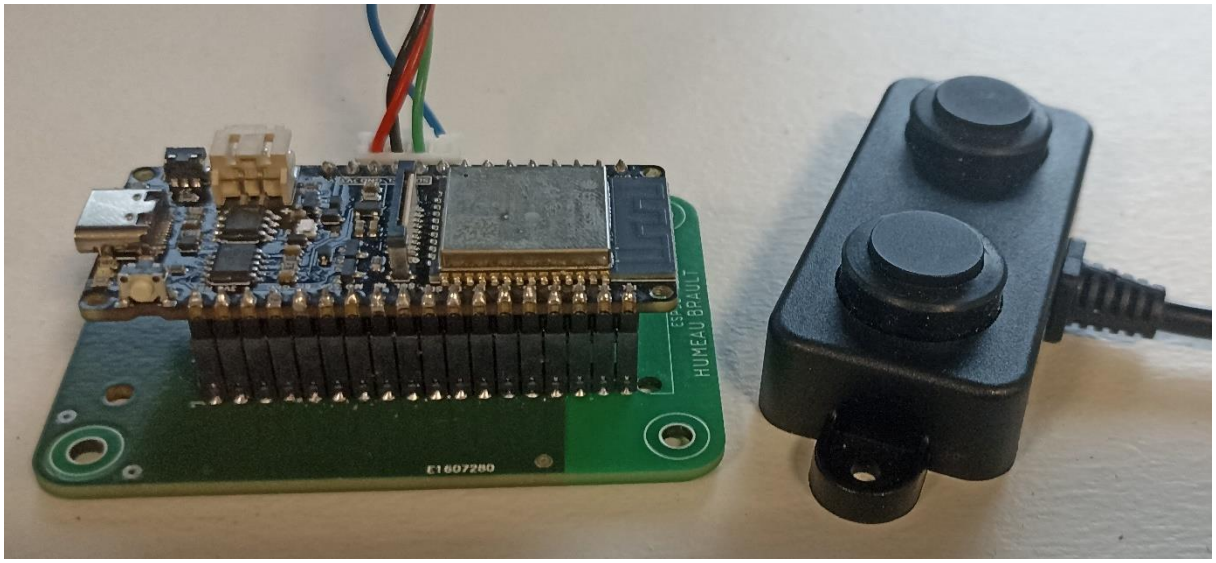
Compte-rendu

Cuve connectée



Projet réalisé par Franck HUMEAU, Marc BRAULT (Groupe G23 semestre 4, 2023 – 2024)

Projet encadré par Philippe LUCIDARME

Avec l'aide de Lionel LEDUC et de Géraldine DENECHEAU

Contact :

Franck HUMEAU : franck.humeau@etud.univ-angers.fr

Marc BRAULT : marc.brault@etud.univ-angers.fr

IUT Angers-Cholet :

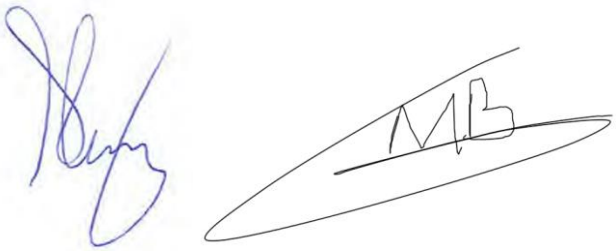
4 boulevard Lavoisier- BP 42018

49016- Angers Cedex

Table des matières

Décharge de responsabilités

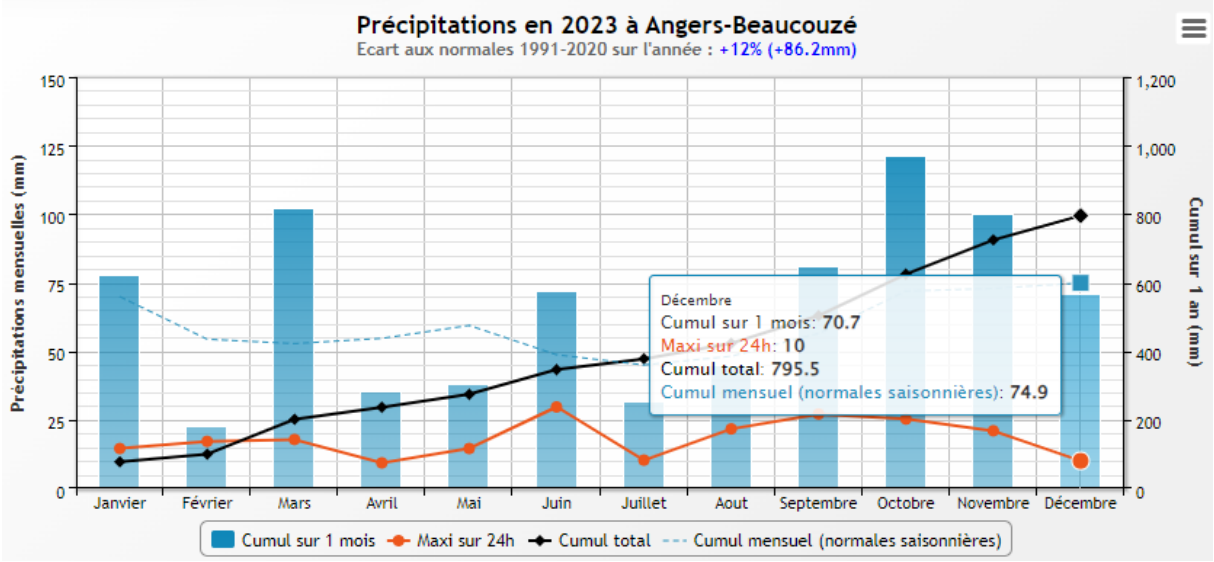
Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et 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.



