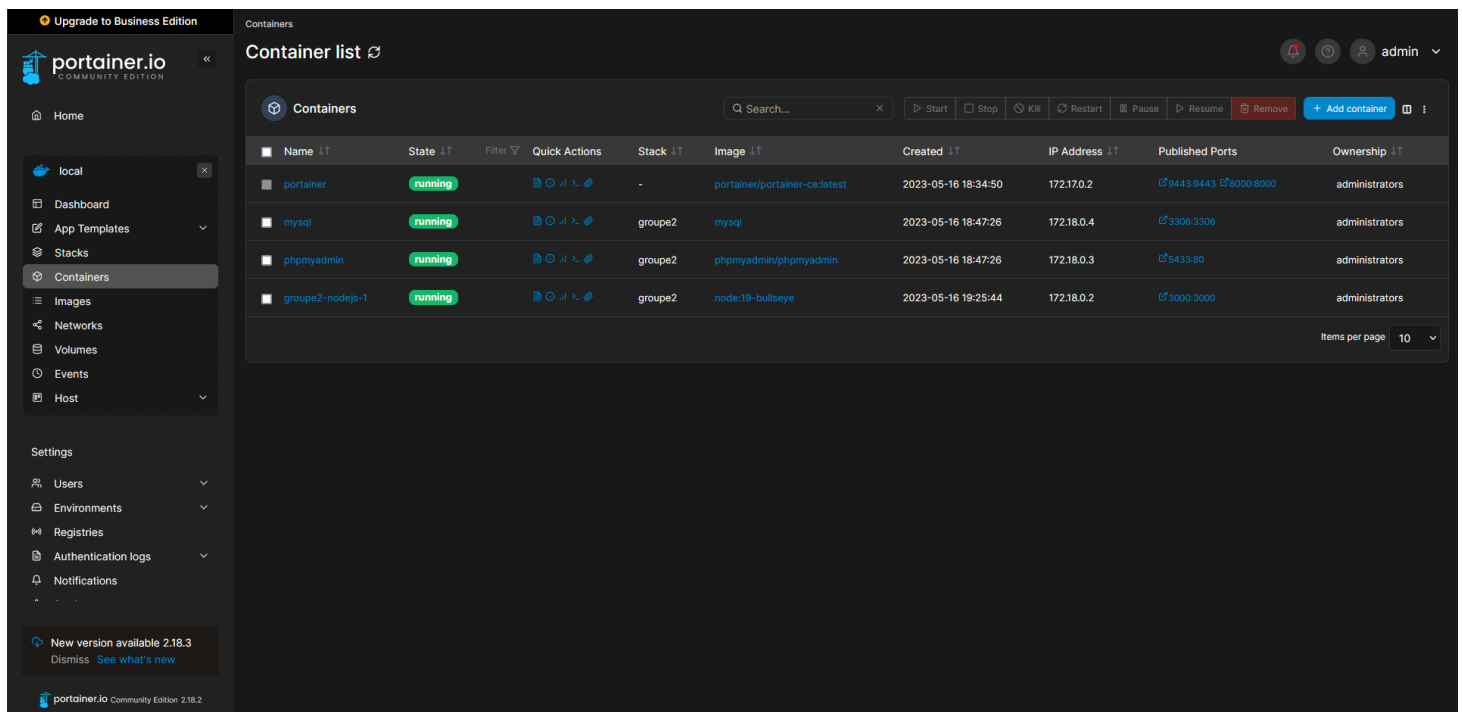
}

5. Docker

Nous avons utilisé Docker afin de lancer les différentes applications au démarrage du Raspberry Pi. Docker est un logiciel permettant de lancer des applications dans des conteneurs.

Ici nous avons décidé d'utiliser Portainer en complément de Docker.

Portainer est un service de container management. Il permet d'utiliser Docker de façon graphique et non uniquement en ligne de commande.



6. Database

3.1 MYSQL

Pour ce projet, nous avons utilisé MYSQL comme service de base de données. Ce service était hébergé sur le Raspberry Pi. Afin de configurer et d'administrer les différentes tables, nous avons en outre utilisé phpMyAdmin qui est une application web ayant pour but de gestionner des bases de données MYSQL et MariaDB.



3.2 PhpMyAdmin

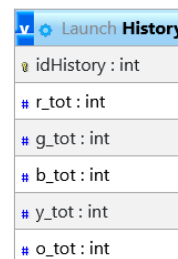
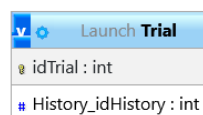
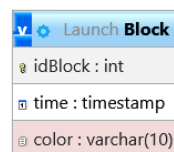
Pour réaliser ce projet, nous avons eu besoin d'une base de données comprenant 3 tables : une table « Block » comprenant tous les blocs scannés, une « Trial » comprenant la séquence en cours et une « History » détenant un historique de toutes les séquences enregistrées.



Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> Block	Parcourir Structure Rechercher Insérer Vider Supprimer	73	InnoDB	utf8mb3_general_ci	32,0 kio	-
<input type="checkbox"/> History	Parcourir Structure Rechercher Insérer Vider Supprimer	14	InnoDB	utf8mb3_general_ci	16,0 kio	-
<input type="checkbox"/> Trial	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb3_general_ci	32,0 kio	-
3 tables Somme		88	InnoDB	utf8mb3_general_ci	80,0 kio	0

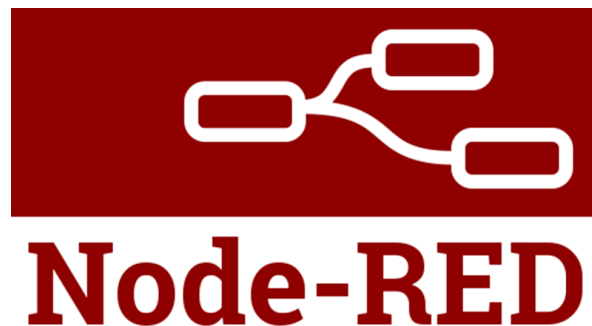
Voici le schéma EER de notre base de données

Nos tables n'ont pas de relations entre elles dans phpMyAdmin car celles-ci ont été faites via Node Red.

