INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

RGB CLOCK – SAE ALTERNANT

Alexis BOURDEAU
Romain BUTET

Marion NORMAND
Maxime ROCHE

Tuteur : Philippe LUCIDARME

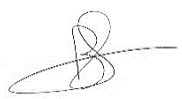
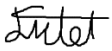
Décharge de responsabilités

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

Licence

BOURDEAU Alexis, BUTET Romain, NORMAND Marion, ROCHE Maxime, auteurs du présent rapport, publions et divulguons celui-ci sous la Licence Creative Commons suivante « CC BY » pour le monde entier et pendant la durée légale de protection des droits d'auteur. Cette licence autorise la représentation, la reproduction, la modification, la création d'œuvres dérivées et l'utilisation y compris à des fins commerciales sous réserve de mentionner les noms et prénoms des auteurs.

LICENCE CC ATTRIBUTION:

BOURDEAU Alexis	BUTET Romain	NORMAND Marion	ROCHE Maxime
			

Charte de mon usage de l'IA

NOTE SUR LA CONFIDENTIALITÉ

SI VOTRE RAPPORT EST CONFIDENTIEL, NOUS VOUS INVITONS À PRENDRE LA RESPONSABILITÉ DES OBLIGATIONS MORALES QUI VOUS INCOMBENT QUANT À LA DIFFUSION D'INFORMATIONS CONFIDENTIELLES À DES TIERS SANS L'ACCORD PRÉALABLE DE L'ENTREPRISE.

DANS CE CAS, RECOURIR À L'IA N'EST PAS RECOMMANDÉ.

1. Après avoir rendu un document qui sera examiné par un logiciel de détection de plagiat et d'utilisation de l'IA, je déclare :

□ Ne pas avoir utilisé l'IA dans mon travail.

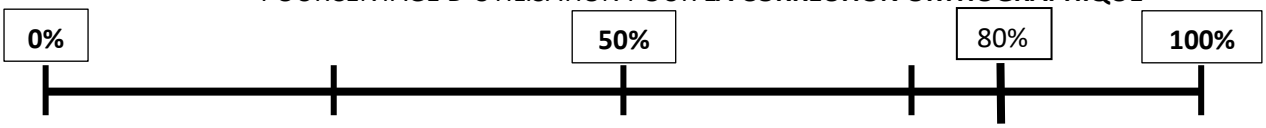
☒ Avoir utilisé l'IA dans mon travail.

1.1. Si vous avez décidé de ne pas l'utiliser, qu'est-ce qui a motivé votre décision ?

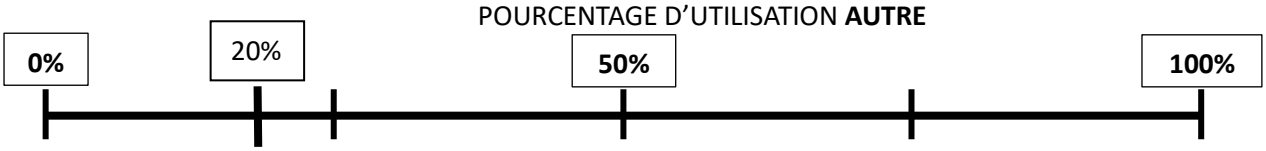
(L'IA ne vous a pas semblé utile ? Vous avez une opinion négative des IA ? Vous avez préféré mobiliser vos propres compétences ? Votre rapport contient des données sensibles ?)

1.2. Si vous avez utilisé l'IA, estimez la proportion de votre travail qui a été impactée par l'utilisation de l'IA. (*Ce pourcentage sera comparé aux résultats du logiciel de détection de plagiat et d'utilisation de l'IA*)

POURCENTAGE D'UTILISATION POUR **LA CORRECTION ORTHOGRAPHIQUE**



POURCENTAGE D'UTILISATION **AUTRE**



2. Cochez les cases correspondant à votre utilisation de l'IA pour ce travail :

EXPLORATION	**AMÉLIORATION**
☒ S'inspirer □ Générer des idées □ Approfondir un sujet □ Recherche documentaire ☐ …	☒ Corriger ses fautes ☒ Reformuler ses phrases □ Demander des conseils ☐ … □ …
RÉSOLUTION DE TÂCHES	**PRODUCTION DE CONTENUS**
☐ Synthétiser □ Répondre à une question □ Réaliser des calculs □ Analyser des données □ Débugger un programme	□ Générer un plan □ Rédiger des parties d'un texte □ Générer des médias (image, son, vidéo, ...) □ Générer du code informatique □ …

3. Mon utilisation de l'IA était-elle sécuritaire ? Développez.
L'aspect sécuritaire consiste principalement dans la fuite de données ou d'informations personnelles, académiques ou appartenant à des entreprises.

Nous avons utilisé l'IA pour un projet donc il n'y a pas d'informations personnelles ou sur une entreprise. Nous avons utilisé l'IA uniquement pour nous inspirer ou faire corriger les fautes dans un texte et reformuler certaines phrases. Nous étions tous d'accord pour utiliser ces IAs et connaissons tous les conséquences.

4. Mon utilisation de l'IA était-elle critique ? Développez.
L'aspect critique concerne votre recul face aux réponses des IA. En quoi avez-vous pu les critiquer ? Les remettre en cause ? Vous assurer de leur véracité ?

Nous avons utilisé l'IA seulement comme source d'inspiration en vérifiant certaines informations sur de vrais sites.

5. Qu'est-ce que vous avez gagné à utiliser l'IA ?
(Quelles difficultés vous a-t-elle permis de contourner ? Quels bénéfices pensez-vous avoir tiré de son utilisation ?)

Nous avons gagné du temps car nous n'avions pas aller chercher sur différents sites pour une simple idée. Si nous devions vérifier une information, nous allions sur un site précis où nous avions directement la réponse. Même si nous devions vérifier la véracité de cette réponse, cela nous faisait quand même gagner du temps.

6. Quelles IA avez-vous utilisées précisément ? Pour quels usages ?

Chat GPT pour nous donner une source d'inspiration, Quillbot pour corriger les fautes et quelques fois pour reformuler certaines phrases. Nous avons utilisé Quillbot pour vérifier le pourcentage d'IA potentiel que notre texte pouvait ressortir, même si la grande majorité de nos textes sont écrits à la main.

Nous attestons sur l'honneur que mes réponses aux questions précédentes sont exactes.

Remerciements

Nous tenons tout d'abord remercier M LUCIDARME, pour ses conseils, son accompagnement, le temps qu'il a pu nous accorder pour la réalisation de ce projet et sa confiance pour l'accès à la salle C34.

Nous remercions également le personnel de l'IUT qui nous est venu en aide : Mme DENECHEAU pour la commande du PCB, M TOUSSAINT pour son aide technique ainsi que M DERENNE pour son expertise et la découpe des matériaux.

Nous souhaitons remercier M RAYER pour la découpe laser de deux de nos pièces.

Sommaire

Introduction

Dans le cadre de notre deuxième année de BUT GEII (Génie Électrique et Informatique Industrielle) à l'IUT d'Angers, et plus précisément dans le cadre du cours « SAE Alternance », nous avons eu l'opportunité de développer nos compétences en travail d'équipe, en conception de circuits électroniques, en conception assistée par ordinateur et en programmation C.

Après avoir vu plusieurs projets, tous aussi intéressants les uns que les autres, notre choix s'est porté sur le projet « RGB CLOCK ». Ce projet consiste à concevoir une horloge connectée capable d'afficher l'heure en temps réel grâce à une connexion Wi-Fi. En plus de cette fonctionnalité principale, elle intègre également la mesure de la température et de l'humidité ambiantes. Son affichage repose sur une matrice LED 8x32 pilotée par un microcontrôleur ESP32-C6.

Ce rapport a pour objectif de détailler l'ensemble du processus de conception, de développement et d'implémentation de cette horloge connectée, réalisée par Alexis BOURDEAU, Romain BUTET, Marion NORMAND et Maxime ROCHE.

Cahier des charges

1. Présentation du projet

   Le projet RGBClock a pour objectif de réaliser une horloge numérique à LED basée sur une matrice RGB pilotée par un microcontrôleur (type ESP32). L'affichage doit être clair, lisible et capable de présenter diverses informations telles que l'heure, la date, la température et l'humidité, avec des effets lumineux personnalisés.

2. Objectifs fonctionnels

   - Récupérer l'heure exacte via une connexion internet et l'afficher en temps réel, avec mise à jour des secondes
   - Afficher la date
   - Afficher la température et le taux d'humidité
   - Offrir une interface simple pour configurer le réseau Wi-Fi, via un portail captif
   - Changer d'affichage toutes les 5 s et rester sur l'heure pendant 1 min
   - Revenir à l'affichage précédent avec un bouton

3. Matériel utilisé

   - Microcontrôleur prenant en charge le Wi-Fi
   - Matrice LED RGB 8x32
   - Capteur de température et d'humidité
   - Alimentation adaptée
   - Boutons pour changer le mode d'affichage

4. Fonctionnalités logicielles

   - Création de bibliothèques pour chaque partie
   - Création d'une table ASCII pour afficher les caractères nécessaires
   - Algorithmes optimisés pour l'affichage de l'heure sans scintillement
   - Fonctions spécifiques :
     - showDate() : affiche la date
     - showTemp() : affiche la température
     - showHumidity() : affiche le taux d'humidité
     - showHeure() : mise à jour optimisée de l'heure sans effacer tout l'écran
   - Portail captif pour saisie des identifiants Wi-Fi

5. Contraintes

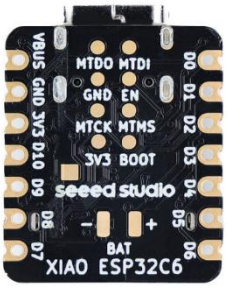
   - Le système doit démarrer automatiquement à la mise sous tension
   - Le système doit pouvoir s'adapter en cas d'absence du réseau internet habituel
   - Le système doit être capable d'informer l'utilisateur en cas d'erreur

6. Livrables attendus

- Rapport détaillant le processus de conception et d’assemblage du prototype
- Schéma de câblage
- Code source documenté
- Fichiers de conception 3D
- Prototype fonctionnel

Choix technologiques

L'ESP32-C6 prend en charge le Wi-Fi 6 (2,4 GHz). Il intègre aussi le Bluetooth Low Energy 5.3 et la radio 802.15.4 pour les protocoles Thread et Zigbee. Son but principal est d'offrir une connectivité rapide et faible latence. C’est donc un microcontrôleur qui s’adapte tout à fait à notre projet et qui permettrait beaucoup d’améliorations futures.



Le bloc d'alimentation : après avoir mesuré les courants consommés par la matrice LED, nous avons convenu de trouver une alimentation capable de délivrer les 9 ampères nécessaires lorsque toutes les LEDs sont allumées en blanc. Nous avons donc choisi une alimentation qui convertit le secteur en courant continu 5 V (pour l'ESP32 et la matrice) et qui peut fournir le courant nécessaire à la matrice.

Le bloc d’alimentation choisi : SDM50-5-U-P6 de la marque CUI INC.

Phase 1 – Conception

1. Schéma fonctionnel

Le projet consiste en la création d'une horloge connectée à Internet, permettant de récupérer l'heure automatiquement et de l'afficher. En plus de l'heure, l'horloge est capable de mesurer la température ambiante ainsi que le taux d'humidité. L'ensemble du système est alimenté par le secteur, permettant une utilisation continue. Des boutons poussoirs seront placés derrière l'horloge, permettant à l'utilisateur de régler l'affichage selon ses préférences.

Résumons le projet : le microcontrôleur, le capteur, les connecteurs des boutons, le connecteur de la matrice LED et le connecteur d'alimentation seront intégrés sur un PCB conçu et routé par Romain. Le microcontrôleur choisi pour ce projet est un XIAO studio ESP32 C6, qui se connecte à Internet via Wi-Fi pour récupérer l'heure grâce au protocole NTP (Network Time Protocol).

L'affichage des données s'effectue à l'aide d'une matrice LED de 8x32 offrant une visibilité claire. Une grille noire imprimée en 3D est utilisée pour rendre l'affichage plus délimité. La gestion des LEDs est faite par l'ESP32, qui utilise les ports GPIO, en s'appuyant sur des librairies développées par Marion.

Le capteur de température, un SHT45, est monté en CMS sur le PCB, et communique avec le microcontrôleur via le protocole I2C. Les données recueillies sont alors traitées par le microcontrôleur et affichées sur la matrice LED, permettant à l'utilisateur de visualiser les informations en temps réel.

Les boutons poussoirs, déportés du PCB, sont connectés à l'ESP32 sur les ports digitaux. Leur fonction principale est de permettre à l'utilisateur de régler l'heure manuellement et de basculer entre les différentes options d'affichage, notamment l'heure, la date, et les mesures de température.

Nous avons donc toutes les informations nécessaires pour la réalisation du premier schéma fonctionnel :

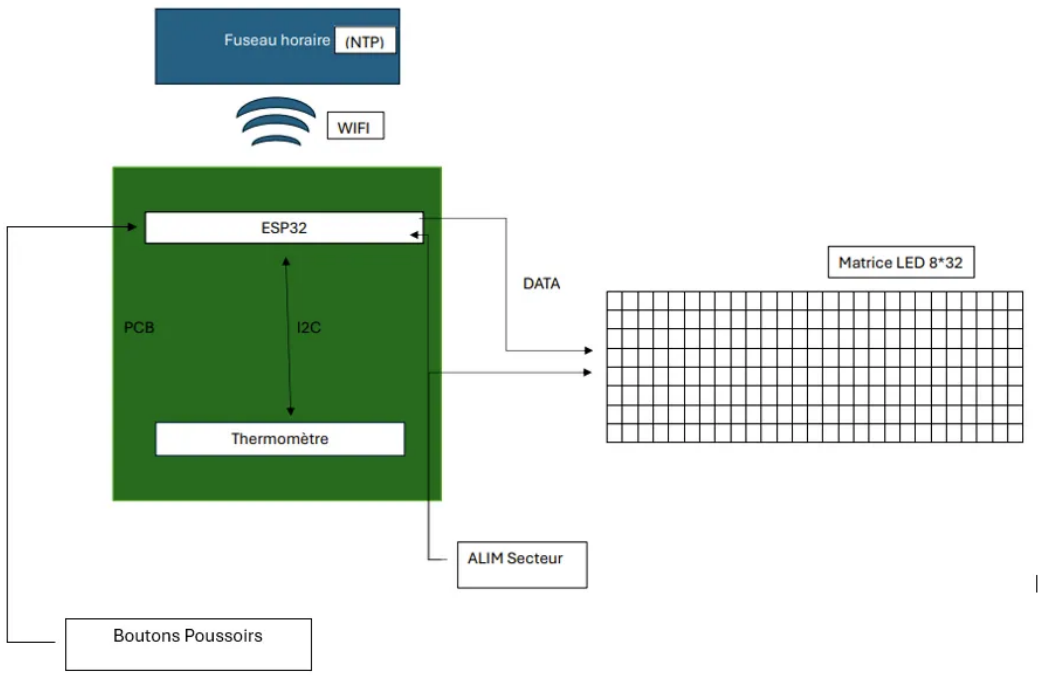


Schéma fonctionnel RGB Clock

2. Choix et commande des composants

Pour ce projet, nous avons dû commander plusieurs composants :

Le capteur d'humidité SHT45 CMS : ce choix a été fait en raison de la fiabilité de ce composant et de sa taille réduite.

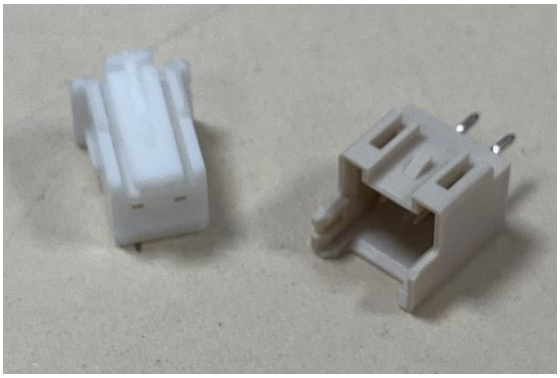
La référence du capteur : SHT45-AD1F-R2 fabriqué par Sensirion.

Le connecteur d'alimentation jack : il a fallu le dimensionner en tenant compte du courant nécessaire pour la matrice. Nous avons donc choisi un connecteur jack compatible avec le bloc d'alimentation, capable de supporter un courant allant jusqu'à 9 A.

Le connecteur d'alimentation jack choisi : PJ-063BH fabriqué par Same Sky.

3 boutons poussoirs : le choix des boutons poussoirs s'est principalement basé sur la couleur, plutôt que sur la forme. Nous avions uniquement besoin de boutons poussoirs déportés pour commander manuellement l'affichage et l'heure. Trois couleurs différentes ont donc été choisies.

Les boutons poussoirs choisis : ISR3SAD200 (noir), ISR3SAD600 (rouge), ISR3SAD300(vert), de la marque APEM.



Pour connecter les Boutons Poussoirs à notre PCB, nous avons choisi des connecteurs XA 2 Points. Tout d'abord parce que la section de fils des BP est suffisamment petite pour que ces petits connecteurs suffisent et ensuite pour avoir un ensemble qui soit démontable rapidement et facilement.