Cahier des charges

Contexte

D'après le site infoclimat.fr, il y a eu 795,5 mm de précipitation à Angers sur l'année 2023, soit 795,5 litres d'eau par mètre carré. Au vu de l'importance de la ressource qu'est l'eau aujourd'hui, l'idée de récupérer de l'eau de pluie à l'aide d'une cuve peut se révéler très utile.



Par exemple à l'aide de ces chiffres, si nous prenons une maison de 100 m² à Angers et que nous prenons un coefficient de perte de 60 % d'eau au niveau du toit et du système de récupération d'eau, cela fait 47 730 litres d'eau de pluie récupérés de cette manière en 2023 (795.5*100*0.6).

C'est dans ce contexte que le projet de facilité l'utilisation d'une cuve a été imaginé par Philippe LUCIDARME. Cette volonté se manifeste dans la création d'une carte électronique pour réaliser une cuve connectée. Ce projet a été réalisé dans le cadre d'une SAE de semestre 4 en tant que BUT2 ESE par HUMEAU Franck et BRAULT Marc en reprenant les différents éléments qui ont déjà pu être produits par d'autres étudiants auparavant.

L'objectif de ce projet est de réaliser une carte électronique fonctionnelle dans le but de connaître la jauge de la cuve à l'aide d'une carte électronique munie d'un capteur à ultrason, un capteur qui renvoie la température et l'humidité et une interface web accessible par l'utilisateur afin qu'il puisse agir en conséquence.

L'objectif de ce projet est de proposer une carte électronique fonctionnelle à faible consommation. Ce rapport détaillera les différents choix de conception ainsi que les différents tests qui ont été réalisés sur ce produit final.

Choix technologique

Au niveau des composants, nous avions repris les choix qui ont été faits par les groupes précédents, c'est-à-dire que nous utilisons le capteur ultrason waterproof A02YYUW pour mesurer le niveau d'eau, pour le capteur de température et d'humidité nous utilisons le capteur SHT41-AD1B. Ce choix à l'avantage que les bibliothèques au niveau du software ont déjà été en partie réalisé et que les composants étaient disponibles à l'IUT ce qui favorise la récupération des composants, ce qui est en accord avec les valeurs du projet initial.

Pour le microcontrôleur, le groupe précédent (Tonin OLAGNON-Louis BETOU-Layla QUEYSSAC) a réalisé une étude sur la consommation de plusieurs microcontrôleurs ESP32 qui pouvait communiquer en wifi et se mettre en mode deepsleep. Cette étude a révélé que le microcontrôleur avec la plus basse consommation pour effectuer les tâches pour la cuve connectée.

Ensuite, le capteur ultrason doit avoir quelques prérequis, premièrement, il faut que ce capteur ne soit pas sensible à l'eau. Deuxièmement, il faut que ce capteur ait une faible consommation et qu'elle puisse avoir la portée nécessaire aux mesures dans la cuve. En le comparant avec d'autres capteurs à ultrason le SEN0311 (A02YYUW) est celui qui consomme le moins avec une consommation inférieure à 8 mA. Si nous le comparons au module JSN SR04T par exemple qui est également étanche, il a une consommation plus basse au repos, sa consommation est à 5 mA contre 30 mA quand il est en fonctionnement. Le capteur ultrason retenu (A02YYUW) est un capteur à communication UART qui peut mesurer de 30 cm à 4,5 mètres avec une consommation inférieure à 8 mA.

Gestion de projet

Dans une optique de gestion de projet, nous avions effectué deux diagrammes de Gantt. Un diagramme de Gantt initial durant la première semaine de projet et un deuxième diagramme de Gantt final afin d'avoir un retour sur le déroulement du projet.