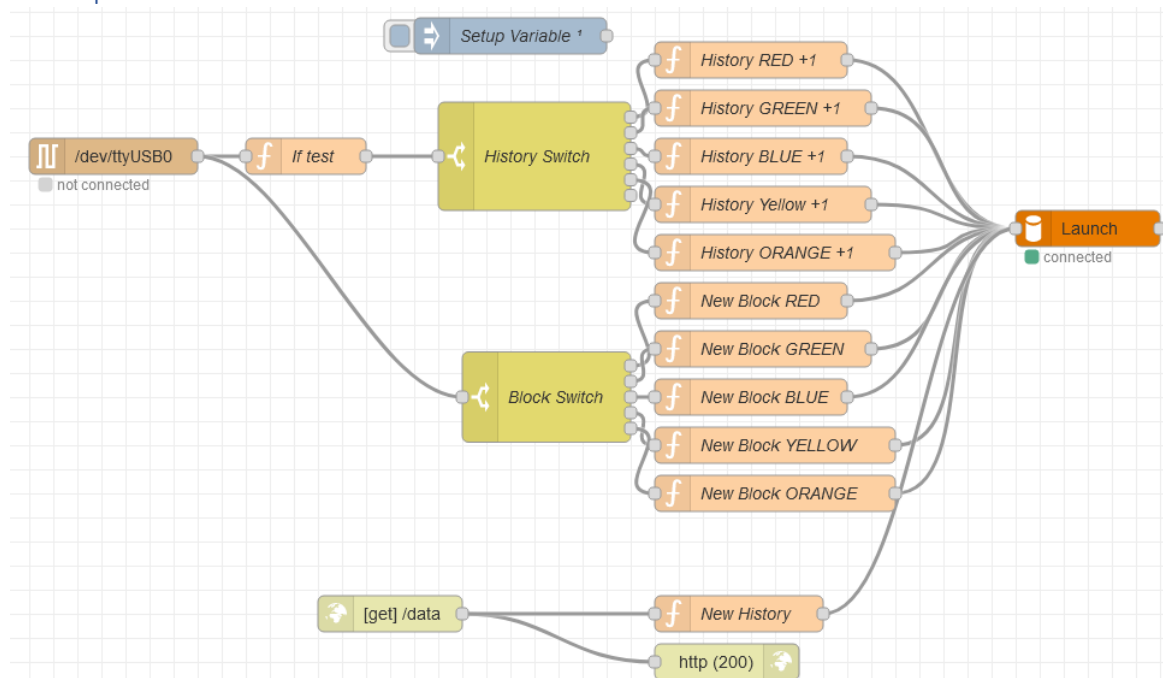


7. Node Red

Pour connecter l'Arduino Mega avec le site internet, nous avons fait appel à Node Red. Il s'agit d'un outil de programmation basé sur le visuel, créé à la base pour connecter des périphériques matériels, des API et des services en lignes ensembles.



5.1 Explication des nœuds



Dans l'ordre :

- La node /de/ttyUSB0 représente la connexion effectuée avec l'Arduino et donne directement la valeur reçue de celui-ci
- La node If test, sert de test pour savoir si une séquence a été enclenchée. Si oui, elle déclenche la node History Switch.
- La node History Switch sert de test afin de savoir quelle couleur a été scannée.
- La node Block switch sert aussi de test pour savoir quelle couleur a été scannée.
- Toutes les nodes History « COULEUR » +1 servent à transmettre une requête SQL à la base de données. Ici, la requête SQL augmente de 1 la valeur de la couleur scannée dans la table History.

- Toutes les nodes New Block « COULEUR » servent aussi à transmettre une requête SQL à la base de données. Cette requête-ci créera un nouveau block de la couleur scannée dans la table Block.
- La node [get]/data représente la connexion avec le site, elle récupère les informations des appuis de boutons sur le site.
- La node New History est un test pour savoir si le bouton appuyé est le bouton « Start » ou le bouton « Stop ». Si le bouton start a été pressé, une requête SQL est envoyée vers la base de données. La requête demande à la base de données de créer une nouvelle séquence dans la table History.
- la node Launch représente la base de données nommée « Launch ».