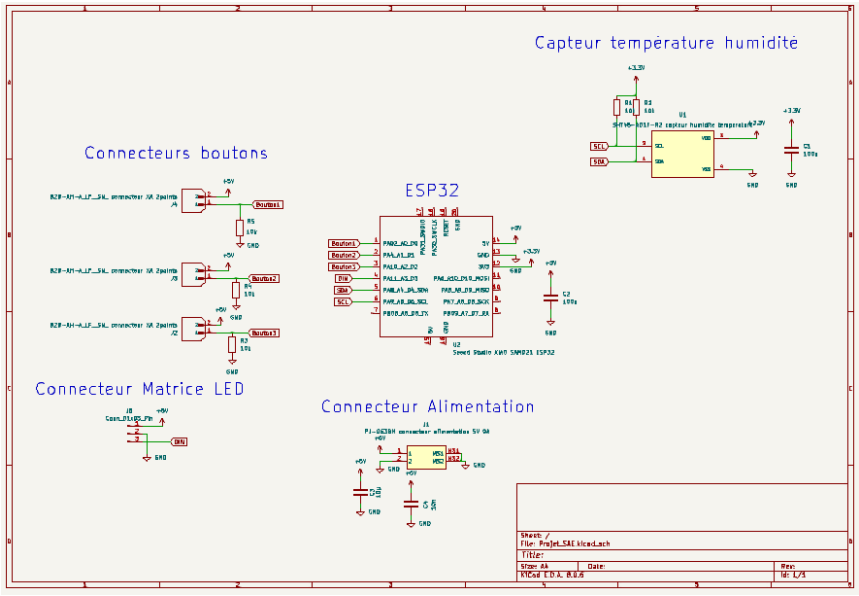
Nomenclature de la commande :

Fournisseur	Quantité	Référence distributeur	Référence fabricant	Description
Mouser Electronics	1	403-SHT45-AD1F-R2	SHT45-AD1F-R2	Capteurs d'humidit enfichables Digital Humidity and Temperature Sensor incl Filter Membrane
Mouser Electronics	1	490-SDM50-5-U-P6	SDM50-5-U-P6	ptateurs CA de bureau ac-dc, 5 Vdc, 9 A, SW, C14 desk-top, P6 center pos plug, no cord, level VI, M
Mouser Electronics	1	490-PJ-063BH	PJ-063BH	DC Power Connectors Power Jacks
Radiospares	1	373-1656	ISR3SAD200	Bouton-poussoir série IS Series, Momentané, SPST, Montage panneau
Radiospares	1	373-1678	ISR3SAD600	Bouton-poussoir série IS Series, Momentané, SPST, Montage panneau
Radiospares	1	373-1662	ISR3SAD300	Bouton-poussoir série IS Series, Momentané, SPST, Montage panneau
Radiospares	4	498-1149	U5125	Manchon APEM pour Interrupteur à bouton-poussoir série IP;IB-IS

Alternatives étudiées des composants :

	+	-
Same Sky PJ-002B : 	5V DC, Jack 2.5mm	Courant max = 2.5A Tout en plastique : risque d'usure/fatigue
TE Connectivity / Alcoswitch 2492672-4 : 	Résistant Choix de couleurs LED Integrée	Trop gros/ imposant pour l'horloge

3. Schéma du PCB



Pour réaliser le schéma, il fallait récupérer les différentes empreintes des composants qui n'étaient pas inclus dans les bibliothèques d'origine sur KiCAD. Une fois ces composants importés, nous avons créé le schéma sur lequel nous avons ajouté les composants.

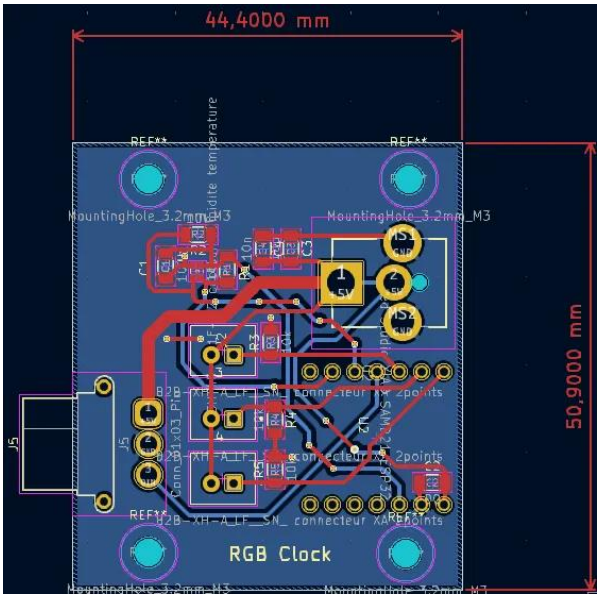
Nous avons commencé par connecter l'alimentation aux bornes de l'ESP32 et l'alimentation de la matrice LED en 5 V, puis celle du capteur de température en 3.3V avec la sortie de l'ESP32. Des condensateurs de découplage de 100 nF sont ajoutés aux bornes de chaque alimentation et 2 condensateurs de découplage (10 µF et 10 nF) sont aux bornes de l'alimentation principale.

L'alimentation 5V sera amenée d'un côté du connecteur des boutons, puis sur l'autre broche du connecteur sera positionnée une résistance de pull-down et la sortie sera connectée sur une broche de l'ESP32. Les 3 boutons seront connectés sur les broches 1, 2 et 3 de l'ESP32. Le signal qui commande la matrice LED (DIN) est connecté à la broche 4 de l'ESP32. Enfin, les 2 résistances situées autour du capteur de

température sont conformes au schéma fourni par le constructeur. Le signal SDA et SCL seront respectivement sur les broches 5 et 6 de l'ESP32.

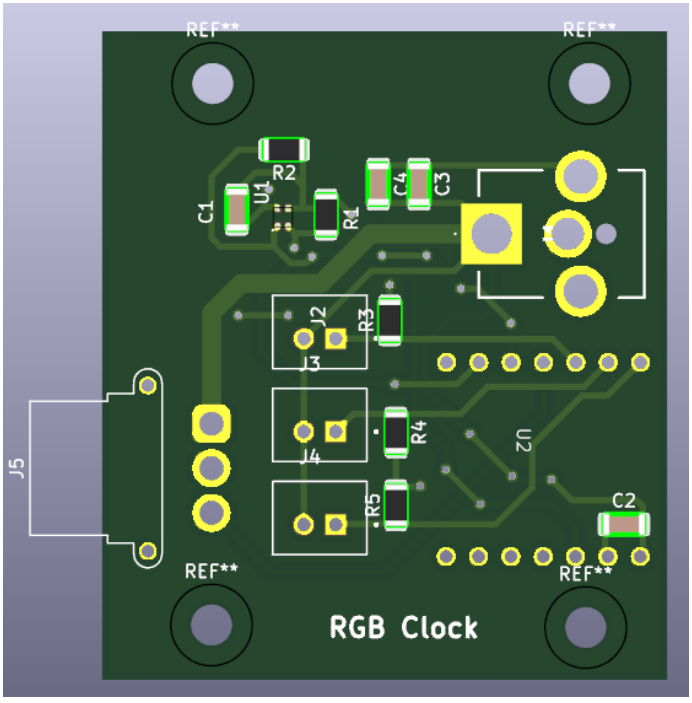
4. Routage du PCB

Pour réaliser le routage, il fallait d'abord définir l'emplacement des composants comme l'alimentation, l'ESP32, le connecteur de la matrice LED et le capteur de température.



Nous devions ensuite importer les empreintes de tous les composants. Nous avons dû en créer une nouvelle pour celle de l'ESP32, car celle que nous avions ne nous convenait pas. L'empreinte possédait trop de broches et prenait donc beaucoup de place sur notre PCB. Celle que nous avons créée possède seulement les broches dont nous avons besoin.

L'alimentation de la carte et l'ESP32 a été placée sur la droite du PCB pour un accès simple et rapide une fois la carte positionnée dans le coffrage en bois. Le connecteur de la matrice LED est placé à l'opposé de l'ESP32 afin de faciliter la connexion.



Le capteur de température/humidité est situé en haut du PCB, loin des autres composants, pour éviter de capter la température de ces composants. Une fois la carte montée à l'arrière du coffrage, le capteur sera en bas pour une meilleure acquisition de la température.

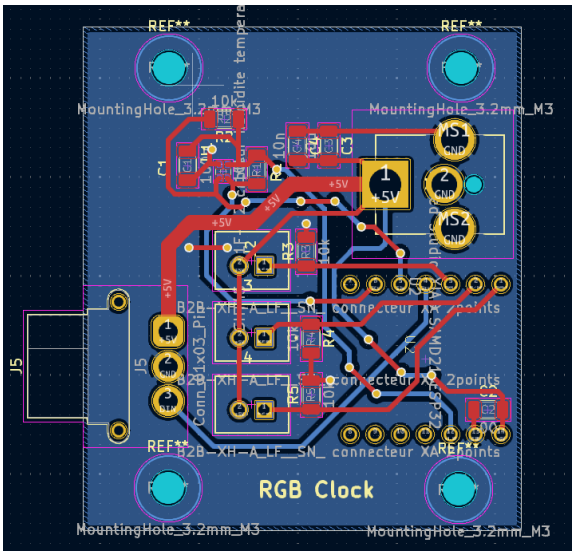
Nous avons dû dimensionner en conséquence la piste reliant l'alimentation générale et le 5V du connecteur de la matrice LED afin qu'elle puisse accepter le courant nécessaire pour la matrice.

Nous avons décidé de positionner l'ESP32 sur un support DIL afin de pouvoir l'enlever simplement. Pour finir, nous avons placé 4 entretoises reliées à la masse à chaque coin de la carte pour pouvoir la monter à l'arrière du coffrage.

Pour dimensionner la carte, il ne fallait pas dépasser la taille de la matrice LED soit 8 cm par 32 cm. La taille finale de cette carte est de 50.9 mm par 44.4 mm.

Nous nous sommes rendu compte lors du câblage de notre PCB que l'alimentation était en court-circuit et donc la carte n'était pas alimentée et ne fonctionnait pas.

Nous avons eu ce défaut car sur l'empreinte du connecteur d'alimentation, au moment du routage, il est écrit que le 5 V est sur les broches 1 et 2 du connecteur, et la masse est présente sur les broches MS1 et MS2. Nous avons donc relié la broche 1 et la broche 2 pensant qu'il s'agissait du même potentiel. Nous avons donc routé un nouveau PCB en supprimant le court-circuit puis en ramenant l'alimentation là où elle n'y était pas, notamment sur le condensateur C2.



À l'avenir, pour éviter de reproduire ce défaut, il faudra mieux regarder la datasheet du composant et ne pas se fier seulement à l'empreinte présente sur le routage.

Nous nous sommes rendus compte d'un autre défaut concernant le PCB. Le bouton 1 qui est connecté à la broche 0 de l'ESP32 doit être connecté sur une autre broche car celle-ci sert à gérer le BOOT de l'ESP32. Le BOOT est activé avec un niveau logique 0, et le bouton possède une pull-down qui met la broche 0 au niveau logique 0 quand le bouton 1 n'est pas appuyé, soit la plupart du temps. De plus, la broche 1 possède une résistance de pull up interne à l'ESP32. Pour un prochain PCB, il faudra décaler d'une broche les boutons pour ne plus avoir ce défaut.

5. Tests fonctionnels de la matrice LED

Avant de réfléchir à l'affichage des différents caractères (heure, température, hygrométrie, paramètres, etc.), nous avons d'abord testé la matrice pour comprendre son fonctionnement et déterminer sa consommation électrique.

Dans un premier temps, nous avons étudié les spécifications techniques grâce à la datasheet : https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf. Cette documentation nous a permis d'identifier les caractéristiques suivantes :

- Spécifications électriques :
- Tension d'alimentation : 3,5 à 5,3 V
- Plage de température de fonctionnement : -25 à 80 °C
- Plage de température de stockage : -40 à 105 °C
- Couleurs des LED :
    - Rouge : 620-625 nm, tension de 2,0-2,2 V
    - Vert : 522-525 nm, tension de 3,0-3,4 V
    - Bleu : 465-467 nm, tension de 3,0-3,4 V

Ne disposant pas d'informations sur la consommation de courant des LED en fonction des couleurs, nous avons effectué des tests en temps réel. Nous avons donc branché le ruban LED comme illustré sur la photo suivante :

L'ESP32-C6 a été programmé à l'aide de l'IDE Arduino avec les paramètres suivants :

Version : 2.1.0 * Librairie installée : FastLED.h

Manager cartes : ESP32 by Espressif Systems version 3.0.5

Fichiers > Préférences > URL de gestionnaire de cartes supplémentaires : https://dl.espressif.com/dl/package_esp32_index.json

Voici le code implémenté dans l'ESP32, inspiré de différents forums Arduino. Ce code utilise la bibliothèque FastLED qui permet un contrôle simple et simultané de toutes les LEDs :

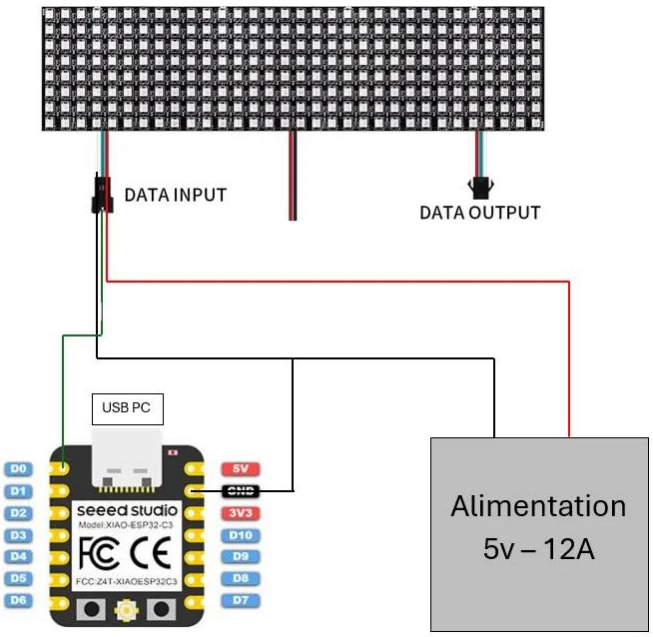
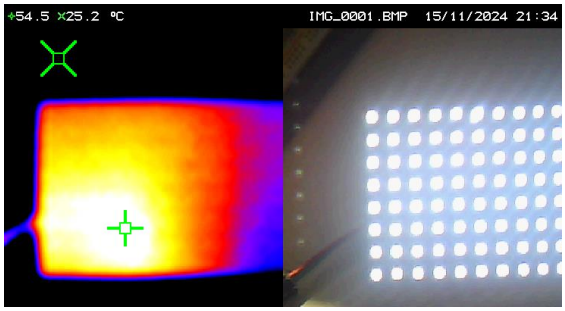
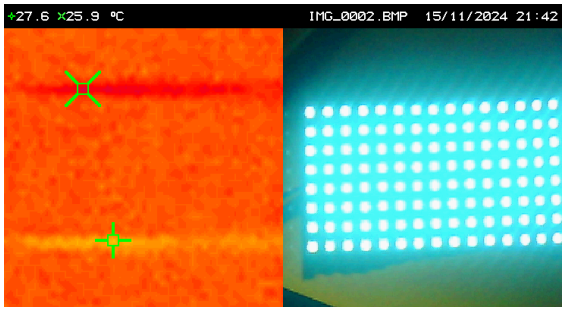
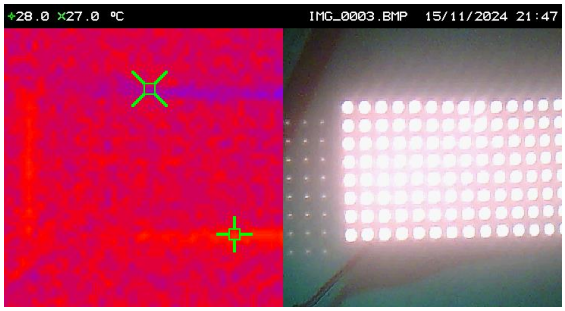


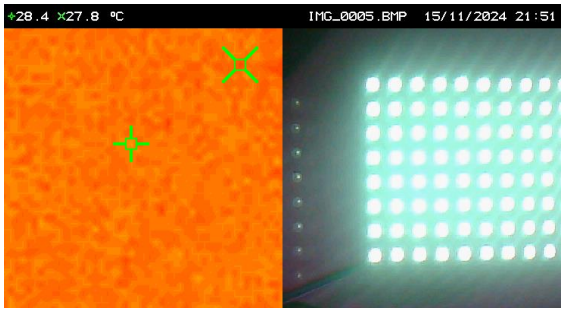
Schéma électrique pour tester toutes les leds de la matrice

```c
#include <FastLED.h>

// Définition du nombre de LEDs (8 lignes x 32 colonnes = 256 LEDs)
#define NUM_LEDS 256
#define DATA_PIN 7  // correspond à D5 sur l'ESP32

// Définition du tableau de LEDs
CRGB leds[NUM_LEDS];

// Configuration de la bibliothèque FastLED
void setup() {
  // Initialisation de la bibliothèque FastLED
  FastLED.addLeds<WS2812B, DATA_PIN, GRB>(leds, NUM_LEDS);

  // Paramètres de la fréquence de mise à jour des LEDs
  FastLED.setBrightness(255);  // Luminosité, entre 0 et 255
}

void loop() {
  // Boucle principale
  for (int i = 0; i < NUM_LEDS; i++) {
    // Affiche l'état actuel des LEDs
    FastLED.show();

    // Définit la couleur de la LED actuelle sur bleu
    leds[i] = CRGB::Blue;
  }
}
```

Une fois toutes les LEDs allumées, nous avons mesuré le courant consommé en fonction des couleurs et la température émise par la matrice à l'aide d'une caméra thermique, en suivant cette procédure :

⇒ Allumer la matrice, attendre 1 minute ; prendre les mesures ; éteindre la matrice pendant 3 minutes ; recommencer avec une nouvelle couleur. Pour la couleur blanche, un temps de repos plus important a été nécessaire pour laisser refroidir le panneau à température ambiante.

Couleur LED	Consommation (A)	Température (°C)
Blanc	6.6	 ➢ **54.5 °C**
Vert	1.7	 ➢ **27.6 °C**
Rouge	3.2	 ➢ **28 °C**

Bleu	3.2	 ➢ **28.4 °C**

Ces tests nous ont permis de démontrer plusieurs points :

- La possibilité de piloter la matrice via un ESP32 facilement grâce à la bibliothèque FastLED.
- La consommation moyenne significative des LEDs selon les couleurs.
- La variation de température de la matrice selon les couleurs.

Phase 2 – Développement Logiciel

Les captures de codes suivant ne sont que des extraits, vous retrouverez en fichier annexes les bibliothèques complètes et commentées.