Diagramme de Gantt initial

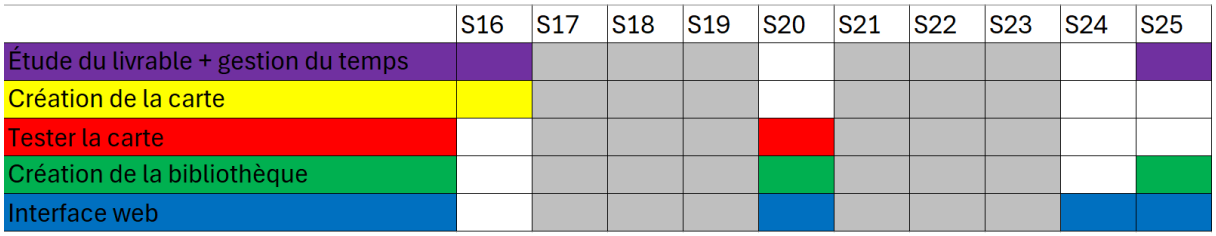
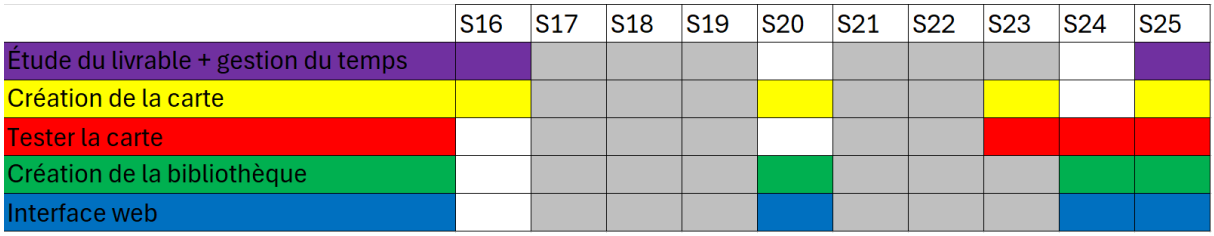


Diagramme de Gantt final



Il y a eu un écart entre les prévisions et la réalité qui peut s'expliquer par le fait que nous pensions commander la carte à la fin de la première semaine au vu du temps que nous avions pour pouvoir en faire une autre s'il y a eu un imprévu au niveau de la carte, cela ne s'est pas réalisé dans un souci au niveau du temps à disposition. Vu que la commande a été retardé et mis pour la semaine 20, Franck a pris l'initiative de la souder et de la tester pendant la semaine 23 qui était une semaine de cours à ce moment-là. Pour compenser au mieux quelques observations qui ont été faites durant le projet, un routage final a été proposé la semaine 25. Au niveau de la programmation de la partie web, un retard général au niveau des exigences a été observé.

Conception électronique

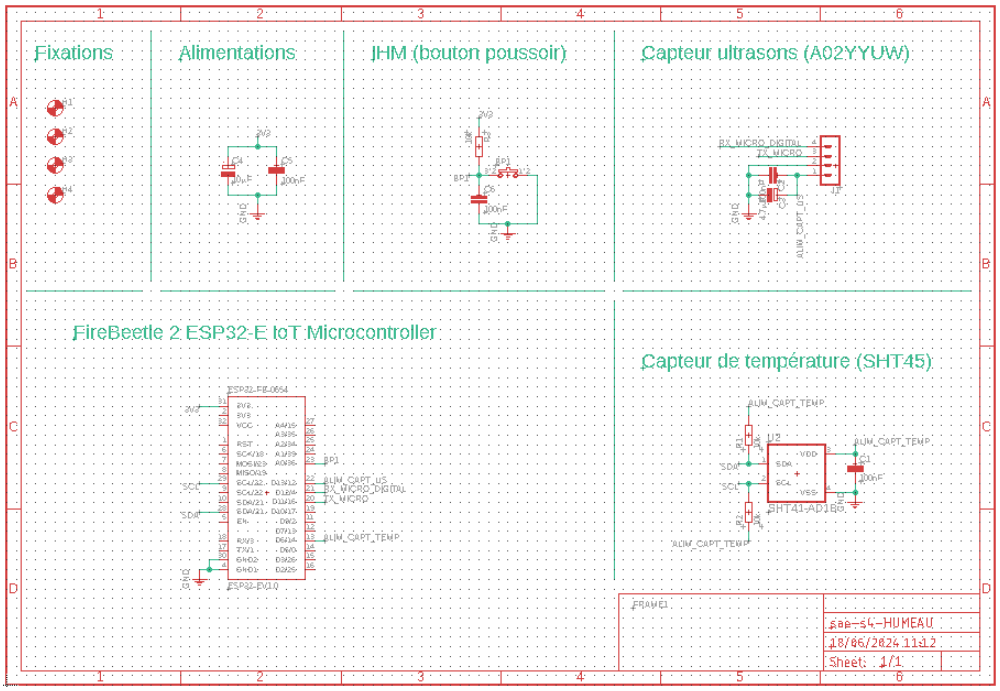
Schématique

Afin de réaliser la partie électronique de ce projet, le logiciel de CAO qui a été utilisé est le logiciel Eagle. C'est une des exigences que nous avions durant ce projet. Avant de détailler la conception

électronique, voici les différents éléments qui sont utile à la création électronique de ce projet:

Part	Value	Device	Package	Description
BP1		BP-TC	11TC05L	1 BP OFF/MON (Momentané)
C1	100nF	C-CMS1206	CCMS1206	Condensateur non Polarisé, Symbole Européen
C2	100nF	C-CMS1206	CCMS1206	Condensateur non Polarisé, Symbole Européen
C3	4.7µF	C_POL-SMC_A	SMC_A	Condensateur Polarisé
C4	10µF	C_POL-SMC_C	SMC_C	Condensateur Polarisé
C5	100nF	C-CMS1206	CCMS1206	Condensateur non Polarisé, Symbole Européen
C6	100nF	C-CMS1206	CCMS1206	Condensateur non Polarisé, Symbole Européen
ESP32-FB-0654	ESP32-EV1.0	ESP32-EV1.0	ESP32-EV1.0	Module Wifi ESP32-EV1.0
H1	HOLE3.2	HOLE3.2	3,2	HOLES
H2	HOLE3.2	HOLE3.2	3,2	HOLES
H3	HOLE3.2	HOLE3.2	3,2	HOLES
H4	HOLE3.2	HOLE3.2	3,2	HOLES
J1		CON4-EMBXAD	XA_4D	Connecteur Mâle, 4 Contacts
R1	10k	R-CMS1206	RCMS1206	Résistance Européenne
R2	10k	R-CMS1206	RCMS1206	Résistance Européenne
R3	10k	R-CMS1206	RCMS1206	Résistance Européenne
U2	SHT41-AD1B	SHT41-AD1B	SHT41-AD1B	Capteur de Température et d'Humidité

La piste initiale de la partie électronique est la reprise de certains éléments du groupe précédent en privilégiant l'ESP32 FireBeetle 2 DFR0654 dans le but de baisser la consommation électrique de la carte. À la suite de cela il eut une première version de ce schématique, puis une dernière avec quelques modifications.



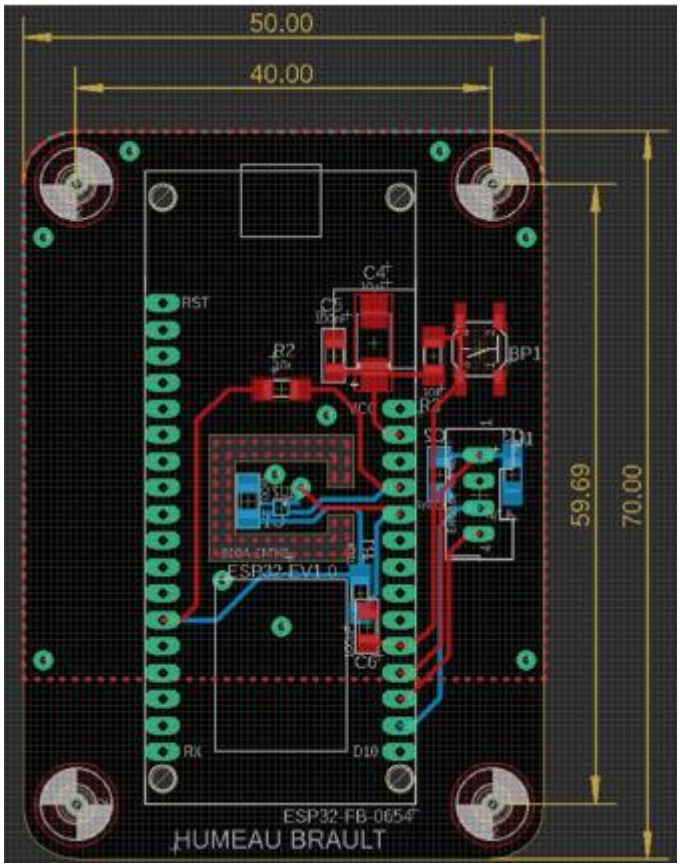
Les modifications en question sont le réajustement de certains éléments comme le bouton pour qu'il soit plus accessible et des changements de pins comme l'inversion du Tx et du Rx de la carte pour que la communication soit croisée par défaut (sans devoir le faire à la main), et en changeant le Rx de la carte pour qu'il soit sur un pin digital car en réalité cette connexion n'est là seulement pour proposer deux modes de transmission différents, le mode "processed value" quand le pin est à l'état haut et le mode "Real-time Output Selection" quand pin est à l'état bas.

Label	Name	Description
1	VCC	Power Input
2	GND	Ground
3	RX	Processed Value/Real-time Value Output Selection
4	TX	UART Output

Fig. Extrait de la documentation technique du capteur SEN0311

Routage

Pour donner suite au schéma électrique, nous avons réalisé le routage suivant :



Sur ce routage, certains éléments sont à souligner, le vide autour du capteur de température permet d'isoler thermiquement avec de l'air ce qui permet de limiter l'influence de la chaleur de la carte sur les mesures. Il est également bien de préciser que le bouton est légèrement surélevé sur le haut de la carte afin qu'il soit plus facile d'accès.

Conception logiciel

Conception logiciel

1. Objectif

Le logiciel embarqué dans le microcontrôleur (MCU) est d'une importance cruciale dans le cadre de notre projet. En utilisant le langage de programmation C++, ce logiciel assure la liaison entre le monde physique et l'utilisateur. Son rôle principal est de collecter les données provenant des capteurs, de les retransmettre à sur une page web. Cette approche vise à garantir la capture précise des informations environnementales.