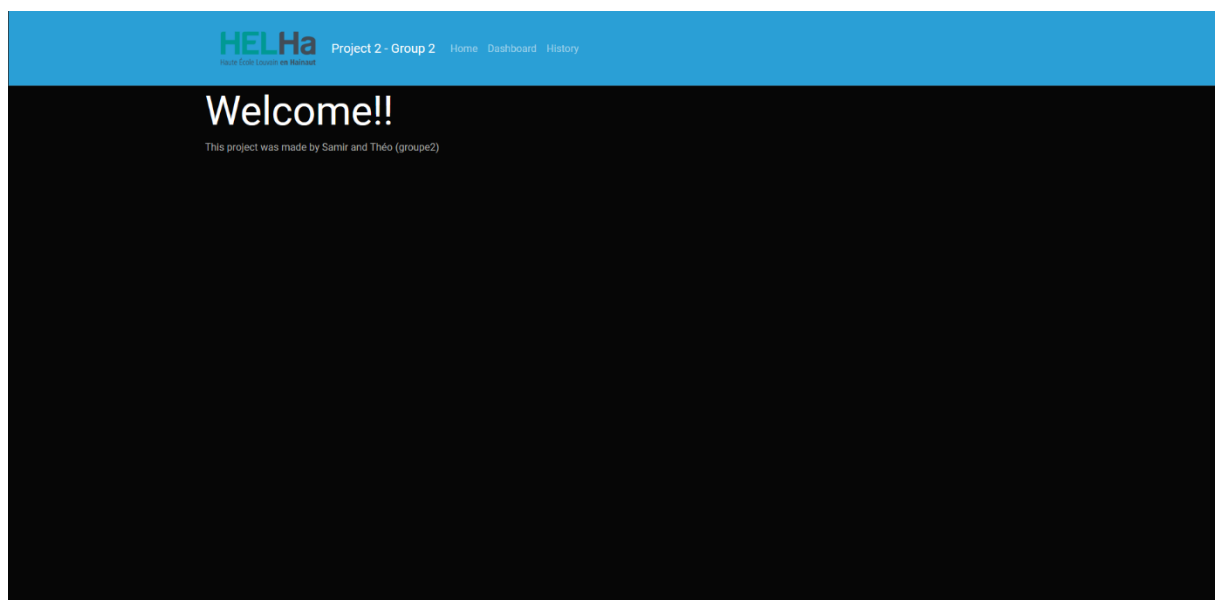
8. Web

Nous devons aussi concevoir un site contenant au minimum deux pages : une page dashboard, montrant en direct les valeurs de la séquence en cours, ainsi qu'une page History, contenant les valeurs de toutes les séquences précédentes. Le site a été développé avec NodeJS. Tout le code est disponible sur GitHub : <https://github.com/TheoDelaere/Project-2>

6.1 NodeJS

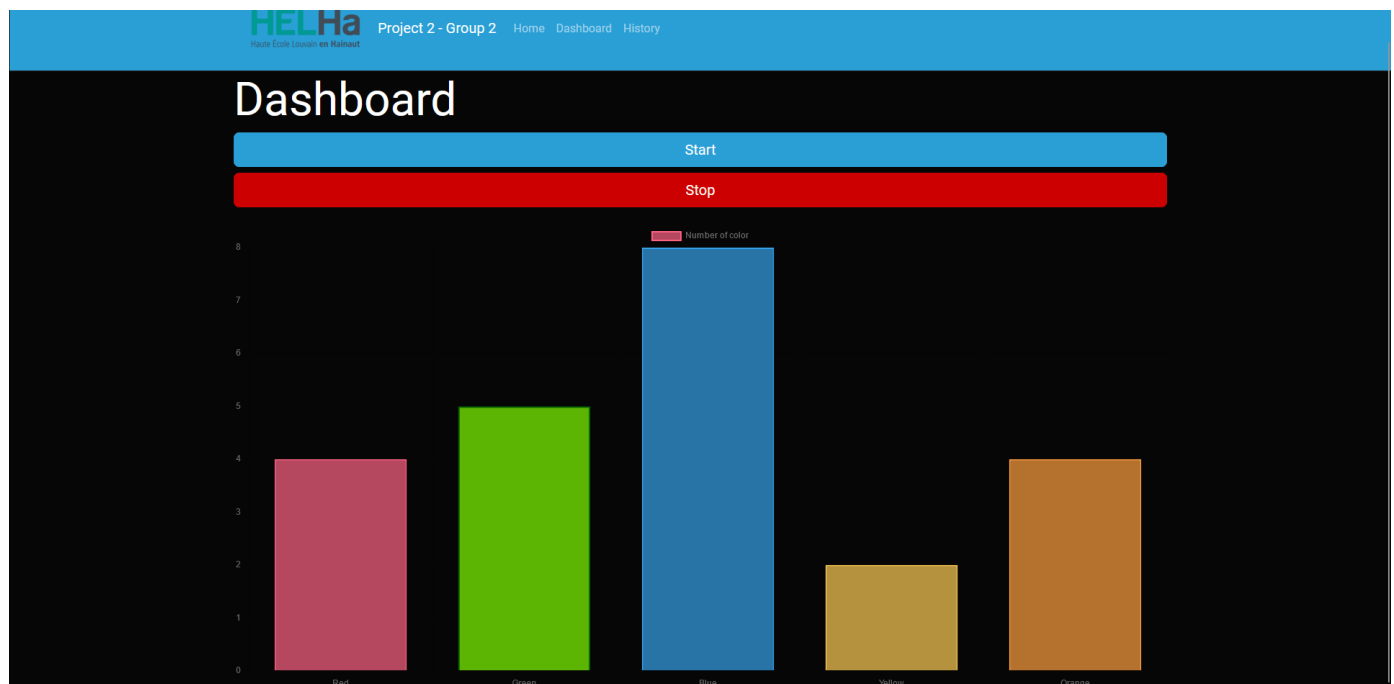
NodeJS est un environnement open-source qui nous a permis de développer l'application web demandée. Il permet d'exécuter du JavaScript coté serveur et aussi de travailler avec d'autres packages open-source. Nous avons entre-autres travaillé avec les packages suivants : express.js, express-generator, mysql, prisma, twig, nodemon.