1. Librairie gestion des pixels

La librairie « matrix » permet de gérer une matrice de LED 8x32. La matrice est composée de LEDs, qui sont de type WS2812. Elles sont également RGB et adressables individuellement.

Le code principale (.ino)

```ino
#include <matrix.h>


void setup()
{
  init();
}

void loop()
{
  // X est la longueur et Y la largeur
  setPixel(5, 6, CRGB::Red, 128);  // Allume la LED à la position (x, y) avec la couleur et l'intensité

  update(); // Affiche les modifications
  delay(1000);     // Attends 1 seconde
}
```

Tout d'abord, on intègre la librairie « matrix » pour pouvoir utiliser les fonctions déclarées dedans.

La fonction **setup()** sert à initialiser tout notre programme une seule fois car elle ne s'exécute qu'au début du programme.

Enfin, la fonction **loop()** est répétée en boucle. Elle allume une LED avec différents paramètres renseignés, puis la matrice affiche les modifications et elle attend une seconde.

Cette ligne : **setPixel(x, y, color, intensity);** elle appelle la fonction setPixel pour allumer la LED à la position (x, y) avec **CRGB::Red;** qui définit la couleur de la LED et **128**; qui sert à définir l'intensité de la LED, ici à 128, soit 50% de la luminosité maximale. Puis le **delay(1000)**; c'est 1 seconde d'attente avant de changer de couleur.

Le fichier de bibliothèque : matrix.h

```h
#include <FastLED.h>

#define DATA_PIN    21        // Pin de données connectée à la matrice
#define NUM_LEDS    256     // Nombre total de LEDs (32 colonnes * 8 lignes)


void init();
void setPixel(int x, int y, CRGB color, int intensity);
void update();
```

Nous utilisons la librairie FastLED pour contrôler les LEDs dans ce programme. Ensuite nous définissons plusieurs paramètres :

DATA_PIN est une constante, elle définit la broche sur laquelle les données sont envoyées par le microcontrôleur pour contrôler les LEDs. On a choisi la broche 21 qui correspond à la pin digital 3 de notre ESP32 C6.

NUM_LEDS est le nombre total de LEDs dans la matrice. Dans le cas de notre projet, il y a 32 colonnes et 8 lignes, soit 256 LEDs au total.

Dans ce fichier on déclare également les fonctions en écrivant leurs protocoles.

Le fichier implémentation : matrix.cpp

```cpp
#include <matrix.h>

CRGB leds[NUM_LEDS];        // Déclare un tableau de LEDs


void init()
{
  FastLED.addLeds<WS2812, DATA_PIN, GRB>(leds, NUM_LEDS); // Utilisation des LEDs de la mareice WS2812
  FastLED.setBrightness(128); // Définit la luminosité qui est parametré à 50%
}

void setPixel(int x, int y, CRGB color, int intensity)
{
  int ledIndex;

  // Si la ligne (y) est paire, la numérotation des colonnes est de gauche à droite
  if (x % 2 == 0)
  {
    ledIndex = (x * 8) + y;  // Calcul classique
  }
  else
  {
    ledIndex = (x * 8) + (8 - 1 - y);  // Si la ligne est impaire, on inverse l'ordre des colonnes
  }

  leds[ledIndex] = color;  // Change la couleur de la LED à cet endroit
  leds[ledIndex].fadeToBlackBy(255 - intensity);  // Applique l'intensité en fonction de l'argument
}
```

On met en lien les déclarations du fichier .h en l'incluant au début du fichier .cpp.

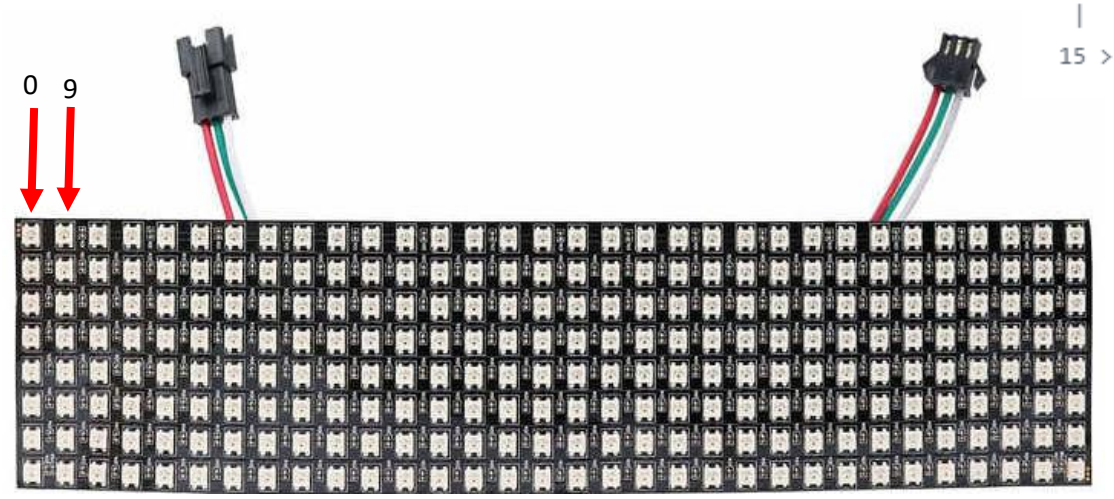
La ligne **leds[NUM_LEDS]** est un tableau où chaque élément correspond à une LED de la matrice. Il est de type CRGB pour pouvoir stocker les valeurs de couleur rouge, verte et bleue pour chacune des leds.

Grâce à la librairie FastLED nous pouvons passer aux initialisations dans la fonction init() avec **FastLED.addLeds<WS2812, DATA_PIN, GRB>** c'est la ligne qui initialise la matrice de LEDs WS2812 et les configure pour qu'elles puissent être contrôlées via la broche DATA_PIN qui est relié à l'ESP32 C6. Le format de couleur utilisé est GRB (Green, Red, Blue). Puis **FastLED.setBrightness(128) ;** c'est la commande qui définit la luminosité des LEDs à 50% car 128 sur 255 qui est la valeur maximale de la luminosité.

La fonction setPixel() permet de personnaliser avec des paramètres l'état d'une LED en fonction de sa position (x, y), de la couleur et de l'intensité programmée. Le code détermine l'index de la LED dans le tableau leds[] en fonction de sa position (x, y) dans la matrice. Si la colonne (x) est paire, les LEDs sont numérotées de haut en bas. Si la colonne (x) est impaire alors l'ordre des LEDs est inversé pour donner l'effet d'un "mouvement zigzag".

```
0 >  1 >  2 >  3 >  4
                    |
 .----<----<----<----'
 |
 5 >  6 >  7 >  8 >  9
                    |
 .----<----<----<----'
 |
10 > 11 > 12 > 13 > 14
                    |
 .----<----<----<----'
 |
15 > 16 > 17 > 18 > 19
```

Cette ligne de code **leds[ledIndex] = color** ; sert à changer la couleur de la led à la position calculée. Et **leds[ledIndex].fadeToBlackBy(255 - intensity)** Cette ligne applique l'intensité à la LED en ajustant sa luminosité.




```
30    void update()
31    {
32      FastLED.show(); //Affiher les modifications
33    }
```

La fonction **update()** avec cette commande **FastLED.show();** appelle la fonction pour appliquer les modifications de couleur à la matrice de LEDs.

Le code répète ce processus pour changer les couleurs de la LED entre rouge, bleu et vert, chaque fois avec une pause d'une seconde.

Pour finir cette librairie contrôle une matrice de 32x8 LEDs WS2812 et permet d'allumer une LED spécifique avec une couleur et une intensité donnée. Il change progressivement la couleur d'une LED à un emplacement fixe donné en paramètre toutes les secondes. La matrice utilise un "effet en zigzag" pour les colonnes impaires.

2. Librairie pour l'affichage caractères

Cette librairie utilise la bibliothèque FastLED et la librairie précédente pour contrôler une matrice de LEDs WS2812 et la librairie pixel. Il affiche des chiffres entre 0 à 9 sur la matrice de LEDs.

Le code principale (.ino)

C'est le fichier principal qui sera exécuter par l'ESP32 C6.

```
#include <matrix.h>
#include <caracteres.h>
#include <tableau.h>


void setup()
{
  init();
}


void loop()
{
  int yb = 2;  // Position Y pour la ligne à laquelle on commence
  int xb = 1;   // Position X pour la colonne à laquelle on commence

  const char* text = "12345678"; // Texte à afficher
  displayText(xb, yb, text, CRGB::Magenta, 150); // Affiche le texte en fonction des paramètres
  update();
  delay(5000);
  clear_ all();
}
```

La fonction **setup()** sert à l'initialisation et elle s'exécute une seule fois au démarrage du programme. Donc elle initialise la matrice et configure tous les paramètres.

La fonction **loop()** est considérée comme principale car elle tourne en boucle et permet de définir le pixel de début grâce au 'x' et au 'y'. On choisit également la chaine de caractère qu'on souhaite afficher puis on utilise la fonction qui affiche le texte avec les paramètres rentrés. Enfin la matrice se met à jour et prend en compte les modifications.

Grâce aux fonctions écrites dans un fichier de librairie le code est plus allégé.

La première librairie : tableau (.h)

Dans le fichier .h il y a toutes les déclarations de fonctions pour le bon fonctionnement du capteur, grâce au prototype pour les différentes configurations.

```
#ifndef TABLEAU_H
#define TABLEAU_H

extern char chiffre[128][5];  // Déclaration du tableau sans l'initialiser

#endif
```

Dans ce fichier bibliothèque on déclare notre tableau de caractère qui ne serait défini qu'une seul fois.

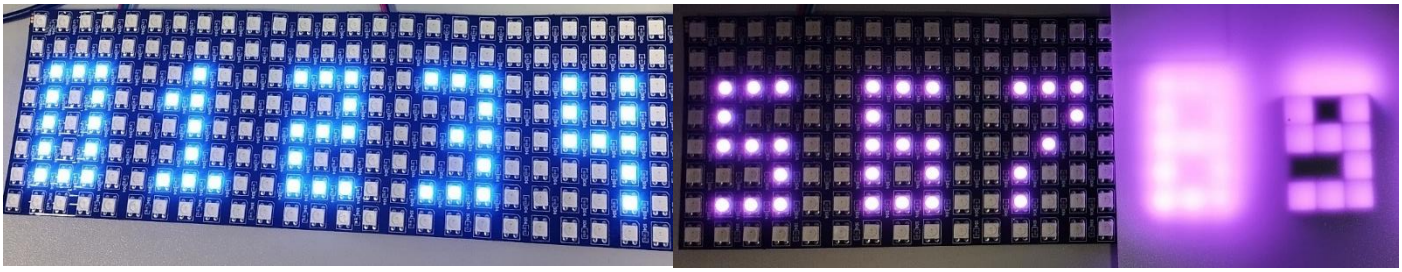
Le fichier implémentation : tableau (.cpp)

```
1   #include "tableau.h"
2
3   char chiffre[128][5] =
4   {
5       {0,0,0,0,0}, // ASCII 0
6       {0,0,0,0,0}, // ASCII 1
```

```
50      {0b000, 0b000, 0b011, 0b000, 0b000}, // ASCII 45('-')
51      {0b000, 0b000, 0b000, 0b000, 0b010}, // ASCII 46('.')
52      {0,0,0,0,0}, // ASCII 47
53      {0b111, 0b101, 0b101, 0b101, 0b111}, // ASCII 48 ('0')
54      {0b010, 0b110, 0b010, 0b010, 0b111}, // ASCII 49 ('1')
55      {0b111, 0b001, 0b111, 0b100, 0b111}, // ASCII 50 ('2')
56      {0b111, 0b001, 0b011, 0b001, 0b111}, // ASCII 51 ('3')
57      {0b101, 0b101, 0b111, 0b001, 0b001}, // ASCII 52 ('4')
58      {0b111, 0b100, 0b111, 0b001, 0b111}, // ASCII 53 ('5')
59      {0b111, 0b100, 0b111, 0b101, 0b111}, // ASCII 54 ('6')
60      {0b111, 0b001, 0b010, 0b100, 0b100}, // ASCII 55 ('7')
61      {0b111, 0b101, 0b111, 0b101, 0b111}, // ASCII 56 ('8')
62      {0b111, 0b101, 0b111, 0b001, 0b111}, // ASCII 57 ('9')
63      {0b000, 0b010, 0b000, 0b010, 0b000}, // ASCII 58 (':')
64      {0b010, 0b000, 0b000, 0b000, 0b000}, // ASCII 59 (';')
```

```
70      {0b111, 0b101, 0b111, 0b101, 0b101}, // ASCII 65 ('A')
71      {0b110, 0b101, 0b110, 0b101, 0b110}, // ASCII 66 ('B')
72      {0b111, 0b100, 0b100, 0b100, 0b111}, // ASCII 67 ('C')
73      {0b110, 0b101, 0b101, 0b101, 0b110}, // ASCII 68 ('D')
74      {0b111, 0b100, 0b110, 0b100, 0b111}, // ASCII 69 ('E')
75      {0b111, 0b100, 0b110, 0b100, 0b100}, // ASCII 70 ('F')
76      {0b111, 0b100, 0b100, 0b101, 0b111}, // ASCII 71 ('G')
77      {0b101, 0b101, 0b111, 0b101, 0b101}, // ASCII 72 ('H')
78      {0b111, 0b010, 0b010, 0b010, 0b111}, // ASCII 73 ('I')
79      {0b111, 0b001, 0b001, 0b101, 0b011}, // ASCII 74 ('J')
80      {0b101, 0b101, 0b110, 0b101, 0b101}, // ASCII 75 ('K')
81      {0b100, 0b100, 0b100, 0b100, 0b111}, // ASCII 76 ('L')
82      {0b101, 0b111, 0b101, 0b101, 0b101}, // ASCII 77 ('M')
83      {0b101, 0b101, 0b111, 0b101, 0b101}, // ASCII 78 ('N')
84      {0b111, 0b101, 0b101, 0b101, 0b111}, // ASCII 79 ('O')
85      {0b111, 0b101, 0b111, 0b100, 0b100}, // ASCII 80 ('P')
86      {0b111, 0b101, 0b101, 0b111, 0b010}, // ASCII 81 ('Q')
87      {0b111, 0b101, 0b110, 0b101, 0b101}, // ASCII 82 ('R')
88      {0b111, 0b100, 0b111, 0b001, 0b111}, // ASCII 83 ('S')
89      {0b111, 0b010, 0b010, 0b010, 0b010}, // ASCII 84 ('T')
90      {0b101, 0b101, 0b101, 0b101, 0b111}, // ASCII 85 ('U')
91      {0b101, 0b101, 0b101, 0b101, 0b010}, // ASCII 86 ('V')
92      {0b101, 0b101, 0b101, 0b111, 0b101}, // ASCII 87 ('W')
93      {0b101, 0b101, 0b010, 0b101, 0b101}, // ASCII 88 ('X')
94      {0b101, 0b101, 0b010, 0b010, 0b010}, // ASCII 89 ('Y')
95      {0b111, 0b001, 0b010, 0b100, 0b111}, // ASCII 90 ('Z')
```

Ces lignes définissent une partie du tableau avec tous les chiffres, lettres et certains symboles en utilisant des tableaux de bits. Chaque caractère est représenté par une matrice de 5 lignes et 3 colonnes de LEDs, où chaque bit représente une LED allumée avec un NL1 ou éteinte avec un NL0. Ce tableau c'est comme une représentation de la table ASCII.



Exemple : CHIFFRE_0 est représenté par les lignes suivantes :

0b111 (111 en binaire, cela signifie que toutes les LEDs sur la première ligne sont allumées)
0b101 (101 en binaire, la première et la troisième LED sont allumées sur la deuxième ligne)
0b101 (même chose que la ligne précédente)
0b101 (même chose que la ligne précédente)
0b111 (toutes les LEDs sont allumées à nouveau)

Pour chaque chiffre un tableau comme celui-ci est créé pour former le chiffre associé.

La deuxième librairie : matrix. Elle est composée comme les autres d'un fichier .h et .cpp pour la déclaration des fonctions puis pour définir l'intégralité des fonctions. Vous la retrouverez ci-dessus dans la partie précédente.

La troisième librairie : caracteres (.h)

```h
#include <matrix.h>
#include <tableau.h>

void displayText(int startX, int startY, const char text[], CRGB color, int intensity);

void drawCaractere(int x, int y, char caractere, CRGB color, int intensity);
```

Nous retrouvons les déclarations de fonction pour les affichages en parcourant le tableau du caractère choisi.

Le fichier caracteres .cpp

```cpp
#include <caracteres.h>

// Fonction pour ecrire un caractère ASCII sur la matrice LED
void drawCaractere(int x, int y, char caractere, CRGB color, int intensity)
{
  for (int row = 0; row < 5; row++) // chaque caractère est constitué de 5 lignes
  {
    int lineValue = chiffre[(int)caractere][row]; // Obtenir la ligne correspondante dans le tableau du caractère sélectionné
    for (int col = 0; col < 3; col++) // Chaque ligne contient 3 colonnes
    {
      if ((lineValue >> (2 - col)) & 1) // On se décale et on masque pour vérifier le bit sélectionné
      {
        setPixel(x + col, y + row, color, intensity); // On allume la LED correspondante
      }
    }
  }
  update(); // Affiche les modifications
}
```

La fonction **drawCaractere()** dessine un chiffre à une position donnée (x, y) sur la matrice de LEDs. Elle prend en paramètre un tableau **chiffre[]** qui contient les informations du tableau sur le caractère à afficher,

et allume les LEDs correspondantes en fonction des données. Le chiffre est composé de 5 lignes, et chaque ligne a 3 colonnes. Pour chaque ligne du chiffre, on vérifie chaque bit avec un masque. Si le bit est au NL1 on allume la LED correspondante.

setPixel(x + col, y + row, color, intensity) ; Cette fonction est appelée pour allumer la LED à la position demandée avec la couleur et l'intensité spécifiées.