2. La récupération des données

Pour la récupération de la donnée, les groupes précédents ont opté pour un capteur de température et humidité SHT45 contrôlé en I2C, ainsi qu'un capteur de distance contrôlé en UART.

Pour cela on utilise les bibliothèques :
« SensirionI2cSht4x.h Wire.h » pour le SHT45 et « sen0311.h » pour le capteur ultra son.

Un morceau de code Arduino qui permet de retourner la valeur du capteur ultra son est en annexe 1 et un morceau de code Arduino qui permet de retourner la valeur du capteur SHT45 en annexe 2

Pour l'ultrason :

Définition des Broches

```
#define Rx 17
#define Tx 16

#define ALIM_CAPT_US 12
```

Ces lignes définissent les broches du microcontrôleur utilisées pour communiquer avec le capteur. Rx (réception) est définie sur la broche 17, Tx (transmission) sur la broche 16, et ALIM_CAPT_US sur la broche 12, utilisée pour alimenter le capteur.

Déclaration des Variables Globales

```
unsigned int test = 1;
int valeur = 0;
sen0311* usSensor;
```

Ces lignes déclarent des variables globales : test est un compteur initialisé à 1, valeur stocke la distance mesurée par le capteur, et usSensor est un pointeur vers un objet de type sen0311, qui sera utilisé pour interagir avec le capteur.

Dans le Setup :

Initialisation de la Communication Série :

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

Cette ligne initialise la communication série à un débit de 115200 bauds, ce qui permet de communiquer avec un ordinateur ou un autre dispositif série ce qui est le cas de UART.

Configuration des Broches :

```
pinMode(Rx, INPUT);
pinMode(Tx, OUTPUT);
pinMode(ALIM_CAPT_US, OUTPUT);
```

pinMode configure les broches Rx et Tx en tant qu'entrée et sortie respectivement. La broche ALIM_CAPT_US est configurée comme sortie comme pensé dans notre routage.

Alimentation du Capteur :

```
digitalWrite(ALIM_CAPT_US, HIGH);
```

Cette ligne alimente le capteur à ultrasons en mettant la broche ALIM_CAPT_US à HIGH donc 3v3 pour que le capteur ne consomme pas tout le temp.

Initialisation du Capteur :

```
usSensor = new sen0311(2, Rx, Tx);
```

Cette ligne crée une nouvelle instance de l'objet sen0311 avec les paramètres spécifiés (probablement un identifiant de capteur et les broches de communication).

Dans la boucle :

Lecture de la Distance :

valeur = usSensor->getDistance();

Cette ligne lit la distance mesurée par le capteur et stocke la valeur dans « valeur ».

Vidange des Données du Capteur :

usSensor->flushUs();

Cette ligne vide le tampon des données du capteur, préparant le capteur pour une nouvelle mesure.

Affichage de la Valeur Mesurée :

printf("%d : valeur de notre capteur finale = %d mm¥n", test, valeur);

Cette ligne affiche le numéro du test actuel et la valeur de la distance mesurée en millimètres.

Incrémentation du Compteur :

test = test + 1;

Cette ligne incrémente le compteur test de 1.

Pour le capteur SHT45 :

Importation des Bibliothèques :

```
#include "SensirionI2cSht4x.h"
#include <Wire.h>
```

Ces lignes importent les bibliothèques nécessaires pour le fonctionnement du capteur SHT4x et la communication I2C. SensirionI2cSht4x.h contient les définitions et fonctions spécifiques au capteur SHT4x, tandis que Wire.h permet la communication I2C.

Définition de la Broche

#define PINTemp3V3 14

Cette ligne définit la broche 14 pour alimenter le capteur de température et d'humidité.

Création de l'Instance du Capteur

SensirionI2cSht4x SensorTH;

Cette ligne crée une instance de la classe SensirionI2cSht4x pour interagir avec le capteur SHT41.

Fonction setup

```
void setup() {
    Serial.begin(115200);
    pinMode(PINTemp3V3, OUTPUT);
    digitalWrite(PINTemp3V3, HIGH);
    Serial.println("Début de la mesure");
    Wire.begin();
    SensorTH.begin(Wire, SHT41_I2C_ADDR_44);

    float temperature, humidity;
    int16_t error = SensorTH.measureHighPrecision(temperature, humidity);

    if (error == 0) {
        Serial.print("Temperature: ");
        Serial.print(temperature);
        Serial.print(" °C, Humidity: ");
        Serial.print(humidity);
        Serial.println(" %RH");
    } else {
        Serial.print("Error reading sensor data. Error code: ");
        Serial.println(error);
    }
}
```

La fonction setup initialise la configuration du microcontrôleur. Voici les étapes en détail :

1. Initialisation de la Communication Série :

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

   Cette ligne initialise la communication série à un débit de 115200 bauds, ce qui permet de communiquer avec l'ordinateur.

2. Configuration et Alimentation du Capteur :

   ```
   pinMode(PINTemp3V3, OUTPUT);
   digitalWrite(PINTemp3V3, HIGH);
   Serial.println("Début de la mesure");
   ```

   pinMode configure la broche PINTemp3V3 comme sortie, et digitalWrite met cette broche à HIGH pour alimenter le capteur. Ensuite, un message "Début de la mesure" est imprimé pour indiquer le début des mesures, cela sert aussi à ne pas consommer trop.