6.2 Index

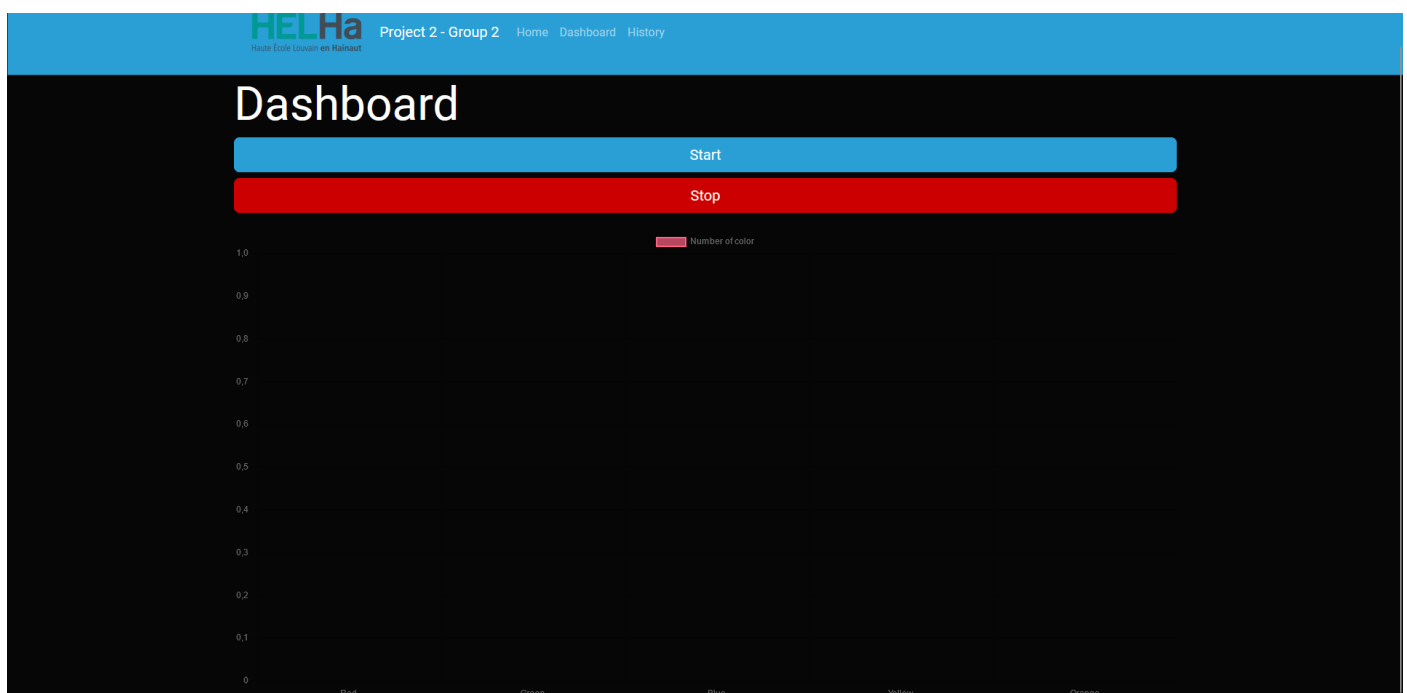


Cette page est la page d'accueil, elle salue l'utilisateur et crédite les développeurs.

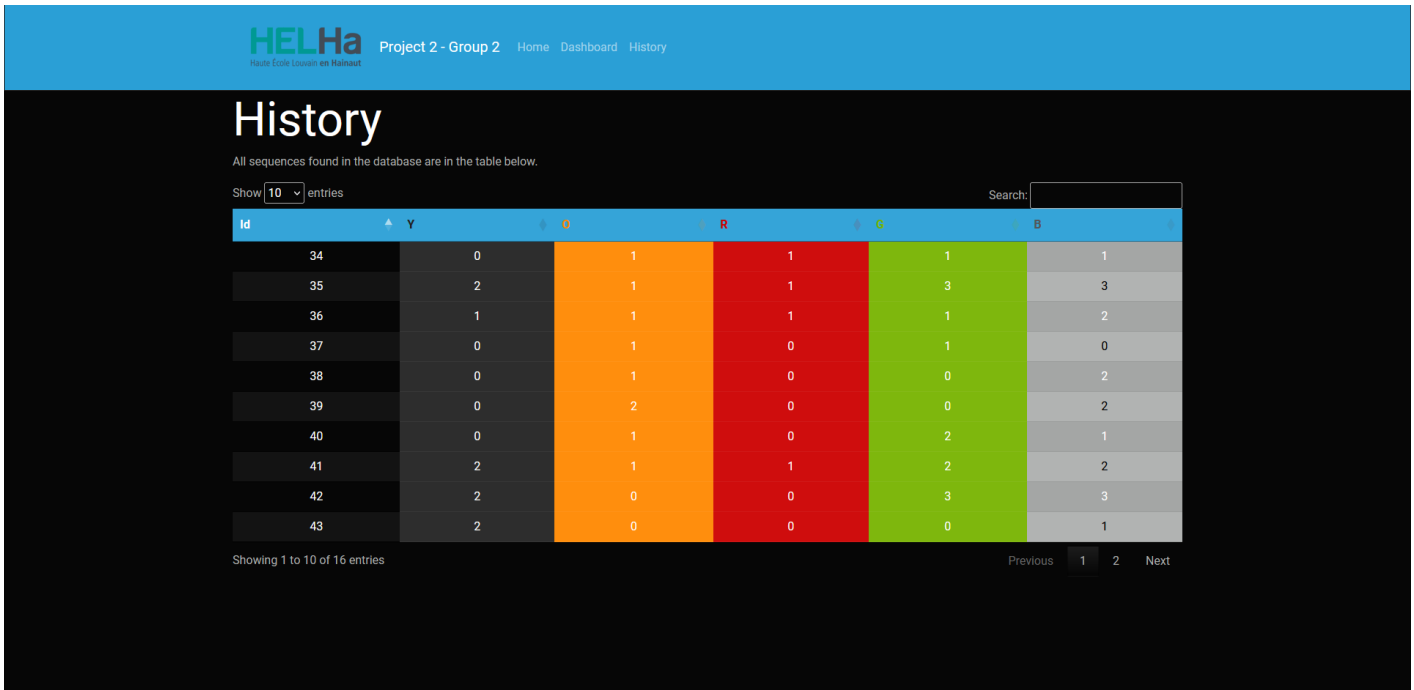
6.3 Dashboard



Le dashboard contient 2 boutons et un graphique. Lorsque l'on appuie sur le bouton « Start » et que la séquence est déjà stoppée, le graphique se remet à 0 et attend que des couleurs soient scannées. Lorsque le bouton, « Stop » est pressé, la séquence s'arrête. Si une couleur est détectée par le capteur, pas besoin de rafraichir la page, le graphique se met automatiquement à jour.