FastLED.show() ; est appelé par la fonction **update()** pour appliquer les changements.

```
20  void displayText(int startX, int startY, const char text[], CRGB color, int intensity)
21  {
22    const int charWidth = 3;
23    const int spacing = 1;
24    for (int i = 0; text[i] != '\0'; i++) // Parcourt chaque caractère du texte
25    {
26      int x = startX + i * (charWidth + spacing);
27      drawCaractere(x, startY, text[i], color, intensity); // Dessine le caractère à la position calculée
28    }
29  }
```

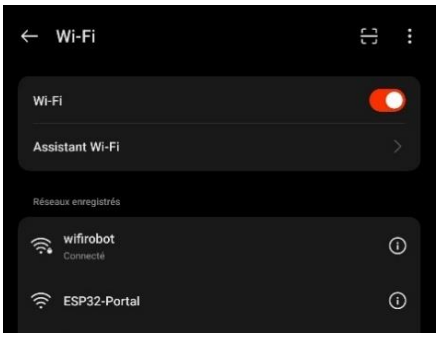
La fonction **displayText()** affiche une séquence de chiffres sur la matrice, un par un. Chaque chiffre est placé et dessiné à une position calculée en fonction de la largeur, de l'indice du chiffre et de l'espacement défini entre les chiffres.

Pour commencer le tableau **text[]** contient les différents chiffres à afficher. Ensuite, la fonction **drawCaractere()** est appelée pour afficher chaque chiffre à la position appropriée.

Ce code Arduino contient tous les tableaux des chiffres et permet un affichage de certains chiffres choisis pour ensuite les afficher sur la matrice après avoir rentré différents paramètres.

3. Gestion réseau & Portail captif

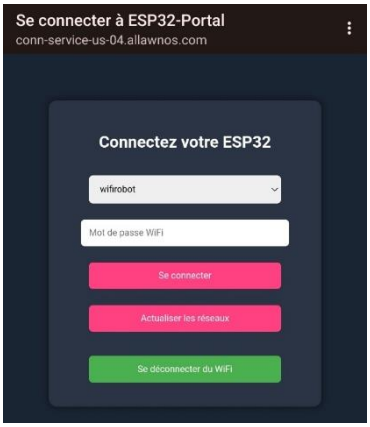
Pour commencer, nous avions besoin de définir les besoins ainsi que les fonctionnalités que nous voulions sur ce portail captif. Le but principal était de pouvoir connecter l'ESP32 à un réseau wifi en passant par un portail captif. Nous voulions une interface facile d'utilisation, une reconnexion automatique au réseau wifi si le réseau avait déjà été enregistré et une page qui affiche l'état de la connexion.



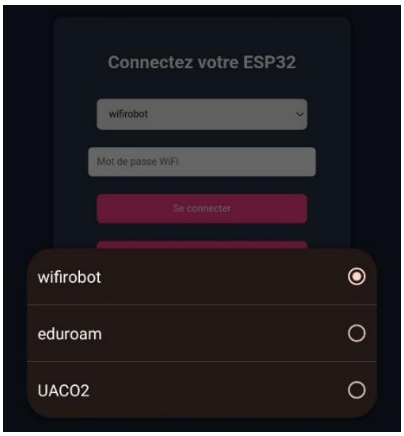
La première étape était de créer un point d'accès Wifi avec l'ESP32 et de pouvoir s'y connecter avec son téléphone à l'aide un mot de passe.

La deuxième partie permet de se connecter en wifi à l'ESP32 avec son téléphone sans mot de passe. Ensuite, il faut ouvrir un navigateur et rentrer l'adresse 192.168.4.1 puis on arrive sur une page WEB où il faut saisir manuellement le nom du réseau Wifi ainsi que son mot de passe. Une fois ceux-ci rentrés, l'ESP32 se connecte au réseau indiqué et on peut observer dans le moniteur série si la connexion est réussie avec le réseau Wi-Fi.

La troisième version du portail captif rajoute une seconde page WEB. Une fois le nom du réseau wifi et le mot de passe rentrés, on est redirigé sur cette seconde page WEB qui nous affiche l'état de la connexion de l'ESP32 au Wi-Fi avec un texte et une pastille de couleur. Cette pastille est verte lorsqu'il est connecté, orange quand il est en cours de connexion et rouge s'il n'a pas réussi à se connecter. En plus de ces 3 pastilles, un texte décrit l'état de la connexion. Nous avons le nom du réseau wifi, le mot de passe, le nombre de tentatives, l'état de la connexion puis l'adresse IP qui sont affichés dans le moniteur série.



Le but de la quatrième étape était d'avoir une interface plus attrayante. Nous avons décidé d'opter pour des couleurs comme le bleu ou le violet. Le texte disant « connectez votre ESP32 » est en haut, les champs pour rentrer le réseau Wi-Fi ainsi que le mot de passe sont blanc, et le bouton pour se connecter est en rose. Une fois redirigé sur la seconde page, nous avons le même style d'interface, en gardant la pastille de couleur ainsi que le texte qui évolue en fonction de l'état de la connexion. Nous avons toujours la communication avec le moniteur série.



La cinquième partie nous permet d'être redirigé directement sur la page WEB lorsque nous nous connectons à l'ESP32. Nous gardons la même interface graphique avec le même fonctionnement ainsi que la communication avec le moniteur série.

La septième partie crée une liste déroulante sur la première page WEB. Le fonctionnement reste le même, mais nous n'avons plus besoin de rentrer le nom du réseau Wifi manuellement. Une liste déroulante apparaît et nous permet de choisir parmi tous les réseaux disponibles. Au début, nous avions certains réseaux Wi-Fi qui apparaissaient plusieurs fois dans cette liste. Nous avons donc dû mettre en place un filtre afin que chaque réseau Wi-Fi ne s'affiche qu'une seule fois.

Pour la huitième partie, nous avons fait en sorte que, la première fois que l'on connecte l'ESP32 à un réseau Wifi, il sauvegarde le nom du réseau ainsi que son mot de passe dans l'EEPROM afin de pouvoir se reconnecter automatiquement si l'ESP32 vient à être débranché puis rebranché ou redémarré. Ce code s'est divisé en deux parties. La première où nous devions enregistrer le nom du réseau et le mot de passe sur l'EEPROM. Ensuite, nous devions faire en sorte de détecter le moment où l'ESP32 est redémarré ou rebranché, puis de lancer une connexion automatique au dernier réseau wifi enregistré.

La neuvième partie permet d'actualiser la liste déroulante régulièrement afin de détecter si de nouveaux réseaux Wi-Fi apparaissent ou disparaissent. Cela évite de relancer le programme ou de devoir redémarrer l'ESP32 pour se connecter à un nouveau réseau wifi. Le fonctionnement reste le même et nous avons toujours la communication avec le moniteur série.

La dixième partie a permis de créer une fonction qui réinitialise l'EEPROM. Elle permet d'être intégrée dans le code final si nous voulons réinitialiser l'EEPROM à un moment précis. Le fonctionnement reste le même et nous avons toujours la communication avec le moniteur série.

La dernière partie rajoute plusieurs fonctionnalités. Premièrement, un bouton pour se déconnecter du réseau Wi-Fi actuel. Une fois déconnecté, l'EEPROM est réinitialisée pour éviter de se reconnecter immédiatement après la déconnexion. Deuxièmement, une fois connecté au réseau Wifi, un PING est envoyé à Google pour vérifier la connexion à internet. Si le PING est réussi, un message s'affiche pour nous avertir que l'ESP32 est bien connecté au Wifi ainsi qu'à Internet. Le test de la connexion au Wi-Fi et le test de la connexion à Internet sont indépendants. On peut donc avoir une connexion réussie au réseau Wi-Fi, mais pas de connexion à Internet. Dans ce cas, le message indique que la connexion au Wi-Fi est réussie, mais que l'ESP32 n'est pas connecté à Internet. Le dernier ajout est un bouton qui permet de mettre à jour manuellement les réseaux Wi-Fi disponibles. Le fonctionnement reste le même et nous avons toujours la communication avec le moniteur série. Communication avec le moniteur série :

```
09:10:25.995 -> Point d'accès activé : ESP32-Portal
09:10:26.996 -> ......
09:10:31.980 -> Connecté !
09:10:32.027 -> Serveur Web démarré.
09:15:34.796 -> Tentative de connexion à : wifirobot
09:15:34.796 -> Mot de passe reçu : robot2004LARIS
09:15:35.841 -> ........
09:15:42.838 -> Connecté !
09:15:42.838 -> Adresse IP : 192.168.10.49
09:15:53.990 -> Ping réussi : accès à Internet confirmé.
```

4. Synchronisation de l'heure via NTP

Le serveur NTP (Network Time Protocol) est un système qui permet de synchroniser précisément les horloges des ordinateurs et autres appareils sur un réseau. Il utilise des serveurs de temps de référence pour garantir une grande exactitude. En communiquant via internet ou un réseau local, le NTP ajuste les horloges des appareils pour qu'elles correspondent à l'heure de référence. Nous allons donc utiliser cette structure pour mettre à jour l'heure et la date sans avoir à régler manuellement l'horloge.

Puisque le NTP est un serveur, l'ESP32 doit être connecté à internet avant d'envoyer quoi que ce soit. Cette tâche est réalisée via le code fourni par Romain. Une fois la connexion Wi-Fi établie, la synchronisation de l'heure avec un serveur NTP peut être initiée. Nous avons choisi d'utiliser le serveur "pool.ntp.org", qui fournit une heure précise et fiable. Mais il existe un certain nombre d'autres serveurs qui fournissent le même service comme « time.google.com » ; « time.apple.com » ; « time.windows.com » ou encore « ntp.ubuntu.com », où chaque adresse est utilisée par les appareils des fabricants en question.

La fonction « configTime() » peut alors être appelée pour configurer le fuseau horaire et le serveur NTP. Après la configuration, la fonction « getLocalTime() » est lancée pour récupérer l'heure actuelle du serveur NTP. Le tout est compris dans la fonction « init_ntp() ».

```
// Connexion wifi... (Code de connexion Wi-Fi fourni par Romain)

Serial.println("WiFi connecté !");
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer); // Configure le fuseau horaire et le serveur NTP

struct tm timeinfo; // Déclare une structure tm pour stocker les informations de temps
if (!getLocalTime(&timeinfo)) { // Récupère l'heure du serveur NTP et la stocke dans timeinfo
    Serial.println("Échec de l'obtention de l'heure via NTP !");
    return false; // Retourne false en cas d'échec
}

currentTime = mktime(&timeinfo); // Convertit la structure tm en time_t et la stocke dans currentTime
Serial.println("NTP synchronisé !");
```

Les informations de temps sont stockées dans une « structure tm », disponible avec la bibliothèque « <time.h> », qui contient les détails de l'année, du mois, du jour, de l'heure, des minutes et des secondes. Pour stocker l'heure récupérée dans un format plus facile à manipuler, nous utilisons la fonction « mktime() » pour convertir la structure en une variable de « type time_t », qui représente le nombre de secondes écoulées depuis le 1er janvier 1970. Cette variable « currentTime » est ensuite utilisée par calcul pour la gestion de l'heure en temps réel. Après la synchronisation initiale avec le serveur NTP, la connexion Wi-Fi est désactivée pour économiser de l'énergie. Cela se fait avec l'appel de « disableWiFi() ». L'utilisation de la bibliothèque « Timezone » écrite par Jack Christensen permet le changement d'heure été/hiver, et est fonctionnelle car nous sommes passés en heure d'été sans problème lors de la rédaction de ce livrable.

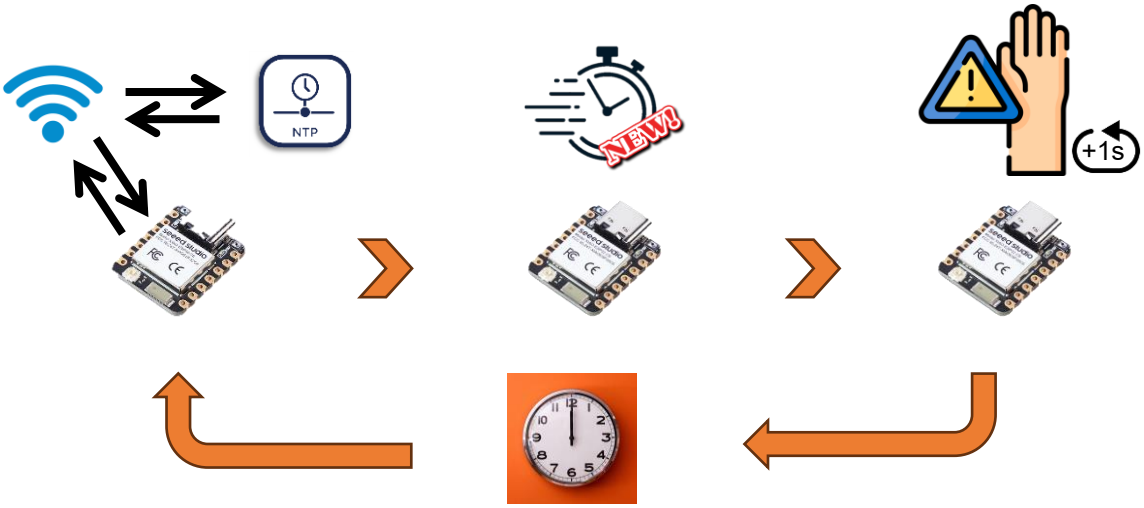
Solliciter constamment le serveur NTP pourrait entraîner une consommation excessive d'énergie et une surcharge du réseau, ce qui n'est pas le but recherché. Nous pourrions même nous faire bannir du serveur NTP pour cause de ping trop régulier. Pour éviter ce problème, nous avons mis en place un timer pour incrémenter l'heure. Ce dernier est configuré pour déclencher une interruption toutes les secondes, évitant ainsi de bloquer le programme principal avec un « delay() », permettant également de ne rien avoir dans le loop() pour le NTP. Il faut bien faire attention de vérifier si un timer existe déjà, et si c'est le cas, il faut l'arrêter avant d'en créer un nouveau. Auquel cas, après la resynchronisation à minuit détaillée après, l'incrémentation ira deux fois plus vite, puis trois fois plus vite le lendemain, puis quatre fois, etc. La solution pour éviter ce

problème est d'arrêter le timer en cours puis de le supprimer, et enfin d'en recréer un autre d'une façon « propre » pour s'assurer une bonne incrémentation. Toutes les 1s, l'interruption pointe donc « onTimerHour() » qui à son tour appelle « updateTime() » permettant d'incrémenter « currentTime ». Le choix d'avoir séparé en une autre fonction la variable de temps, au lieu de la mettre directement dans « onTimerHour() », se justifie par le souhait d'un code structuré et compréhensible.

```
// Si un timer existe déjà, on l'arrête avant de créer un nouveau timer
if (timerHandle != NULL) {
    esp_timer_stop(timerHandle); //Arrêt de l'ancien timer
    esp_timer_delete(timerHandle); //Suppression de l'ancien timer
}

// Configuration du timer pour incrémenter l'heure chaque seconde
const esp_timer_create_args_t timerArgs = {
    .callback = &onTimerHour, //Pointeur de la fonction à appeler ttes les 1s
    .arg = NULL,
    .dispatch_method = ESP_TIMER_TASK,
    .name = "ClockTimer" //Nom du timer
};

if (esp_timer_create(&timerArgs, &timerHandle) == ESP_OK) {
    esp_timer_start_periodic(timerHandle, 1000000); // Toutes les 1 seconde
    Serial.println("Timer démarré !");
    Serial.println(getHour());
    Serial.println(getDate());
} else {
    Serial.println("Erreur lors de la création du timer !");
    return false;
}
```

```
// Fonction appelée toutes les secondes pour incrémenter le temps
void updateTime(void* arg) {
    currentTime++; //incrémentation de +1 dans la variable
}
// Fonction exécutée par le timer toutes les secondes
void onTimerHour(void* arg) {
    updateTime(NULL);
}
```

En vue de s'assurer du bon fonctionnement du code, des tests ont donc été menés pour évaluer la précision de la synchronisation, en comparant l'heure affichée par notre horloge avec celle de nos téléphones. Ces expérimentations ont montré qu'aucune différence était présente, autrement dit, nous avions la même heure à la seconde près. Néanmoins, il est important de noter qu'une dérive de l'horloge interne de l'ESP-32 peut survenir à long terme, nécessitant une resynchronisation périodique pour maintenir une précision respectable, que nous avons définie à 24 heures. Le schéma suivant image la bibliothèque avec la synchronisation lorsque « getHour() » renvoie « 23:59:59 » :



getHour() ==
« 23:59:59 » ?

5. Intégration du capteur SHT45

Le code Arduino ci-dessous permet d'utiliser le capteur SHT45 d'Adafruit pour mesurer la température et l'humidité dans un environnement. Ce capteur a différentes précisions et permet de récupérer les deux données importantes pour analyser les conditions environnementales. Le programme configure le capteur dans un premier temps puis il ajuste les paramètres de précision et du chauffage, enfin il récupère les valeurs de température et d'humidité toutes les 3 secondes.

La structure du code se divise en trois fichiers distincts. Tout d'abord, le fichier .ino où il y a tout le code principal comme les initialisations dans la fonction setup() et la boucle principale dans le loop(). On inclut également toutes les bibliothèques nécessaires tout au début. Ensuite, le fichier .h contient toutes les déclarations de fonction avec leurs prototypes. Il inclut aussi les bibliothèques nécessaires pour le bon fonctionnement du code. Pour finir le fichier .cpp sert à définir toutes les fonctions avec tout leur contenu pour les initialisations, la configuration de la précision, l'activation du chauffage du capteur et la lecture de la température et l'humidité.

Le code principale (.ino)

C'est le fichier principal qui sera exécuté par l'ESP32 C6.

```
#include <capteur.h>

void setup()
{
  init_tests();      // Initialisation du capteur et de la communication série
  precision();       // Configure la précision de la mesure
  chauffage();       // Configuration du chauffage du capteur (optionnel)
}

void loop()
{
  temperature_humidite();  // Récupèré et affiché la température et l'humidité
  delay(3000);             // Attendre 3 secondes avant la prochaine mesure
}
```

La fonction **setup()** sert à l'initialisation et elle s'exécute une seule fois au démarrage du programme. Donc elle initialise le capteur et configure tous ces paramètres.