3. Initialisation de la Communication I2C :

   ```
   Wire.begin();
   ```

   Cette ligne initialise la communication I2C.

4. Initialisation du Capteur :

   ```
   SensorTH.begin(Wire, SHT41_I2C_ADDR_44);
   ```

   Cette ligne initialise le capteur en utilisant l'instance Wire pour la communication I2C et l'adresse I2C du capteur SHT41_I2C_ADDR_44.

5. Mesure de la Température et de l'Humidité :

   ```
   float temperature, humidity;
   int16_t error = SensorTH.measureHighPrecision(temperature, humidity);
   ```

   Ces lignes déclarent les variables temperature et humidity pour stocker les valeurs mesurées, et appellent la fonction measureHighPrecision pour effectuer une mesure de haute précision.

6. Affichage des Résultats ou des Erreurs :

```
if (error == 0) {
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.print(" °C, Humidity: ");
    Serial.print(humidity);
    Serial.println(" %RH");
} else {
    Serial.print("Error reading sensor data, Error code: ");
    Serial.println(error);
}
```

3. L'envoie des informations sur une page web

Pour l'envoie des informations sur internet, on utilise les bibliothèques suivantes :
« WiFi.h WebServer.h Page.h »

En annexe 3 un code pour afficher une valeur des capteurs.

Wifi.h nous sert à créer la connexion wifi, webserve.h à faciliter l'utilisation de l'ESP32 comme server, Page.h contient les chaines de charactère qui contiennent l'html et le javascript des page web

1. Initialisation de la Communication Série :

```
Serial.begin(115200);
delay(1000);
Serial.println("¥n");
```

Initialise la communication série à un débit de 115200 bauds avec un délai d'attente initial pour stabiliser la communication.

2. Configuration des Broches :

```
pinMode(pinLed, OUTPUT);
pinMode(ALIM_CAPT_TEMP, OUTPUT);
pinMode(ALIM_CAPT_US, OUTPUT);
```

```
pinMode(Rx, INPUT);
pinMode(Tx, OUTPUT);
```

Configure les broches pour la LED, les capteurs de température et de distance.

3. Connexion WiFi :

```
WiFi.persistent(false);
WiFi.begin(ssid, password);
Serial.print("Tentative de connexion...");

while (WiFi.status() != WL_CONNECTED) {
   Serial.print(".");
   delay(100);
}
Serial.println("/n");
Serial.println("Connexion etablie!");
Serial.print("adresse IP: ");
Serial.println(WiFi.localIP());
```

Désactive la persistance de la connexion WiFi, puis tente de se connecter au réseau WiFi en utilisant les identifiants fournis. Affiche l'état de la connexion et l'adresse IP obtenue.

4. Configuration du Serveur Web :

```
server.on("/", homePage);
server.on("/onoff", ActionOn_Off);
server.on("/WeatherPage", WeatherPage);
server.onNotFound(pageIntrouvable);
server.begin();
Serial.println("Serveur web disponible!");
```

Définit les routes du serveur web et les fonctions associées. Initialise le serveur web.

5. Initialisation et Mesure des Capteurs :

```
digitalWrite(ALIM_CAPT_US, HIGH);
Serial.println("Début de la mesure distance");
```

```
usSensor = new sen0311(2, Rx, Tx);
distance = usSensor->getDistance();
usSensor->flushUs();

digitalWrite(ALIM_CAPT_TEMP, HIGH);
Serial.println("Début de la mesure temp/humi");
Wire.begin();
SensorTH.begin(Wire, SHT41_I2C_ADDR_44);

int16_t error = SensorTH.measureHighPrecision(temperature, humidity);

Serial.print("Temperature: ");
Serial.print(temperature);
Serial.println(" °C");

Serial.print("Humidity: ");
Serial.print(humidity);
Serial.println(" %RH");

Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" mm");

if (!(error == 0)) {
   Serial.println("Error reading sensor data.");
}
digitalWrite(ALIM_CAPT_TEMP, LOW);
digitalWrite(ALIM_CAPT_US, LOW);
```

Alimente les capteurs, initialise la communication avec eux, effectue des mesures et affiche les résultats. Désactive ensuite l'alimentation des capteurs.

Dans la boocle :

```
void loop() {
  server.handleClient();
}
```

La fonction loop gère les requêtes des clients connectés au serveur web en appelant server.handleClient().

4. Les pages web

Pour contenir les page web, dans une bibliothèque à part je créer des chaines de charactère stockés en mémoire flash, en annexe 4 qui sont envoyer comme précédemment expliqué.

Déclaration de la Variable Weather :

```
const char Weather[] PROGMEM = R"=====( ... )=====";
```

La page HTML est stockée en mémoire flash grâce à la macro PROGMEM pour économiser la RAM du microcontrôleur.