6.4 History



The screenshot shows the 'History' page of the HELHa application. The page has a blue header with the HELHa logo and navigation links: Project 2 - Group 2, Home, Dashboard, and History. Below the header, the title 'History' is displayed, followed by the text 'All sequences found in the database are in the table below.' There is a search bar and a dropdown menu to show 10 entries. The table displays 16 entries, with the first 10 visible. The table has columns for 'id', 'Y', 'D', 'R', 'G', and 'B'. The rows are numbered 34 to 43. The 'D', 'R', 'G', and 'B' columns are color-coded: orange for 'D', red for 'R', green for 'G', and grey for 'B'. The 'Y' column is grey. The table shows the following data:

id	Y	D	R	G	B
34	0	1	1	1	1
35	2	1	1	3	3
36	1	1	1	1	2
37	0	1	0	1	0
38	0	1	0	0	2
39	0	2	0	0	2
40	0	1	0	2	1
41	2	1	1	2	2
42	2	0	0	3	3
43	2	0	0	0	1

At the bottom of the table, it says 'Showing 1 to 10 of 16 entries'. There are also 'Previous', '1', '2', and 'Next' navigation links.

La page historique contient un tableau de toutes les valeurs des dernières séquences. Elle ne se met pas à jour toute seule. Nous pouvons choisir combien d'entrée maximum nous voulons.

6.5 Route

```

/* GET home page. */
router.get('/', function(req, res, next) {
  console.log("Accès route principale");
  res.render('index', { title: 'Projet multidisciplinaire 2' });
});

router.get('/data', function(req, res, next) {
  retrieveData();
  res.render('data', { title: 'Projet multidisciplinaire 2', historyData:historyData });
});

router.get('/api/data', (req,res)=>{
  retrieveData();
  res.json({historyData:historyData});
});

router.get('/history', function(req, res, next) {
  retrieveData();
  res.render('history', { title: 'Projet multidisciplinaire 2', historyData:historyData });
});

async function retrieveData(){
  historyData = await prisma.History.findMany()
  console.log(historyData)
}

module.exports = router;

```

6.5.1 Route « / »

Cette route sert tout simplement à afficher la page d'accueil « index ». Elle récupère le contenu du fichier index.twig (html) et l'affiche sur la page « Home » du site.

```

{% extends 'layout.twig' %}

{%block body %}
  <div class="container">
    <h1>Welcome!!</h1>
    <p>This project was made by Samir and Théo (group2)</p>
  </div>
  <div class="position-absolute top-100 start-50 translate-middle">
    <p>TandS&copy; | BM 2023 - {{ 'now' | date('Y')}}</p>
  </div>
{% endblock %}

```

6.5.2 Route « /data »

Cette route sert à afficher la page « data ». Elle récupère les information du fichier data.twig et l'affiche sur la page « Dashboard » du site.

```

{% extends 'layout.twig' %}

```

```
{% block body %}

    <div class="container">
        <h1>Dashboard</h1>
        <div class="d-grid gap-2">
            <button class="btn btn-lg btn-primary" type="button"
id="btnStart">Start</button>
            <button class="btn btn-lg btn-danger" type="button"
id="btnStop">Stop</button>
        </div>
        <br>
        <div>
            <canvas id="myChart"></canvas>
        </div>

    </div>
{% endblock %}
{% block scripts %}
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></scri
pt>

<script src="javascripts/data.js"></script>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

{% endblock %}
```

Ici, la page fait aussi appel au fichier JavaScript data.js que voici :

```
$(function () {
    const ctx = document.getElementById('myChart');
    var myChart;

    // Déplacez la création du graphique en dehors de la fonction update()
    myChart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ['Red', 'Green', 'Blue', 'Yellow', 'Orange'],
            datasets: [{
                label: 'Number of color',
                data: [],
                backgroundColor: [
```

```

        'rgba(255, 99, 132, 0.7)',
        'rgba(127, 255, 0, 0.7)',
        'rgba(54, 162, 235, 0.7)',
        'rgba(255, 205, 86, 0.7)',
        'rgba(255, 159, 64, 0.7)',
    ],
    borderColor: [
        'rgba(255, 99, 132)',
        'rgba(0, 100, 0)',
        'rgba(54, 162, 235)',
        'rgba(255, 205, 86)',
        'rgba(255, 159, 64)',
    ],
    borderWidth: 2
  }]
},
options: {
  scales: {
    y: {
      beginAtZero: true
    }
  }
}
});

$("#btnStart").click(function (e) {
  e.preventDefault();
  $.ajax({
    type: "get",
    url: "http://192.168.0.13:1880/data",
    data: { status: true },
    dataType: "jsonp",
    success: function (response) {
      console.log(response);
    }
  });
});

$("#btnStop").click(function (e) {
  e.preventDefault();
  $.ajax({
    type: "get",
    url: "http://192.168.0.13:1880/data",
    data: { status: false },
    dataType: "jsonp",
    success: function (response) {

```

```

        console.log(response);
    }
    });
});

setInterval(update, 500);

function update() {
    $.ajax({
        url: "/api/data",
        success: function (data) {

            // Mettez à jour les données du graphique
            myChart.data.datasets[0].data = [
                data.historyData[data.historyData.length - 1].r_tot,
                data.historyData[data.historyData.length - 1].g_tot,
                data.historyData[data.historyData.length - 1].b_tot,
                data.historyData[data.historyData.length - 1].y_tot,
                data.historyData[data.historyData.length - 1].o_tot
            ];

            // Mettez à jour le graphique
            myChart.update();
        }
    });
}
});

```

6.5.3 Route /api/data

Cette route sert à récupérer les données de la base de données grâce à la fonction `retrieveData()`. Ensuite, une réponse contenant les données au format json sera envoyée (`historyData`).