La fonction **loop()** est considérée comme principale car elle tourne en boucle et permet de récupérer les données de température et d'humidité toutes les trois secondes.

Grace aux fonctions écrites dans un fichier de librairie à côté, le code est beaucoup plus allégé.

Le fichier bibliothèque (.h)

Dans le fichier .h il y a toutes les déclarations de fonctions pour le bon fonctionnement du capteur, grâce aux prototypes pour les différentes configurations.

```
#include "Adafruit_SHT4x.h"  // bibliothèque pour le capteur SHT4x
#include <stdio.h>            // bibliothèque basique pour les entrées/sorties

void init_tests();          // prototype fonction d'initialisation
void precision();           // prototype fonction pour la précision du capteur
void chauffage();           // prototype fonction pour le chauffage du capteur
void temperature_humidite();// prototype fonction pour les données de température et d'humidité
```

La fonction **init_tests()** initialise la communication série le capteur SHT45, ici on voit seulement son prototype pour l'initialiser. Les deux prochaines fonctions **precision()** et **chauffage()** permettent de configurer les paramètres du capteur que ce soit la précision et le chauffage pour une meilleure performance. Enfin, le dernier prototype concerne la fonction de collecte des données.

Le fichier implémentation(.cpp)

```
#include <capteur.h>

Adafruit_SHT4x sht4 = Adafruit_SHT4x();

void init_tests(void)
{
  Serial.begin(115200); // Initialisation de la communication série
  while (!Serial) delay(10);  // Attente pour la connexion

  Serial.println("Test du capteur Adafruit SHT4x");

  if (! sht4.begin())  // Vérification de la connexion avec le capteur
  {
    Serial.println("Capteur SHT4x introuvable");
    while (1) delay(1);  // Arrêt si le capteur n'est pas trouvé
  }

  Serial.println("Capteur SHT4x trouvé");
  Serial.print("Numéro de série 0x");
  Serial.println(sht4.readSerial(), HEX);
}
```

La première fonction ici **init_tests(),** elle initialise la communication série de l'esp32 C6 et le capteur. Donc si le capteur n'est pas trouvé, le programme s'arrête, en revanche s'il est bien identifié alors on affiche également son numéro de série.

```
void precision(void)
{
  sht4.setPrecision(SHT4X_HIGH_PRECISION);  // Réglage de la précision
  switch (sht4.getPrecision())
  {
    case SHT4X_HIGH_PRECISION:
      Serial.println("Haute précision");
      break;
    case SHT4X_MED_PRECISION:
      Serial.println("Moyenne précision");
      break;
    case SHT4X_LOW_PRECISION:
      Serial.println("Petite précision");
      break;
  }
}
```

Ensuite la fonction **precision()** permet de configurer la précision de la mesure en fonction du besoin en rapport avec son application.

```
40  void chauffage(void)
41  {
42    sht4.setHeater(SHT4X_NO_HEATER);  // Pas de chauffage
43    switch (sht4.getHeater())
44    {
45      case SHT4X_NO_HEATER:
46        Serial.println("Pas de chauffage");
47        break;
48      case SHT4X_HIGH_HEATER_1S:
49        Serial.println("Chauffage élevé pendant 1 seconde");
50        break;
51    }
52  }
```

De plus, la fonction **chauffage()** sert à configurer l'état du chauffage. Ce qui peut améliorer ou non la précision des mesures par rapport à l'environnement et aux conditions dans lesquelles se trouve le capteur.

```
55  void temperature_humidite()
56  {
57    sensors_event_t humidity, temp;
58    sht4.getEvent(&humidity, &temp);  // Lecture de la valeurs de température et d'humidité
59
60    Serial.print("Température : ");
61    Serial.print(temp.temperature);
62    Serial.println(" °C");
63
64    Serial.print("Humidité : ");
65    Serial.print(humidity.relative_humidity);
66    Serial.println(" %");
67  }
```

Enfin, la fonction **temperature_humidite()** récupère les mesures de la température et de l'humidité de la pièce et les affiche dans le moniteur série (pour ensuite pouvoir être exploitées). La valeur de température est en Celsius et l'humidité est affichée en pourcentage.

Ce projet permet de récupérer de manière fiable les données de température et d'humidité grâce au capteur SHT45. Le code est structuré en plusieurs fichiers pour avoir une meilleure organisation et utilisation. Il peut être facilement adapté pour des applications nécessitant la surveillance en temps réel des conditions environnementales.

6. Gestion des interactions via les boutons poussoirs

Le code suivant gère la détection et les interruptions générées par l'appui sur un ou plusieurs boutons.

La bibliothèque Bouton permet de lire l'état d'un bouton et de le conserver en mémoire jusqu'à la prochaine mise à jour. Pour cela, on utilise une fonction lireBoutons() qui permet de modifier l'état d'une variable currentState afin de garder en mémoire l'état d'un bouton. La fonction isButtonXPressed() permet de détecter lorsqu'on appuie sur un bouton, et ainsi de mettre à jour la variable correspondante.

Enfin, la fonction etat_prece() permet de stocker, dans une variable lastState, l'état précédent d'un bouton, afin d'éviter les rebonds et de s'assurer qu'on détecte un seul front pour chaque appui.

7. Développement de la logique d'affichage

Le terme « logique d'affichage » fait référence à la structure et au déroulé de l'IHM (Interface Homme–Machine). Pour définir ce que l'utilisateur allait voir, nous nous sommes appuyés sur le cahier des charges, qui imposait l'affichage des éléments suivants :

- L'heure
- La date
- La température intérieure
- L'humidité intérieure
- Potentielle mise à jour : la météo & température extérieure via une API

Une fois les fonctions listées, nous nous sommes penchés sur la représentation graphique de chacune d'entre elles à l'aide d'un tableau Excel simulant l'affichage de la matrice.

Au démarrage, pendant la phase d'initialisation des composants et la récupération de l'heure, la matrice affiche une animation multicolore avec le texte « RGB CLOCK », nom du projet.

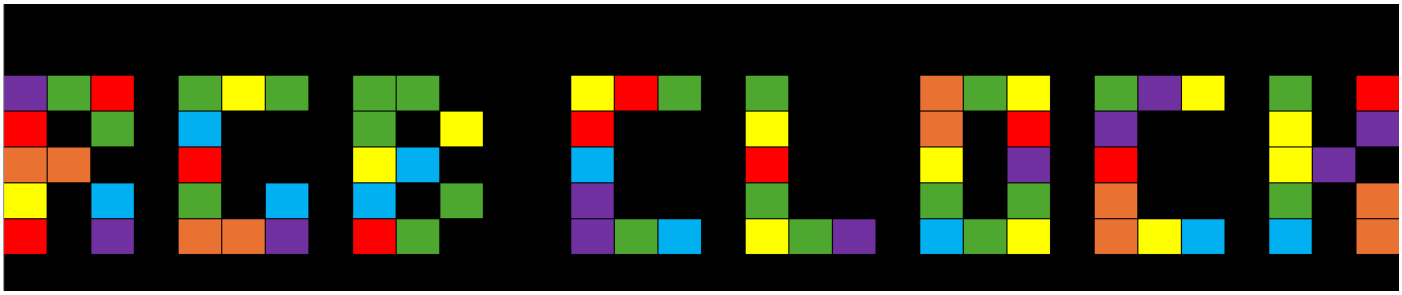


Illustration de l'animation au démarrage

L'animation dynamique repose sur l'utilisation de la fonction millis() qui permet de suivre le temps écoulé. À chaque pixel affiché, une teinte de couleur est calculée en fonction du temps, de la position du pixel, et d'un intervalle prédéfini. Cette teinte est ensuite convertie en une couleur CHSV (Teinte, Saturation, Valeur), créant un effet de changement de couleur fluide. La fonction setPixel() applique ensuite cette couleur dynamique à chaque LED. L'intervalle de temps contrôle la vitesse de l'animation. Pour afficher le nom du projet, il suffit d'appeler la fonction adéquate dans le loop.

```
void loop() {
  //Placement du premier pixel en haut - 2 pixels à gauche avec 200 d'intensité lumineuse
  displayTextMulticolorDynamic(0, 2, "RGB CLOCK", 200);
}
```

```
unsigned long previousMillis = 0; // Temps écoulé pour l'animation des couleurs
const long interval = 300; // Intervalle pour changer la couleur (en millisecondes)

void drawCaractereMulticolorDynamic(int x, int y, char caractere, int intensity, unsigned long currentTime)
{
  // Si le caractère est un espace, on ne dessine rien, mais on avance de 2 colonnes
  if (caractere == ' ') {
    return; // Ne pas dessiner le caractère, juste avancer
  }
  for (int row = 0; row < 5; row++) { // Chaque caractère est constitué de 5 lignes
    int lineValue = chiffre[(int)caractere][row]; // Obtenir la ligne correspondante dans le tableau du caractère sélectionné
    for (int col = 0; col < 3; col++) {// Chaque ligne contient 3 colonnes
      if ((lineValue >> (2 - col)) & 1) { // On se décale et on masque pour vérifier le bit sélectionné
        // Dynamiser la teinte en fonction du temps pour que la couleur change en temps réel
        int colorShift = (currentTime / interval + x + y + col + row) * 10; // Changer la couleur au fil du temps
        CRGB dynamicColor = CHSV(colorShift % 255, 255, 255); // Teinte changeante qui évolue avec le temps
        setPixel(x + col, y + row, dynamicColor, intensity); // Allume la LED correspondante avec une couleur dynamique
      }
    }
  }
}
```

Une fois le serveur NTP connecté et les données récupérées, l'heure et la date sont affichées sur la matrice. Les caractères utilisés sont de simples caractères ASCII. Nous avons donc pu réutiliser notre bibliothèque de caractères, en y ajoutant toutefois une logique spécifique pour les caractères « : » et « - ».

En effet, ces caractères provoquaient un espacement excessif dû à la logique standard de décalage entre les caractères. Pour corriger cela, nous avons modifié la fonction displayText() en y intégrant une variable dynamique. Celle-ci détecte les caractères spéciaux et ajuste l'espacement en conséquence : elle recule d'un pixel, supprime l'espace du caractère précédent, affiche le caractère, puis ajuste à nouveau la position x.

```
void displayText(int startX, int startY, const char text[], CRGB color, int intensity) {
  const int charWidth = 3; // Largeur d'un caractère
  const int spacing = 1; // Espace entre les caractères

  int x = startX; // Position de départ sur l'axe X
  for (int i = 0; text[i] != '\0'; i++) {
    if (text[i] == ' ') {
      x += 1; // Décalage pour l'espace
    } else {
      bool specialChar = (text[i] == ':' || text[i] == '-' || text[i] == '.'); //variable dynamique
      bool specialCharSpacing = (text[i] == ':' || text[i] == '.'); //variable dynamique
      if (specialChar) x -= 1; // Décalage pour les caractères spéciaux
      drawCaractere(x, startY, text[i], color, intensity); // Dessine le caractère
      x += charWidth + (specialCharSpacing ? 0 : spacing); // Ajuste x
    }
  }
}
```

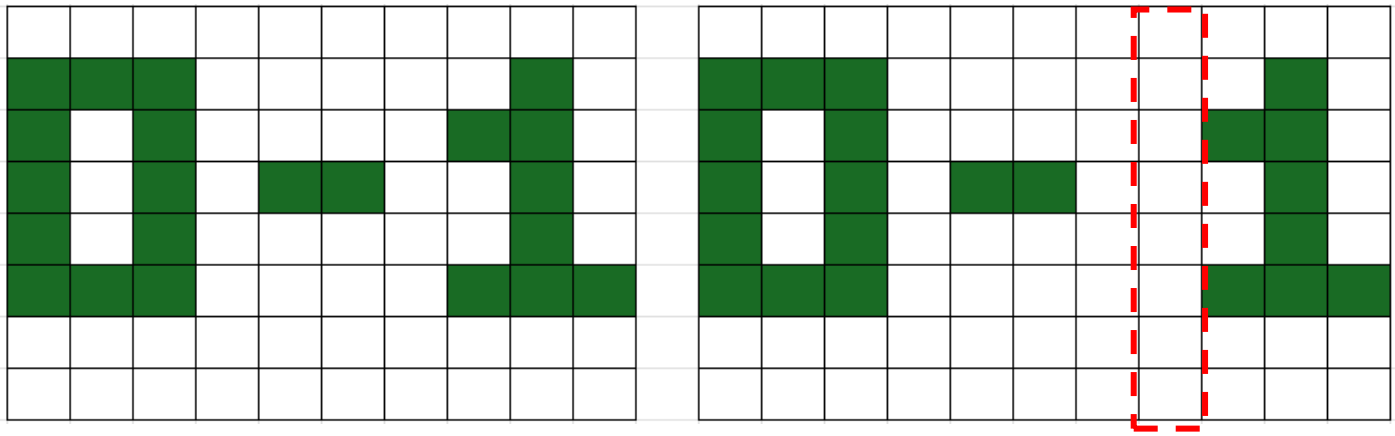


Illustration du problème des caractères spéciaux comme « - », avec et sans traitement spécial. La zone rouge représente l'espace en trop.

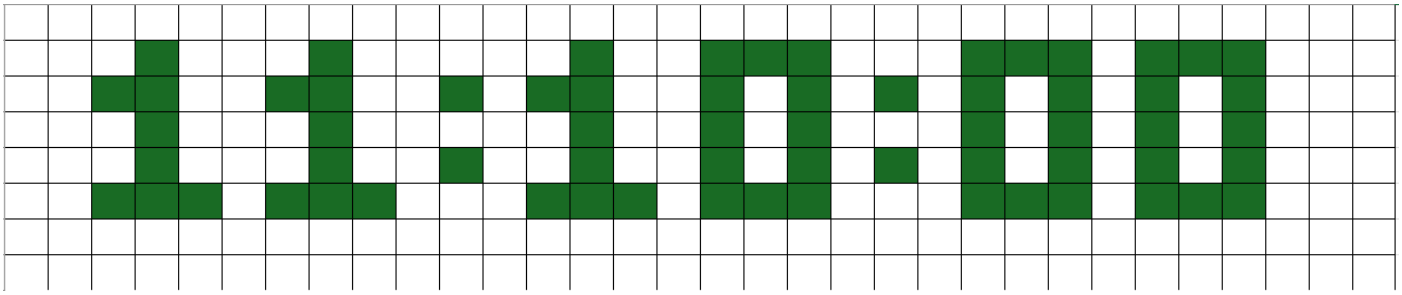


Illustration de l'affichage « Heure »

Nous avons choisi le caractère « : » pour séparer les heures et les minutes, car c'est le plus courant sur les horloges numériques.

Pour la date, nous avons opté pour un « – » (tiret), plus épuré qu'un « / », tout en restant lisible.

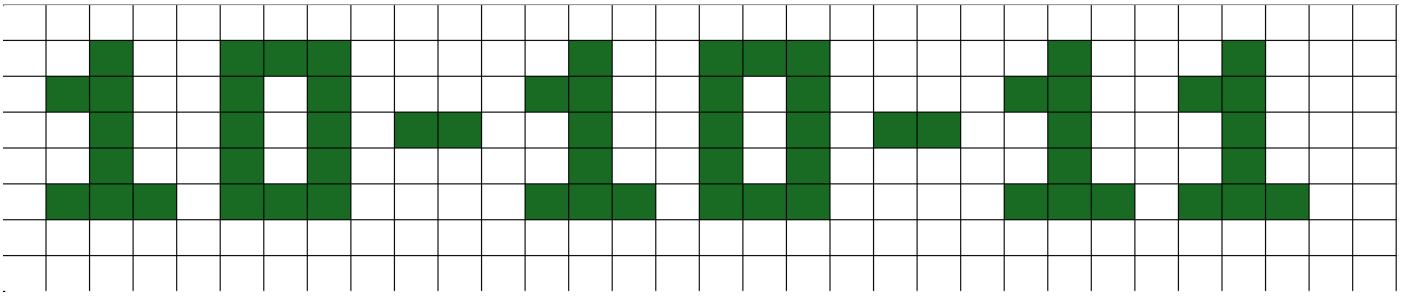


Illustration de l'affichage « Date »

L'espace limité sur la matrice ne permet pas d'afficher des icônes pour toutes les données. Cependant, pour la température et l'humidité, nous avons pu ajouter un petit pictogramme devant chaque valeur, améliorant ainsi la lisibilité et la compréhension des données affichées. Le symbole de température utilise les caractères « ° » et « C », déjà présents dans la table ASCII. Le caractère « % », utilisé pour l'humidité, a été dessiné en pixels 3×5. Bien qu'il soit moins esthétique, il reste fonctionnel. À terme, il pourrait être remplacé par un icône dédié.