Structure HTML :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Live Sensor Data</title>
</head>
<body>
  <div>
    <p>Temperature: <span id="responsetemp">Loading...</span></p>
    <p>Humidity: <span id="responsehumi">Loading...</span></p>
    <p>Distance: <span id="responsedist">Loading...</span></p>
  </div>
</body>
</html>
```

Le corps du document HTML contient trois paragraphes pour afficher les valeurs de température, d'humidité et de distance, initialement définies sur "Loading...".

Script JavaScript :

```
<script type="module">
  // Importe la fonction 'ajax' à partir du fichier spécifié
  import { ajax } from 'https://lucidar.me/en/javascript-modules/files/ajax.min.js';
```

Le script utilise le type "module" pour permettre l'importation de la fonction ajax d'un fichier JavaScript externe.

Fonction updateSensorData :

```
  function updateSensorData() {
    ajax('http://192.168.43.147/JsonPage')
      .get()
      .then(function(response) {
        let data = JSON.parse(response);

        // Met à jour les éléments du DOM avec les nouvelles données
        document.getElementById('responsetemp').textContent = data.temp;
        document.getElementById('responsehumi').textContent = data.humi;
        document.getElementById('responsedist').textContent = data.dist;
      })
      .catch(function(failure) {
        console.error('Error:', failure);
      });
  }
```

La fonction updateSensorData effectue une requête GET vers l'URL spécifiée (/JsonPage) pour obtenir les données des capteurs. Les données JSON sont ensuite analysées et utilisées pour mettre à jour les éléments HTML correspondants.

Intervalle de Mise à Jour :

```
  // Met à jour les données toutes les secondes
  setInterval(updateSensorData, 1000);
```

La fonction updateSensorData est appelée toutes les secondes pour maintenir les données affichées à jour.

Fonctions d'Initialisation :

```
// Fonction pour mettre à jour les options (à personnaliser)
function updateOptions() {
    console.log('Updating options...');
    // Ajoutez ici le code pour mettre à jour les options de votre page
}

// Fonction pour initialiser le statut au chargement (à personnaliser)
function statusOnload() {
    console.log('Status on load...');
    // Ajoutez ici le code pour initialiser le statut de votre page
}

// Appelle les fonctions d'initialisation au chargement du DOM
document.addEventListener('DOMContentLoaded', function() {
    updateOptions();
    statusOnload();
});
```

Ces fonctions sont placeholders pour des personnalisations futures. updateOptions et statusOnload sont appelées au chargement du DOM pour effectuer des configurations initiales ou des mises à jour des options.

Bibliothèque

Une fonction a également été produite afin de faciliter l'initialisation du capteur. La bibliothèque se somme "SensorCuve" (Annexe 1). Dans le SensorCuve nous pouvons voir les lignes de codes suivantes :

#ifndef SensorCuve_h

#define SensorCuve_h

Et #endif

Pour bien comprendre, il nous faut comprendre comment est structuré une bibliothèque en C. Il y a deux fichiers pour qu'une bibliothèque fonctionne, le .cpp et le .h. Le .cpp va contenir les corps des différentes fonctions, autrement dit les différents codes fonctionnels et le .h va contenir les initialisations. Le code énoncé plus tôt permet d'être sûr d'avoir fait appel à la bibliothèque qu'une seule fois. #ifndef SensorCuve_h demande si la variable SensorCuve_h n'a pas été initialisée, si cela est

le cas le compilateur va compiler le code jusqu'à #endif. Si cela n'est pas le cas, c'est-à-dire que la variable a déjà été initialisée alors le compilateur ne va pas compiler jusqu'au prochain #endif. Vu que l'on définit la variable à l'intérieur de notre condition et notre code avec, nous sommes sûrs de ne pas l'avoir compilé plusieurs fois pour rien.

Bilan des compétences

Dans cette partie, nous détaillons individuellement les différentes compétences :

HUMEAU Franck

Concevoir : réalisation du Gantt, réalisation du schématique et du routage sur Eagle

Implanter : échange régulier avec le client (M. LUCIDARME) afin de se rapprocher au mieux à la demande client par mail et en présentiel

Vérifier : réalisation des tests de continuité de la carte, test du bouton et test du capteur ultrason

Maintenir : création de la bibliothèque "SensorCuve" afin de faciliter l'utilisation du capteur ultrason

BRAULT Marc

Concevoir : réalisation d'un serveur web

Implanter : échange régulier avec le client (M. LUCIDARME) afin de se rapprocher au mieux à la demande client en présentiel.

Vérifier : vérification du bon fonctionnement du code en procédant par création de petit bout de code.

Conclusion

Franck :

Malgré la courte durée que j'ai pu m'accorder pour ce projet, j'ai pu apprendre quelques notions importantes de projet. Si c'était à refaire, j'accorderai plus d'importance à la vérification des modèles précédents en voulant gagner du temps sur la conception, j'en ai perdu à faire des modifications qui ne seraient pas arrivées si j'avais plus pris mon temps pour lire les différentes datasheets, je pense notamment au capteur ultrason avec la borne Rx qui sert à modifier le mode de transmission. En résumé plus respecter une des règles d'or de l'électronique "prendre son temps, une étape après l'autre". J'ai également pris conscience pour la partie software qu'il a beaucoup d'outils en ligne quand on ne connaît pas quelque chose, ici je pense aux classes en c++ et dans l'utilisation des fonctions en général.

Marc :

En conclusion, le travail fournit sur le software de notre groupe m'a permis d'en apprendre davantage sur le fonctionnement et la réalisation d'un serveur web, bien que ce même travaille fournit n'apporte pas une grande utilité sur l'avancée de ce projet, car les groupes précédant avaient déjà fourni un travail très bien pensé et réalisé. Quant au hardware, j'ai appris auprès d'Internet et de mon binôme à me servir de nouveau outils sur le logiciel de CAO Eagle.

Annexe

Annexe 1 :

Le code dans le Sensor.cpp :