6.5.4 Route /history

Cette route sert à afficher la page « history ». Elle récupère les informations du fichier `history.twig` et l'affiche sur la page « History » du site.

```

{% extends 'layout.twig' %}

{% block stylesheet %}
<link href="https://cdn.datatables.net/v/dt/dt-1.13.4/datatables.min.css"
rel="stylesheet"/>
{% endblock stylesheet %}

{% block scripts %}

```

```

<script src="https://cdn.datatables.net/v/dt-
1.13.4/datatables.min.js"></script>
<script defer src="/javascripts/dataTable.js"></script>
{% endblock scripts %}

{% block body %}
<div class="container">
  <h1>History</h1>
  <p>
    All sequences found in the database are in the table below.
  </p>
  <table class="table table-responsive table-striped text-center"
id="myTable">
    <thead class="table-primary">
      <tr>
        <th>Id</th>
        <th class="text-light">Y</th>
        <th class="text-warning">O</th>
        <th class="text-danger">R</th>
        <th class="text-success">G</th>
        <th class="text-secondary">B</th>
      </tr>
    </thead>
    <tbody id="table-body">
      {% for data in historyData %}
      <tr>
        <td>{{ data.idHistory }}</td>
        <td class="table-light">{{ data.y_tot }}</td>
        <td class="table-warning">{{ data.o_tot }}</td>
        <td class="table-danger">{{ data.r_tot }}</td>
        <td class="table-success">{{ data.g_tot }}</td>
        <td class="table-dark">{{ data.b_tot }}</td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</div>

{% endblock body %}

```

Ici, la page fait aussi appel au fichier Javascript dataTable.js :

```

$(document).ready(function () {
  $('#myTable').DataTable();
});

```


6.5.5 Fonction retrieveData

La fonction retrieveData sert à stocker dans la variable historyData les valeurs reçues de la commande prisma.History.findMany(). Cette commande va chercher grâce à Prisma les valeurs de la base de données « History ».

9. Conclusion

This multidisciplinary project was both interesting and challenging. Despite the short 7-day deadline, we learned a lot about sensors, using of Arduino Mega, MySQL and nodeJS. Working in groups taught us valuable teamwork and time management skills. Overall, the project allowed us to apply theoretical knowledge in a practical setting and gain valuable experience in the web and embedded system domains.

10. Appendices

7.1 Datasheets



I2C_1602_LCD.pdf

LCD-I2C:



A000067-datasheet.
pdf

BTN-RED:

10.2 Code

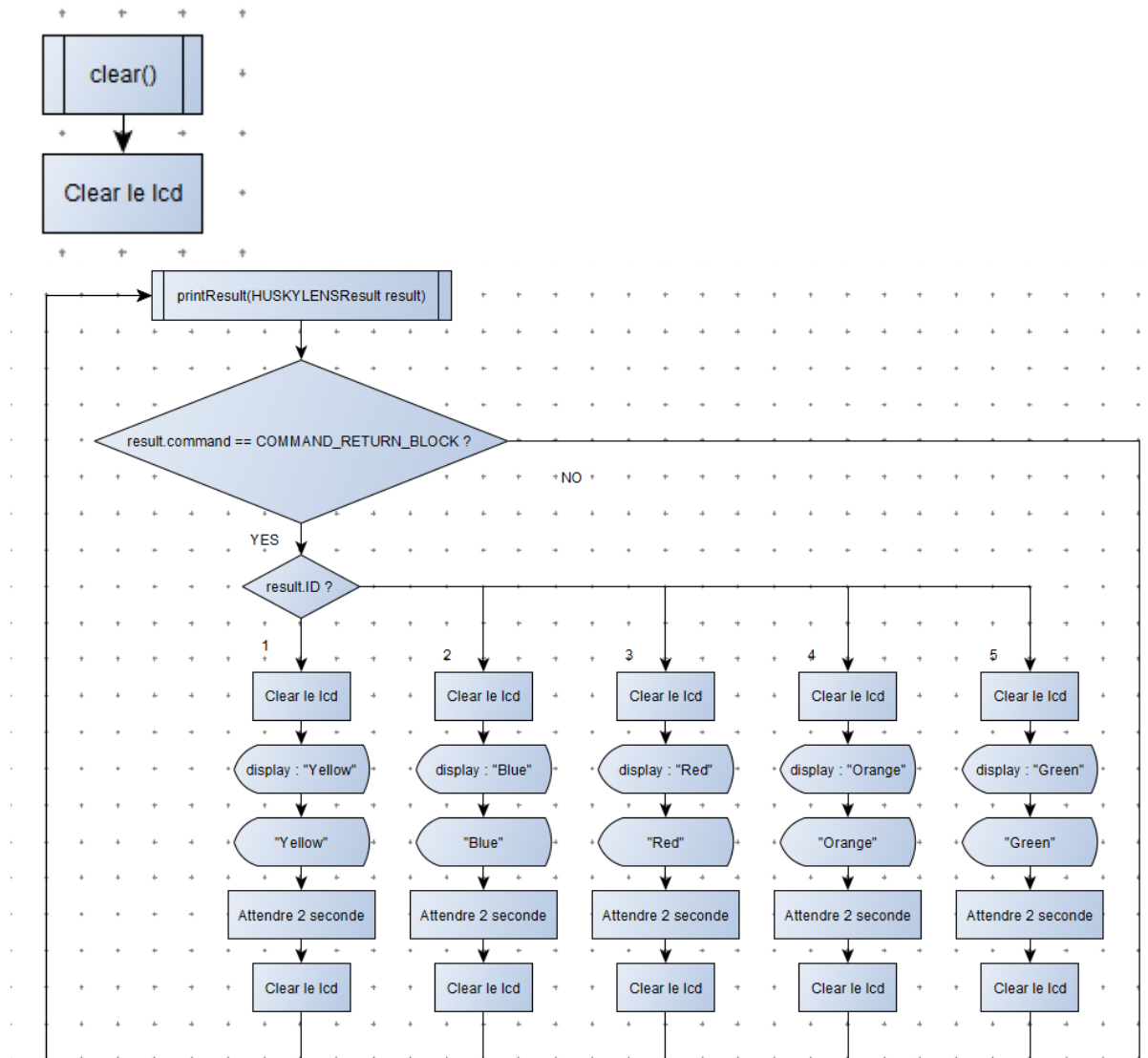
10.2.1 Web

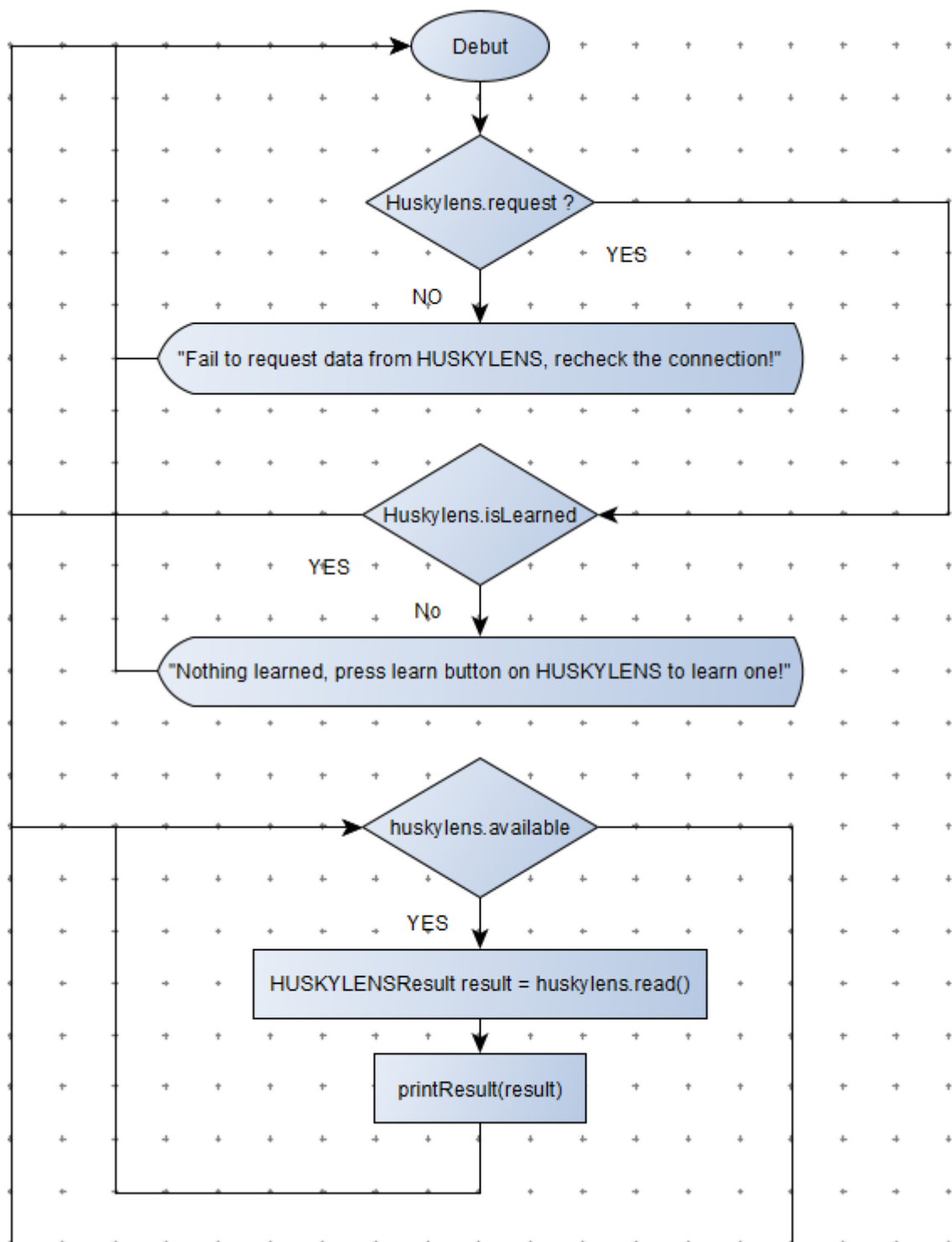
Tout le code web est présent dans le GitHub :

<https://github.com/TheoDelaere/Project-2>

10.2.2 Arduino

Analyse Arduino :





Le code Arduino :

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "HUSKYLENS.h"
#include "SoftwareSerial.h"

HUSKYLENS huskylens;
SoftwareSerial mySerial(10, 11); // RX, TX
//HUSKYLENS green line >> Pin 10; blue line >> Pin 11

LiquidCrystal_I2C lcd(0x27, 16, 2);
byte btn = 2;

void clear() {
  lcd.clear();
}

void printResult(HUSKYLENSResult result);

void setup() {
  // put your setup code here, to run once:

  pinMode(btn, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(btn), clear, FALLING);
  lcd.init();
  lcd.backlight();
  Serial.begin(115200);
  mySerial.begin(9600);

  while (!huskylens.begin(mySerial))
  {
    Serial.println(F("Begin failed!"));
    Serial.println(F("1.Please recheck the \"Protocol Type\" in
HUSKYLENS (General Settings>>Protocol Type>>Serial 9600)"));
    Serial.println(F("2.Please recheck the connection."));
    delay(100);
  }
}

void loop() {
  // put your main code here, to run repeatedly:

  if (!huskylens.request()) Serial.println(F("Fail to request data from
HUSKYLENS, recheck the connection!"));
  else if (!huskylens.isLearned()) Serial.println(F("Nothing learned,
press learn button on HUSKYLENS to learn one!"));
  else
  {
    while (huskylens.available())

```