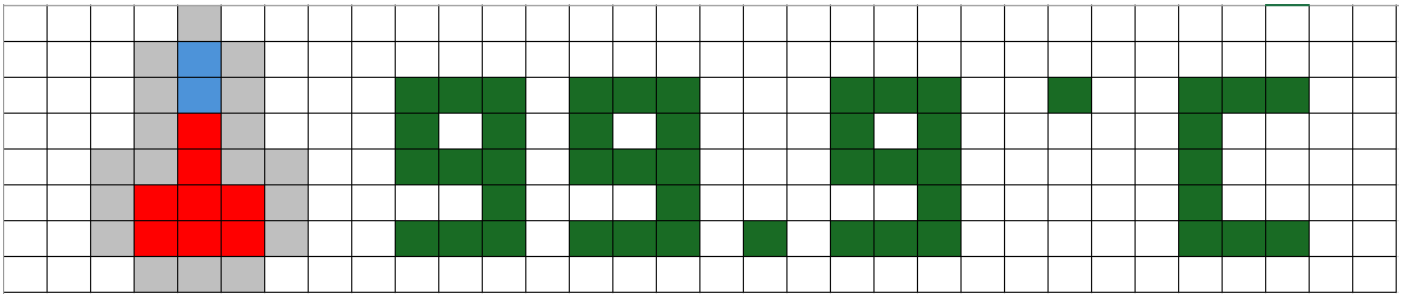


Illustration de l'affichage « Température »

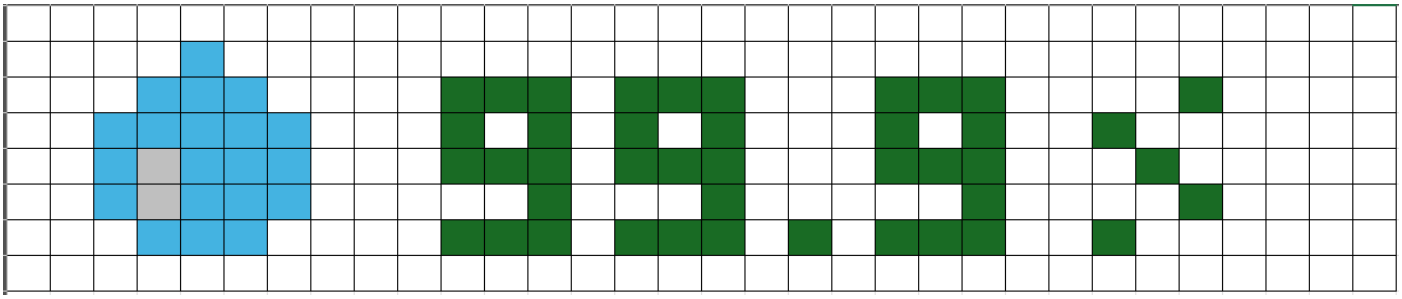
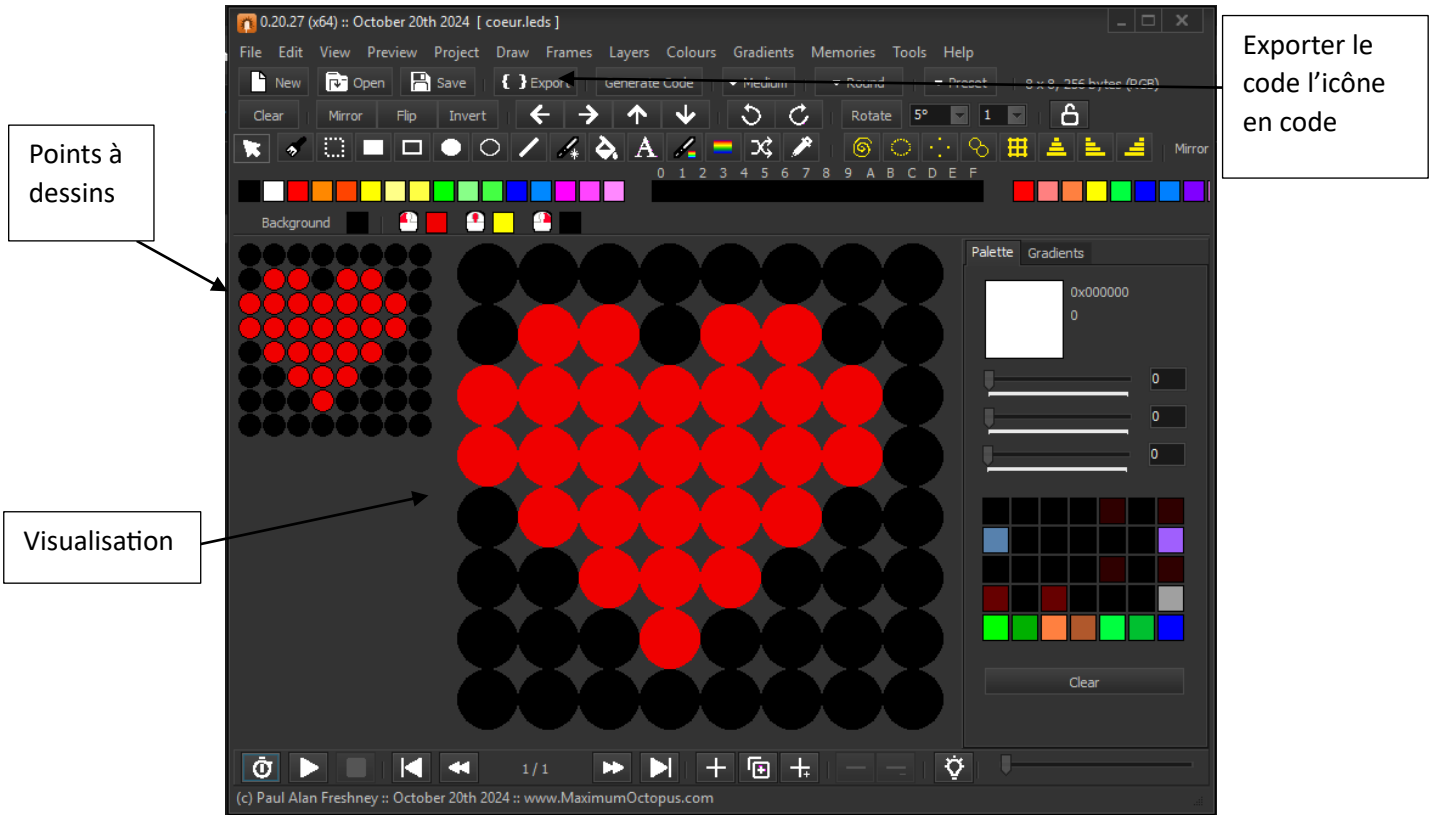


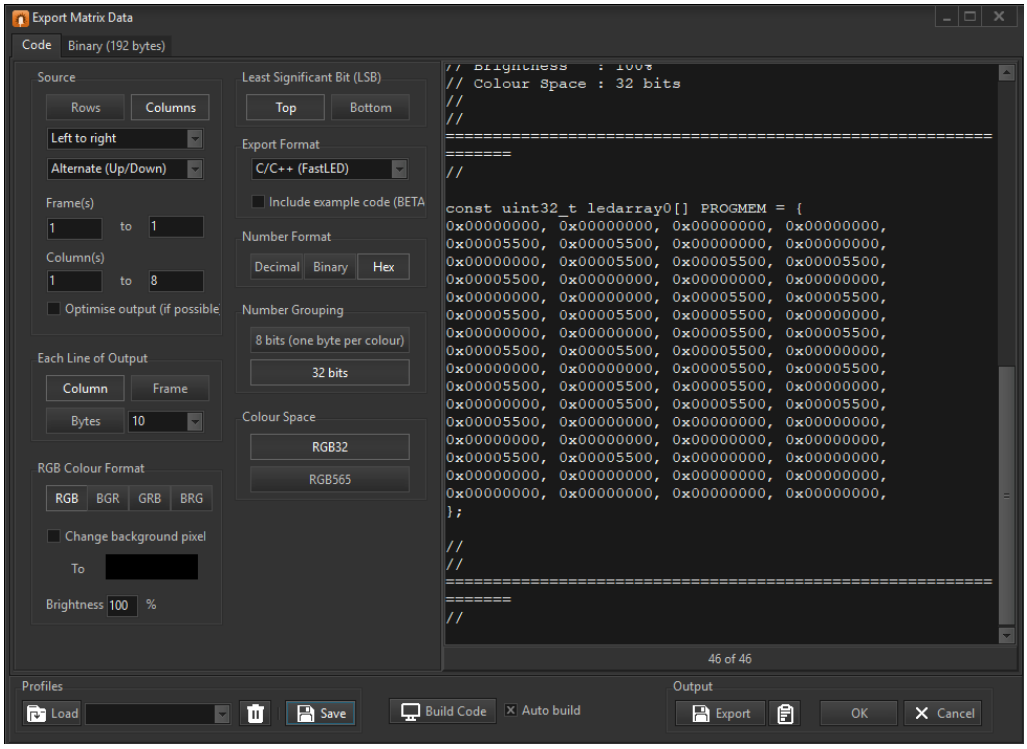
Illustration de l'affichage « humidité »

L'ajout d'icônes s'est avéré être un véritable atout visuel et ergonomique. Pour cela, nous avons utilisé un outil gratuit et portable nommé LEDMatrixStudio (disponible sur GitHub : https://github.com/MaximumOctopus/LEDMatrixStudio).

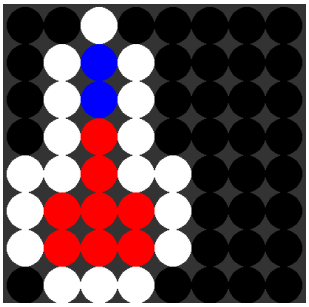
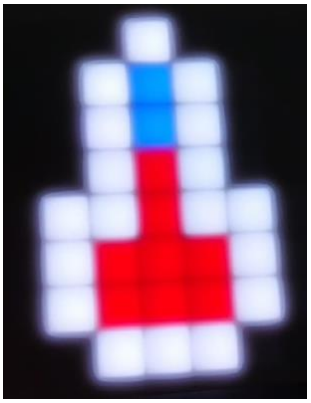
L'interface est intuitive et facile à prendre en main, elle se compose principalement des éléments suivants :

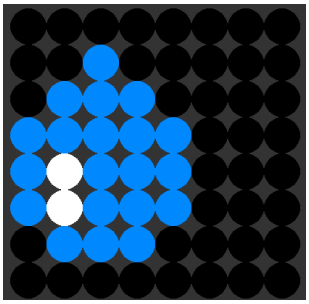
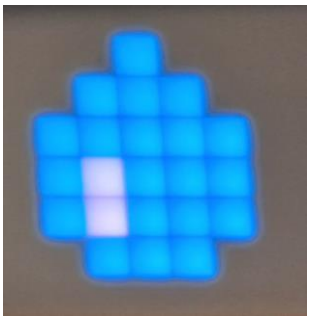
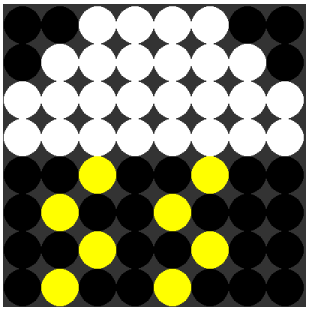
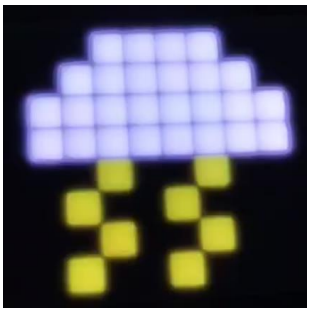
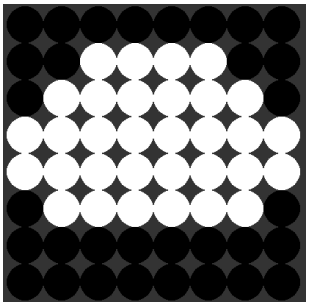
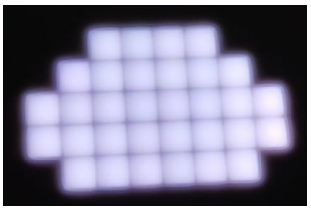
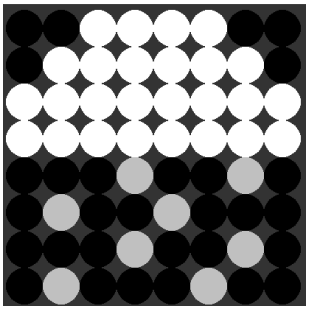
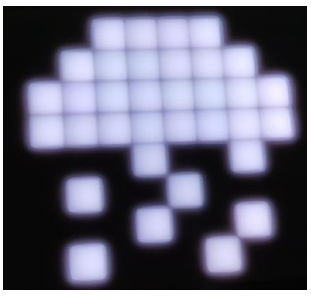


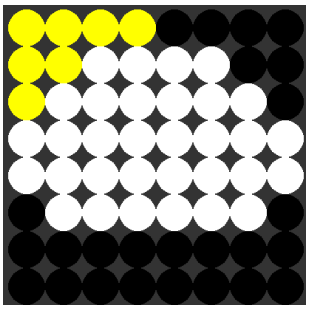
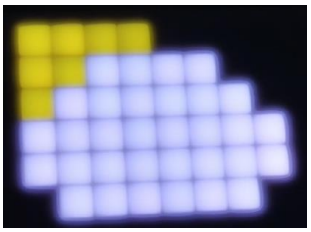
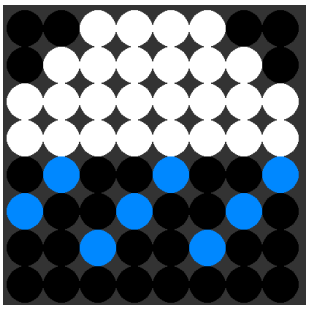
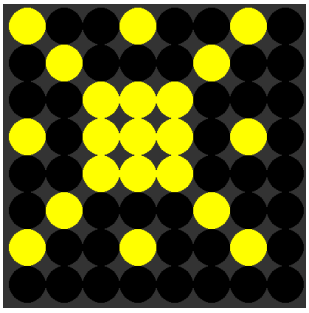
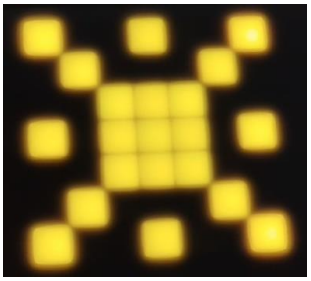
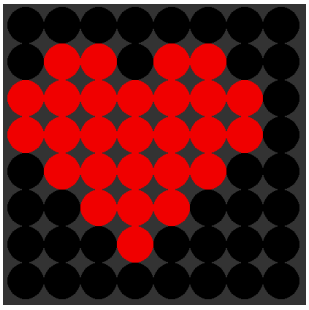
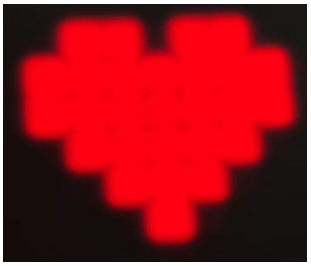
Pour réaliser une exportation en code, il faut connaitre les détails caractéristiques de notre matrix vu précédemment comme le type d'enchainement des pixels, si on a une alternance haut/bas etc. Pour notre matrice, les paramètres suivants ont été définis pour récupérer le tableau correspondant à notre matériel.



Une fois l'outil maîtrisé, seule notre imagination représentait une limite. Nous avons ainsi conçu plusieurs icônes personnalisées : pour la température et l'humidité dans un premier temps, puis pour des événements particuliers comme la Saint-Valentin. Le tableau ci-dessous présente un récapitulatif des icônes créées et leur rendu sur la matrice LED :

Simulation sur LEDMatrixStudio	Résultat sur la matrice
Icone thermomètre présent dans « showTemp() ».	
	
Icone goutte d'eau présent dans « showHumidity() ».	

	
Icone orage – pour l'amélioration « showMeteo ».	
	
Icone nuage – pour l'amélioration « showMeteo ».	
	
Icone neige – pour l'amélioration « showMeteo ».	
	
Icone ciel nuageux avec du soleil – pour l'amélioration « showMeteo ».	

	
Icone pluie (toutes les sortes de pluie) – pour l'amélioration « showMeteo ».	
	
Icone soleil – pour l'amélioration « showMeteo ».	
	
Icone cœur – pour l'amélioration « showEvent », servant d'indication à l'utilisateur pour des jours spéciaux, comme ici, le 14 février, la St Valentin.	
	

Pour faciliter l'utilisation de la bibliothèque dans le code principal, nous avons développé quatre fonctions qui prennent en paramètre le texte à afficher ainsi que la couleur et effectuent les opérations suivantes :

- « showDate() » :
  - Efface la matrice
  - Utilise la fonction « displayText() » pour afficher le texte

    - Met à jour la matrice

- « showTemp() » & « showHumidity() » :
    - Effacent la matrice
    - Affichent l'icône associée vue ci-dessus
    - Utilisent la fonction « displayText() » pour afficher le texte
    - Mettent à jour la matrice

- « showHeure() » : utiliser le même principe que « showDate() » aurait pu fonctionner. Cependant, à chaque appel pour réactualiser les secondes, l'utilisateur percevrait un scintillement désagréable dû à l'effacement total de la matrice puis au réaffichage de tous les caractères. Pour pallier ce problème, nous avons déclaré dans un tableau les coordonnées de chaque caractère. En plus de l'heure et de la couleur, la fonction demande également l'heure précédente, ce qui permet de comparer les caractères qui ont changé pour « effacer » celui en question et réafficher le nouveau aux coordonnées correspondantes. L'effacement s'effectue par la localisation du chiffre, puis on applique la couleur noire (LED éteinte) à chaque pixel.

Le développement de la logique d'affichage nous a permis de nous projeter significativement dans l'interface utilisateur en choisissant la position parfaite du logo ainsi que les caractères (en x et y de la matrice). Concernant les icônes des fonctions « showMeteo » et « showEvent », elles restent pour le moment une potentielle mise à jour et non le cahier des charges établi au début.

8. Intégration et validation du code complet

Développer le code principal est, par définition, d'harmoniser toutes les bibliothèques écrites par le groupe. Pour réaliser cela, et dans le but de faciliter l'écriture, chacun a inscrit le nom de ses fonctions, de ceux qu'elles demandaient en paramètre, et ce qu'elles retournaient.

Plusieurs architectures ont été écrites ; la première a été du code en « polling ». Ce terme désigne le fait qu'au lieu d'attendre passivement qu'un événement se produise, le programme vérifie de manière répétée l'état d'une ressource pour voir si une action est nécessaire. Par exemple, vous attendez que votre linge soit sec. Avec des interruptions, le sèche-linge vous enverrait un signal quand le linge est sec. Vous pourriez faire autre chose en attendant et seriez averti seulement quand c'est prêt. Alors qu'avec du polling, vous iriez vérifier le sèche-linge toutes les 5 minutes pour voir si le linge est sec. Vous devez interrompre ce que vous faites pour aller vérifier, même si le linge n'est pas encore prêt. Cette structure de code que nous avions faite fonctionnait, mais elle n'était pas du tout optimale. Nous avons donc abandonné le « polling » pour utiliser les interruptions logicielles, déjà utilisées pour incrémenter d'une seconde le compteur de l'heure.

L'interruption utilisée est basée sur un timer matériel (ISR) de 50ms qui n'a pas été choisie au hasard. L'effet arc-en-ciel de « RGB CLOCK » qui se lance dès l'alimentation de l'horloge change de couleur toutes les 50ms, de plus, c'est une valeur permettant d'atteindre exactement 1s tout comme 1min. Une machine à états est également présente pour fractionner les différents états de l'horloge, l'ISR de 50ms est aussi là pour changer l'état de la structure. Concernant cette dernière, deux fonctions ont été codées pour faciliter le changement d'état en avant ou en arrière (les boutons permettant de revenir dans l'état d'affichage précédent) : les deux fonctions récupèrent l'état en cours et appellent la fonction de changement d'état qui modifie simplement la variable de l'état en cours avec l'état suivant ou précédent.

Dans le but de comprendre ce que fait le code principal, nous avons séparé chaque élément pour réaliser le résumé suivant :

1 - setup :
=> Serial à 115200

=> Initialisation du timer de 50ms

2 - loop :
=> si heure = '23:59:59' => synchronisation NTP

Dans tous les programmes Arduino, nous avons une partie setup correspondant aux initialisations de tous les périphériques et une partie loop où le programme principal est réalisé en boucle.

Dans notre programme, le setup ne comprend que l'initialisation du port série à 115200 bauds ainsi que l'initialisation du timer de 50ms décrit plus haut. Ce dernier est créé à l'aide de la bibliothèque « esp_timer.h ». Le loop est également léger et ne fait qu'une action en boucle en polling : l'appel de la fonction de synchronisation du NTP lorsqu'on change de jour.

Ensuite, on a les interruptions qui représentent le pilote du programme en appelant la fonction qui passe à l'état suivant. Lorsque l'état en cours est Date, alors l'interruption demande « std::pair<float, float> temperature_humidite() » pour mettre à jour la température et l'humidité. Autrement dit, cette mise à jour survient toutes les 1min 15s.

Pour utiliser des variables de temps dans les interruptions, il est essentiel de les déclarer avec le mot-clé « volatile ».

3 – interruption logicielle – toutes les 50ms

=> Changement de couleur de l'arc-en-ciel si état = INIT

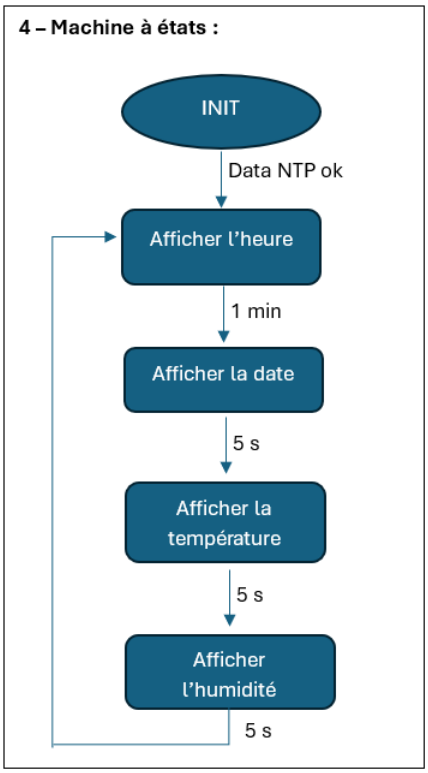
=> Actions de la machine à état

=> 5 secondes passées ? → Changement d'état

=> 1 minute passée & heure ? → Changement d'état

=> Etat = Date ? → mise à jour de la température et de l'humidité

4 – Machine à états :



Le 3e élement principal du code est la machine à état. Ce dernière est composé de 5 états répartie dans un structure case :

- « INIT » servant d'initilisation à toutes variables, fonctions, périphériques et paramètres du projet soit le setup du programme. On passe à l'état suivant si la requete NTP est finie et valide.
- « SHOW_HOUR » pour afficher l'heure grâce à la fonction « showHeure() » présentée dans la section 7. Développement de la logique d'affichage. Rester 1min sur cet état semble correcte puisque c'est avant tout une horloge, l'heure doit donc être visible plus longtemps que le reste.
- « SHOW_DATE » pour afficher la date grâce à la fonction « showDate » présentée dans la section 7. Développement de la logique d'affichage.

- « SHOW_TEMP » et « SHOW_HUMIDITY » appellent respectivement « showTemp() » et « showHumidity() » en testant si les variables des données sont suppérieur à 0. Si ce n'est pas le cas, ça veut dire que le capteur est en erreur on affiche donc trois points de suspension « … » pour informer l'utilisateur sans bloquer le programme.

Pour les trois derniers états, une condition est présente pour éviter le scintillement qu'on a pu rencontrer avec la fonction « showHeure() ». La variable « transitionDone », qui est réinitialisée à chaque changement d'état, permet d'appeler la fonction en question qu'une seule fois, comme par exemple ci-dessous avec l'état « SHOW_HUMIDITY ».

```
case SHOW_HUMIDITY:
  if (!transitionDone) {
    showHumidity(lastHumidity > 0 ? String(lastHumidity, 1) : "...", CRGB::White);
    transitionDone = true;
  }
  break;
```

Dans les lignes suivantes, nous allons détailler l'état INIT. Comme énoncé dans l'introduction de cette section, nous avons chacun listé toutes nos fonctions, y compris les fonctions d'initialisation. Ce sont ces dernières qui sont appelées dans cet état. Le déroulé est le suivant :

- Matrice : pour afficher « RGB CLOCK » dès le lancement, la matrice doit être initialisée dès le lancement aussi.
- Capteur STH45 : cette initialisation ne prend que très peu de temps et nous informe si le capteur est trouvé ou non.
- Connexion internet : on appelle la fonction « initPortail() » dans une condition nous permettant d'agir suivant ce que retourne la fonction.
    - Si un accès à internet est possible, alors :
        - On lance l'initialisation du NTP puis on affiche l'heure sur la matrice si la réponse à la requête est bonne, auquel cas, on affiche « ERROR NTP » en multicolore. « initBoutons » est également effectué à ce moment-là.
    - Si aucun accès à internet n'est possible, alors :
        - On refait une initialisation du portail et on affiche « ERROR WF » en multicolore.

Cette architecture logicielle s'est révélée être la bonne d'après nos nombreux tests avec toujours le même réseau à proximité. Lors de l'alimentation du système, l'horloge se connecte bien à internet, récupère l'heure et la date et les affiche correctement. La température et l'humidité sont également un succès :

=> Affichage de l'heure :

=> Affichage de la date :

=> Affichage de la température :

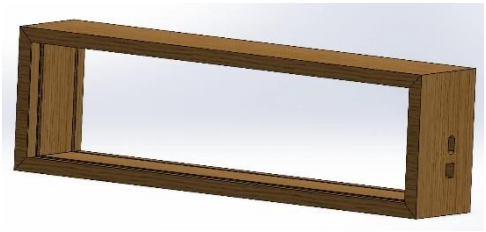
=> Affichage de l'humidité :

Le déplacement de l'horloge hors du réseau Wi-Fi déjà enregistré dans l'EEPROM a mis en lumière une erreur de code se traduisant par l'affichage constant de « ERROR WF ». Après le rendu de ce livrable, une phase de debug pour résoudre ce problème sera effectuée et pourra faire l'objet d'une note écrite présentant la solution.

Phase 3 – Fabrication et Assemblage

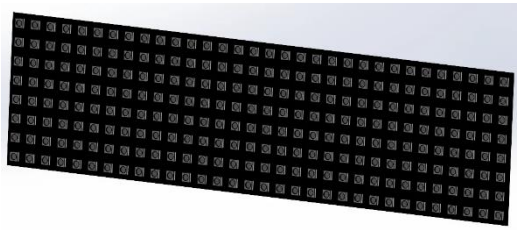
1. Conception de l'assemblage (CAO)

Modéliser le projet en 3D nous a permis de nous projeter sur un potentiel rendu final. Pour cela, nous avons utilisé SolidWorks (version étudiante gratuite). Dès le départ, nous avons porté une attention particulière au design de la boîte qui allait accueillir les composants et le PCB. L'objectif a été de créer une solution à la fois fonctionnelle et esthétique. Nous avons opté pour un matériau noble qu'est le bois. Pour accentuer le côté esthétique, les découpes de la boîte ont été réalisées en biseau, apportant une finition élégante. Évidemment, au vu du nombre de composants et de la complexité de certains, ces derniers ont été téléchargés sur les sites grabcad.com/library/ et mouser.fr. Le tableau suivant résume les composants et leurs spécifications de modélisation :



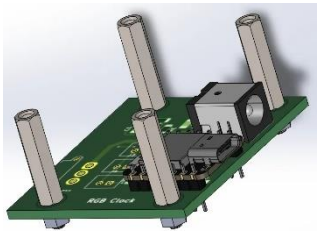
Encadré de la boite

Les rainures ainsi que les encoches pour passer les câbles ont été dessinés à la fin permettant une précision de l'emplacement.



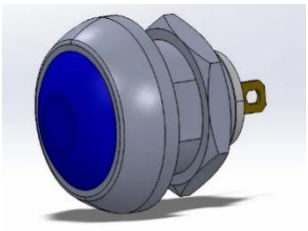
Matrice 8*32

Ne trouvant pas exactement la même matrice sur internet, nous avons réalisé un modèle équivalent sans condensateurs pour alléger la modélisation.



Assemblage du PCB

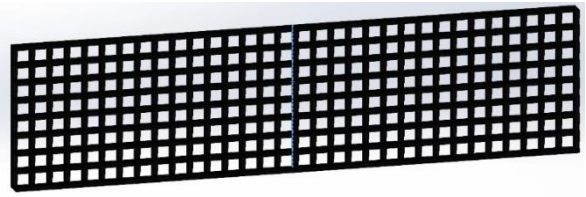
Cet assemblage a dû se faire en corrélation avec l'équipe Schéma du PCB qui a pu nous fournir une image du schéma 3D, que nous avons mis en projection sur un pavé droit coloré. Pour le connecteur d'alimentation et l'esp32, les fichiers ont été téléchargés sur les différents site évoqués plus haut. Les entretoises ainsi que les écrous ont été modélisé par le logiciel.



Bouton de paramétrage

Ces boutons ont été téléchargés sur mouser.fr et modifiés par nos soins car des bugs de modélisations étaient présents.

Grille matrice

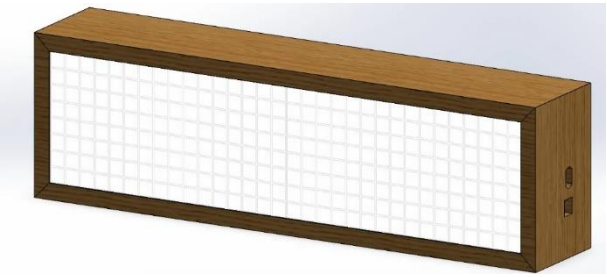


Cette grille matrice sera réalisée à l'imprimante 3D. Pour les tests de couleur (blanc ou noir) nous avions réalisé un échantillon et avions regardé ce qui rendait le mieux quand la matrice était allumée mais aussi éteinte. Dans un souci matériel, la grille sera imprimée en deux partie (trait bleu sur l'image). Elle permettra d'ajouter un espace entre la plaque d'acrylique et la matrice en plus de confiner chaque LEDs dans un espace cubique pour concentrer le faisceau de lumière et donnant un effet de pixel.

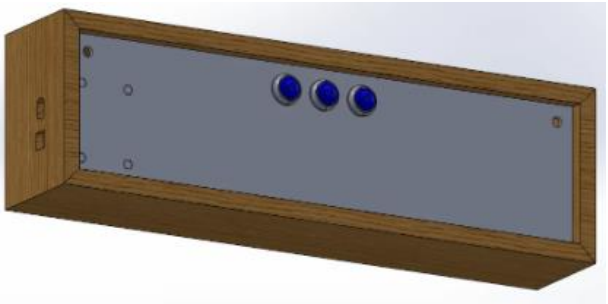
Pour le reste des composants, à savoir les plaques rigides et la plaque d'acrylique, elles ont été formées par un pavé droit comprenant :

- Pour la face arrière : 3 trous avec détrompeur pour enficher les boutons ; 2 trous de perçage pour accrocher le tout au mur ; 4 trous hexa pour placer les têtes de vis qui tiennent le PCB
- Pour la face avant : 1 trou en rainure droite permettant le passage des fils de la matrice.

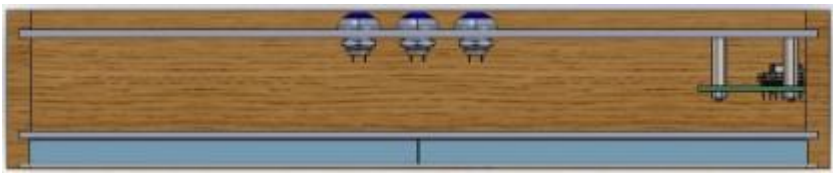
Le final de la modélisation 3D a permis de faire ces clichés :



Face avant



Face arrière

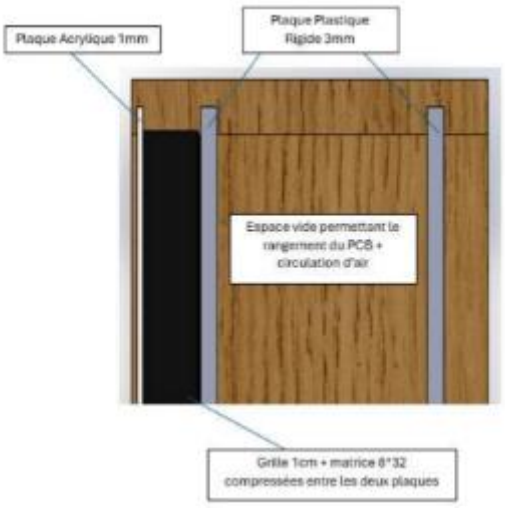


Intérieur vu du dessus

2. Assemblage mécanique

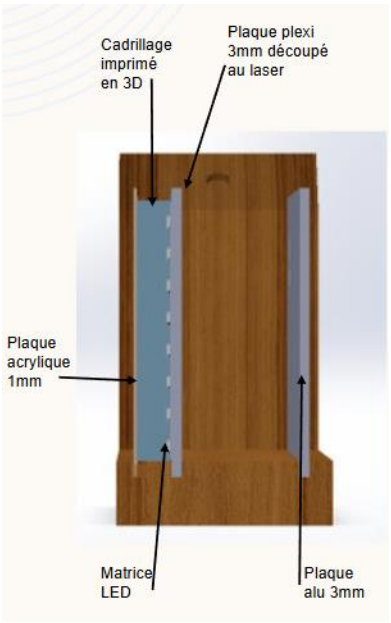
a. Récupération et modification des fichiers et plans CAO

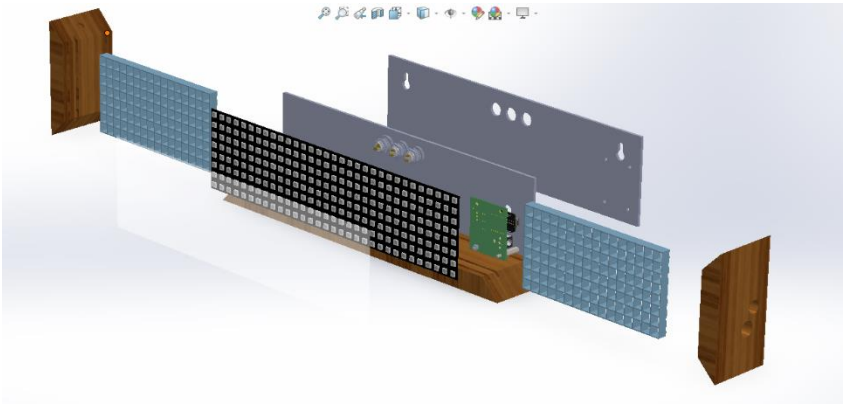
Avant de procéder à la mise en plan des différentes pièces pour l'usinage, il a fallu effectuer certaines modifications pour tenir compte des contraintes mécaniques et matérielles.



Vue en coupe avant modifications des plans

Il a été convenu que la plaque de droite ne serait pas en plastique pour des raisons de rigidité. En effet, cette plaque servira de support pour le PCB. Nous avons donc choisi une plaque en aluminium de 3 mm d'épaisseur.