```
{
    HUSKYLENSResult result = huskylens.read();
    printResult(result);
}

}

}

void printResult(HUSKYLENSResult result){
    if (result.command == COMMAND_RETURN_BLOCK){
        switch(result.ID){

            case 1:
                lcd.clear();
                lcd.print("Yellow");
                Serial.println("Yellow");
                delay(2000);
                result.ID=0;
                lcd.clear();
                break;

            case 2:
                lcd.clear();
                lcd.print("Blue");
                Serial.println("Blue");
                delay(2000);
                result.ID=0;
                lcd.clear();
                break;

            case 3:
                lcd.clear();
                lcd.print("Red");
                Serial.println("Red");
                delay(2000);
                result.ID=0;
                lcd.clear();
                break;

            case 4:
                lcd.clear();
                lcd.print("Orange");
                Serial.println("Orange");
                delay(2000);
                result.ID=0;
                lcd.clear();
                break;

            case 5:
                lcd.clear();
```

```

    lcd.print("Green");
    Serial.println("Green");
    delay(2000);
    result.ID=0;
    lcd.clear();
    break;

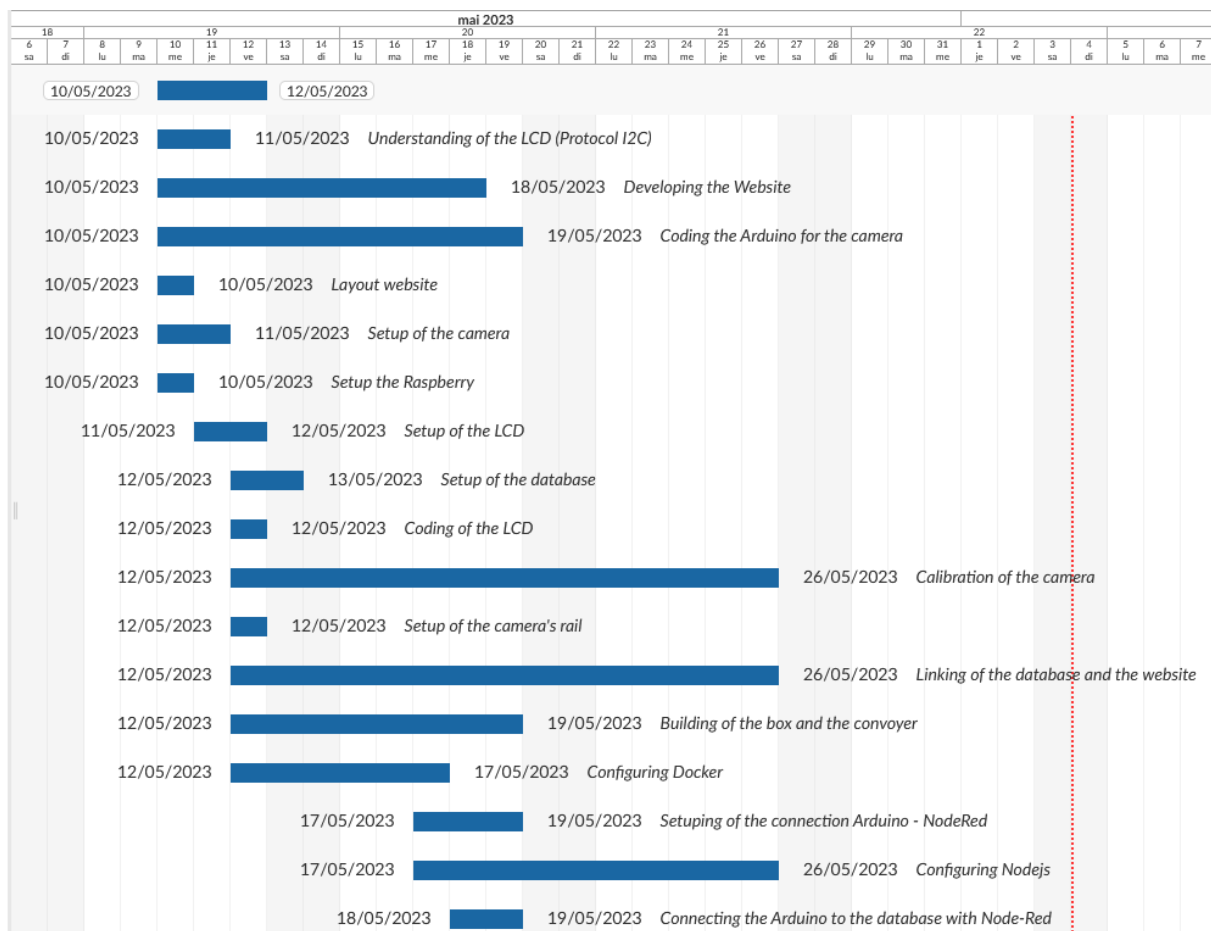
  }

}

}

```

10.3 GANTT



11. Webographie

Arduino - LCD I2C / Arduino Getting Started. (s. d.). Arduino Getting Started.

<https://arduinogetstarted.com/tutorials/arduino-lcd-i2c>

GitHub : Let's build from here. (s. d.). GitHub.

<https://raw.githubusercontent.com/>

Wokwi - Online ESP32, STM32, Arduino Simulator. (s. d.).

<https://wokwi.com/>

Featured Products. (s. d.).

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

Mega 2560 Rev3 / Arduino Documentation. (s. d.).

<https://docs.arduino.cc/hardware/mega-2560>

Chart.js. (s. d.). Open source HTML5 Charts for your website.

<https://www.chartjs.org/>

Prisma. (s. d.). Prisma / Next-generation ORM for Node.js & TypeScript.

<https://www.prisma.io/>

Node-RED. (s. d.).

<https://nodered.org/>

MySQL. (s. d.).

<https://www.mysql.com/fr/>