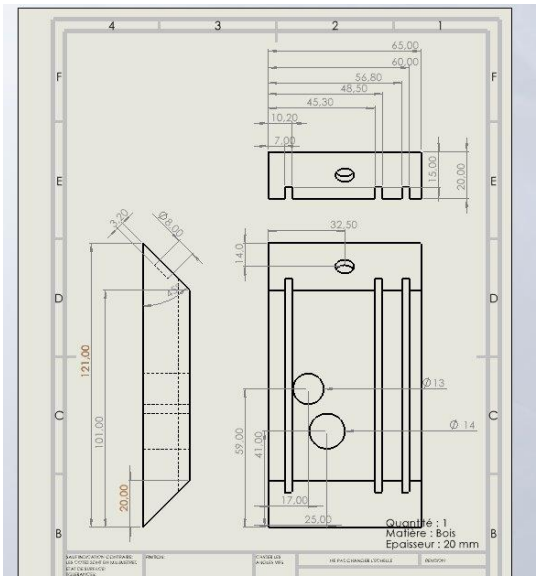


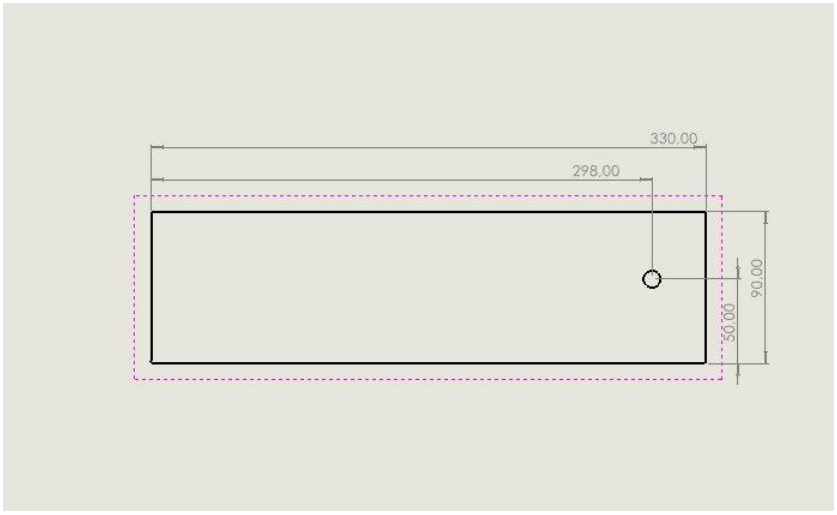
Vue en coupe et éclatée après modifications des plans et ajustement des mesures

La première version des plans a dû être modifiée après l'achat des planches de bois destinées au contour de l'horloge. Lors de la conception des pièces, nous avions initialement prévu une épaisseur de 10 mm. Cependant, cette valeur dépendait de la matière première disponible. Lors de sa recherche de matériaux, M. Lucidarme a trouvé des planches de chêne d'une épaisseur de 20 mm. Il a donc été nécessaire d'adapter les dimensions des fichiers 3D en conséquence.

b. Mise en plan des fichiers de fabrication

Pour que les planches de bois et la plaque d'aluminium puissent être usinées correctement, nous avons mis en plan les fichiers 3D. Pour cela, nous avons utilisé SolidWorks afin de générer les différentes vues 2D des pièces.





Mise en plan da la pièce de droite

Mise en plan de la plaque en PMMA

Il a ensuite fallu coter précisément chaque distance à partir d'une origine commune. Tous les perçages, extrusions et taraudages ont été annotés sur le plan, incluant :

La position (x, y) de chaque élément ainsi que les dimensions spécifiques (largeur, diamètre, angle, etc.), selon l'opération à effectuer.

Ces plans 2D ont ensuite été transmis à M. DERENNE chargé de l'usinage des pièces. Plusieurs échanges ont été nécessaires avec lui afin d'obtenir des plans clairs et précis.

c. Echanges avec M. Rayer (FabLab Polytech Angers)

Pour deux de nos pièces, nous avons choisi de les réaliser en PMMA et de les faire découper au laser par M. Rayer, technicien du FabLab de Polytech Angers. Après discussion avec lui sur les matériaux compatibles avec la découpe laser, il nous a recommandé le PMMA, un plastique transparent souvent appelé Plexiglas.

Pour la mise en plan de ces deux pièces (le cadre de 2 mm et la plaque de 3 mm), nous avons suivi la même méthode que pour les pièces en bois :

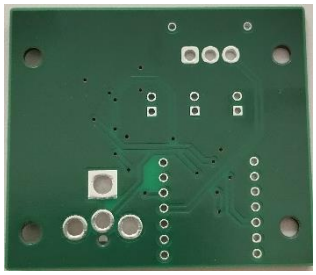
Utilisation d'une cotation claire avec une origine commune. Puis, nous avons transmis les fichiers au format DXF, un format couramment utilisé en CAO/DAO pour l'échange de plans destinés à des machines à commande numérique. Après l'envoi de ces fichiers à M. Rayer, nous nous sommes rendus au FabLab de Polytech Angers pour récupérer nos pièces.

Plaque en PMMA de 3mm découpée au laser

3. Assemblage de la carte

Après avoir fait le routage de notre carte électronique pour le fonctionnement de notre projet horloge. La commande de la carte électronique a été faite à notre retour le lundi 27 janvier puis nous l'avons reçu dans les plus brefs délais le vendredi 31 janvier. Les différentes étapes qui vont suivre permettent de s'assurer que la carte est prête à l'utilisation et sans erreur.

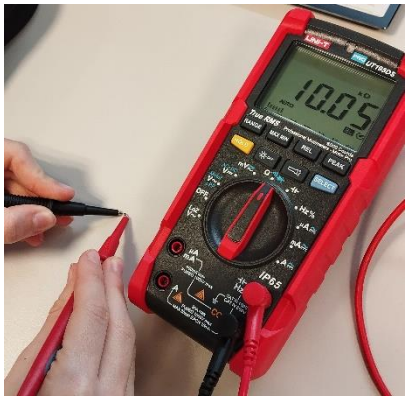
A la réception de la carte il faut tout d'abord précéder aux tests visuels de la carte. Ceci consiste à vérifier visuellement tous les défauts évidents qui peuvent se trouver sur la carte. Voici ce à quoi il faut faire attention en regardant la carte, les pistes car elles doivent être le plus nettes possible sans coupure ni court-circuit. Les vias qui traversent la carte car ils doivent être entièrement métalliser pour faire la connexion entre les différentes couches ainsi que les pads afin qu'ils soient sans défaut et propre. Les couches du PCB ont besoin également une inspection pour s'assurer qu'elle ne soit pas endommagée. La numérotation des composants doit être correcte et correspondre à la nomenclature.



C'est important de vérifier la continuité entre les pistes et les via avant de souder les composants. Pour cela nous avons besoin d'un multimètre en mode testeur de continuité afin qu'il puisse indiquer avec un bip sonore lorsque les 2 pins sont reliés. C'est ce qui nous permet de vérifier toutes les traces de circuit pour voir si elles sont connectées correctement donc de savoir si les pistes sont coupées ou en court-circuit.

De plus, j'ai demandé la nomenclature de la carte faite par Romain pour aller chercher tous les composants nécessaires afin de créer la carte.

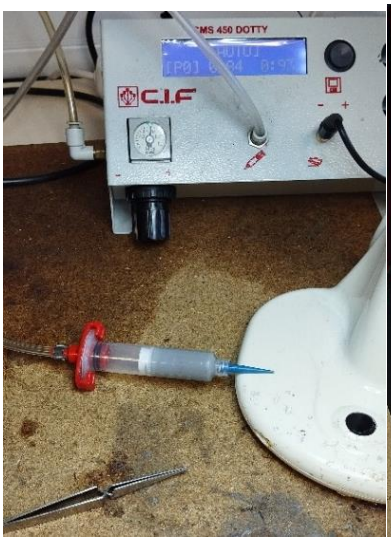
NOMENCLATURE			
References	**Valeur**	**Footprint**	**Quantité**
C1 C2	100nF	Capacitor_SMD:C_1206_3216Metric	2
C3	10µF	Capacitor_SMD:C_1206_3216Metric	1
C4	10nF	Capacitor_SMD:C_1206_3216Metric	1
J1	PJ-063BH connecteur alimentation 5V 9A	Librairie:PJ063BH	1
J2 J3 J4	B2B-XH-A_LF__SN_ connecteur XA 2points	B2B-XH-A_LF__SN_:JST_B2B-XH-A_LF__SN_	3
J5	Conn_01x03_Pin	Connector_AMASS:AMASS_MR30PW-FB_1x03_P3.50mm_Horizontal	1
R1 R2 R3 R4 R5	10k	Resistor_SMD:R_1206_3216Metric	5
U1	SHT45-AD1F-R2 capteur humidite temperature	Librairie:SHT4X	1
U2	Seeed Studio XIAO SAMD21 ESP32 C6	Connector_DIN:slot_esp32	1



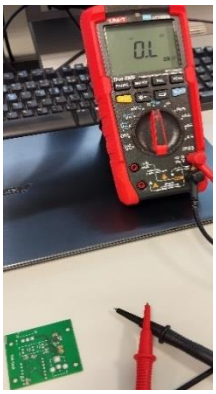
Après avoir récupéré les résistances, il faut tester leurs valeurs c'est indispensable car ce n'est pas toujours bien trié pour cela il faut s'en assurer à l'aide d'un multimètre en mode ohmmètre. On place une PIN sur chaque côté de la résistance CMS (Composant Monté en Surface) puis on vérifie la valeur qui s'affiche par rapport à celle voulu.

Après la première session de test fait, j'ai commencé à braser le PCB. En sachant qu'il y a une méthode et un ordre pour souder les composant. Il faut toujours commencer par les composants les plus petits et qui se soude sur le dessus ou le dessous de la carte afin d'avoir suffisamment de place et pour pouvoir atteindre le plus facilement leurs emplacements, puis on augmente petit à petit jusqu'au traversants aussi placé du plus petit au plus grand. J'ai entamé la pose du capteur SHT45 puis les résistances et condensateur CMS car ce sont les plus petits composants de ma carte et il se monte en surface da la carte. Pour effectuer la soudure des CMS sur le PCB j'utilise la soudure par fusion grâce à de la pâte à braser que j'applique en petite dose sur les pads des composants concerné puis je dépose délicatement les

composants dessus grâce à une pince inversée. Une fois cette étape réalisée je mets au four à fusion pendant 5 min.

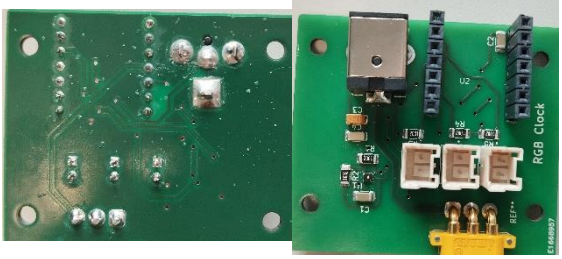


Quand la carte sort du four je m'assure que les brasures soient le plus propre possible et bien brillante ceci me permet de vérifier la bonne fusion de l'étain. Après avoir jeté un œil rapide je procède à une seconde série de tests de continuité grâce à un multimètre toujours en mode testeur de continuité. Il faut bien vérifier chaque composant soudé en s'assurant qu'il n'y a pas de court-circuit entre leur 2 bornes et qu'ils soient bien soudés pour avoir une bonne connexion.



Pour la suite du montage de la carte j'ai continué avec les composants traversants. Pour ce faire il y a besoin d'un poste à souder qui composte un fer, une bouche d'aspiration, si besoin de la tresse a déssouder et une troisième main.

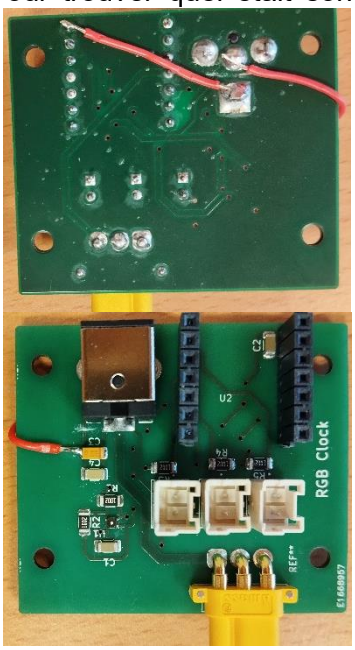
Afin de souder les composant restant on a besoin de fil d'étain qu'on vient faire fondre avec le fer à l'endroit voulu pour le maintenir. Il faut faire des soudures le plus propre possible et brillante. Il faut bien penser à commencer par le composant le plus petit et le moins haut pour ne pas se gêner pour la suite.



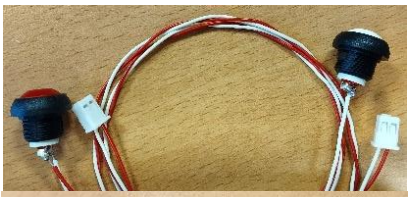
Après cette dernière partie de confection de la carte, il est indispensable de refaire des tests de continuité sur toute la carte avec tous les composants. Il faut aussi vérifier les connecteurs installés sur la carte donc leur connectivité, pour ça on va faire un test de continuité pour s'assurer qu'elles sont bien câblées et qu'il n'y a pas de court-circuit entres les broches.

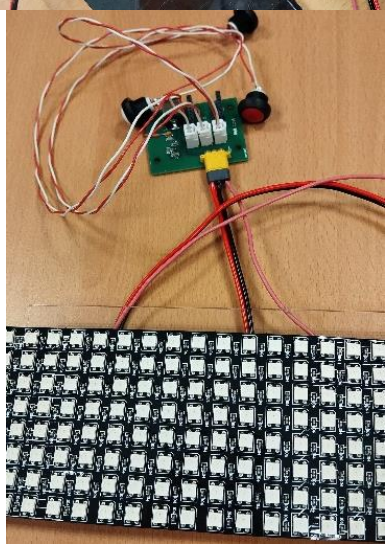
Il ne faut pas oublier de faire un test de l'alimentation qui permet de convertir le 230V du secteur en 5 V adapter pour notre carte. Pour se faire, on banche la prise au secteur puis avec un multimètre en position voltmètre on prend la mesure de la tension de sortie pour être sûr que ce soit 5V ce qui correspond à ce dont notre carte a besoin.

Après avoir banché l'alimentation, j'ai commencé les tests avec le voltmètre afin de retrouver les tensions attendues grâce au schéma de routage, mais je n'avais rien sur toute la carte. J'ai aussi surveillé la consommation de la carte lorsqu'on l'alimente et le courant était vraiment très bas proche de 0mA ce qui m'a permis d'identifier un court-circuit ou un problème autre. Sachant que la tension n'allait pas au-delà du connecteur, j'ai donc conclu que ça venait du connecteur de l'alimentation. Pour trouver quel était son problème exact, j'ai fait des tests avec celui souder sur la carte et avec celui qui était en plus pour voir s'il y avait des différences. Je me suis également renseigné en recherchant la documentation du composant pour connaitre les propriétés de chaque PIN. J'ai découvert que la pin 1 et 2 sont les 2 potentiels différents de notre carte, cependant sur le routage de la carte ils ont été reliés ensemble, donc il était tout le temps en court-circuit. J'ai donc dû faire des modifications sur le PCB afin de répondre aux fonctionnalités. Dans un premier temps, j'ai coupé 3 pistes pour ne plus faire de lien déjà entre le 5V et le GND mais aussi pour le GND et le 5V de l'esp32C6 ainsi que pour les condensateurs de découplage qui étaient reliés à aucun des potentiels. Ensuite j'ai rectifié la liaison entre le 5V et l'esp32 avec un bout de câbles puis j'ai fait pareil pour le GND et les 2 condensateurs de découplage. Après tous ces changements effectués, j'ai refait des tests de continuité pour être sur des modification faite. Il faut aussi s'assurer qu'on a les bonnes tensions à chaque point important. A l'aide d'un multimètre toujours en mode voltmètre on se met à la masse puis on vérifie toutes les tensions. On a bien les tensions attendues sur toute notre carte. Il y a beaucoup de point 5V sur notre carte mais notre capteur SHT45 s'alimente en 3,3V donc cette tension est bien en entrée de notre capteur. Mais en continuant les tests, je ne comprenais pas pourquoi l'ESP32-C6 n'étais pas alimenté. Quand j'ai continué les tests pour les points de de masse j'ai remarqué que la PIN du GND étais relier au plan de masse qui était relier aux 2 PINS du connecteur de l'alimentation qui ne sont ni 5v ni GND donc j'ai fait un pont d'étain entre le GND et les 2 autres PINS du connecteur ce qui a permis de regénérer un plan de masse correcte. Après toutes ces observations et ces modifications la carte est fonctionnelle en tous points.



Après tous ces changement la carte semble en ordre au niveau de l'alimentation et de la connectivité, donc on peut passer aux tests fonctionnels de la carte. Pour cela j'ai intégré l'esp32C6 pour pouvoir le programmer et communiquer avec le capteur SHT45 qui est implanté sur la carte. Grace au programme précédant pour gérer la récupération de données du capteur je vais pouvoir les afficher sur mon PC pour vérifier si la connexion est bonne et si la carte est fonctionnelle. Après avoir téléversé le programme j'ai testé la température et l'humidité dans la pièce ainsi que dehors et les valeurs récupérées dans le moniteur série de test sont très cohérente donc j'ai pu valider une partie des fonctionnalités de la carte. Avant de pouvoir continuer cette série de tests il a fallu réaliser le câblage des 3 boutons poussoir avec une connectique de types MOLEX pour ça on utilise la pince à sertir afin de serrer la PIN et le câble ensemble. Puis j'ai fait la deuxième partie du connecteur de la matrix LED en soudant les câbles directement dessus. Une fois que tout est banché sur la carte je programme de nouveau l'ES32¨-C6 pour faire fonctionner la matrix LED donc afficher une série de chiffre. Avec le résultat obtenu je peux valider la carte même au niveau fonctionnel.





Au vu de toutes les modifications qu'on a dû apporter à la carte on a préféré en refaire fabriquer une avec les modifications qu'on avait apportées. A son arrivée, j'ai recommencé le protocole de tests et aucun problème n'est survenue.

Conclusion

Le projet RGB CLOCK nous a permis de mettre en pratique les compétences acquises depuis le début de notre formation, notamment en électronique et en informatique embarquée. Nous avons également utilisé des compétences que nous n'avons pas abordées durant ces deux années, comme la modélisation 3D. Nous avons pu découvrir de nouveaux protocoles, comme l'I²C pour la communication entre l'ESP32 et le capteur SHT-45.

Malgré les difficultés rencontrées, telles que les défauts sur le PCB ou encore la gestion des boutons, qui n'a pas pu être implantée en raison des contraintes de temps, nous avons su nous adapter pour réaliser ce projet dans les délais. Le projet final respecte le cahier des charges initial : l'heure, la date, la température et l'humidité sont bien affichées, et le portail captif permettant de se connecter à Internet est fonctionnel. Le tout intégré dans un boîtier esthétique et moderne.

Ce projet nous a permis d'améliorer nos compétences techniques, mais aussi notre capacité à travailler en équipe, car nous étions quatre sur ce projet. Il constitue un bon point de départ pour des perspectives futures, telles que l'ajout d'une page de configuration sur le portail captif, l'intégration d'icônes pour les différents événements de l'année, l'utilisation des boutons pour naviguer à travers les différents écrans, ou encore l'ajout d'un haut-parleur.