```
#include <Arduino.h>
#include <HardwareSerial.h>
#include "SensorCuve.h"
/*brief: fonction qui réalise l'initialisation pour le capteur ultrason
  default value : Rx 4, Tx 16, ALIM_CAPT_US 12
  param: la pin de Rx, la pin de Tx, la pin de l'alimentation du capteur ultrason
  retourne rien
*/
void initCaptUs(unsigned char Rx, unsigned char Tx, unsigned char ALIM_CAPT_US){
 pinMode(Rx, INPUT);              //Déclare Rx comme une entrée
 pinMode(Tx, OUTPUT);                    //Déclare Tx comme une sortie
 pinMode(ALIM_CAPT_US, OUTPUT);//Déclare le pin qui sert à alimenter le capteur comme une sortie
}
```

Le code dans le Sensor.h :

```
#ifndef SensorCuve_h
#define SensorCuve_h


void initCaptUs(unsigned char Rx, unsigned char Tx, unsigned char ALIM_CAPT_US); //Initialiser la fonction


#endif
```

Annexe 2

```
#include "sen0311.h"

#define Rx 17
#define Tx 16

#define ALIM_CAPT_US 12

unsigned int test = 1;
int valeur = 0;

sen0311* usSensor;

void setup() {
  Serial.begin(115200);
  pinMode(Rx, INPUT);
  pinMode(Tx, OUTPUT);
  pinMode(ALIM_CAPT_US, OUTPUT);
  digitalWrite(ALIM_CAPT_US, HIGH);
  usSensor = new sen0311(2, Rx, Tx);
}

void loop() {

  valeur = usSensor->getDistance();
  usSensor->flushUs();
  printf("%d : valeur de notre capteur finale = %d mm\n", test, valeur);
  test = test + 1;

}
```

Annex 3

```
#include "SensirionI2cSht4x.h" // Include the Sensirion SHT4x sensor library
#include <Wire.h> // Include the Wire library for I2C communication

#define PINTemp3V3 14 // Define the pin for powering the temperature sensor

SensirionI2cSht4x SensorTH; // Create an instance of the SHT41 sensor class

void setup() {
    Serial.begin(115200); // Initialize serial communication
    pinMode(PINTemp3V3, OUTPUT); // Set the pin mode for the temperature
sensor power pin
     digitalWrite(PINTemp3V3, HIGH); // Power the temperature sensor
  Serial.println("Début de la mesure");
  Wire.begin(); // Initialize I2C communication
  SensorTH.begin(Wire, SHT41_I2C_ADDR_44); // Initialize the sensor with the
I2C bus and address

  float temperature, humidity; // Variables to store the measured temperature
and humidity
  int16_t error = SensorTH.measureHighPrecision(temperature, humidity); //
Perform a high precision measurement

  if (error == 0) { // Check if the measurement was successful
    Serial.print("Temperature: "); // Print the temperature label
    Serial.print(temperature); // Print the measured temperature
    Serial.print(" °C, Humidity: "); // Print the humidity label
    Serial.print(humidity); // Print the measured humidity
    Serial.println(" %RH"); // Print the humidity unit
  } else {
    Serial.print("Error reading sensor data. Error code: "); // Print an error
message if the measurement failed
    Serial.println(error);
  }
}

void loop() {

}
```

Annex 4

```
#include <WiFi.h>
#include <WebServer.h>
#include "Page.h"
#include <stdio.h>

#include "Arduino.h"
#include <HardwareSerial.h>

#include "SensirionI2cSht4x.h" // Include the Sensirion SHT4x sensor library
#include "sen0311.h"

//#include "fonction.h"

sen0311* usSensor;
SensirionI2cSht4x SensorTH; // Create an instance of the SHT41 sensor class

#define pinLed 2  //Onboard LED LED controlled by pin IO2/D9
#define Rx 17
#define Tx 16

#define ALIM_CAPT_TEMP 14
#define ALIM_CAPT_US 12

const char *ssid = "Wait A Minute ";
const char *password = "568e32d110b9?";

int distance = 0;
float temperature, humidity; // Variables to store the measured temperature
and humidity
/*
char Json[100];
printf(Json, "{\"temp\":%.2f,\"humi\":%.2f,\"dist\":%d}", temperature,
humidity, distance);
*/
WebServer server(80);

void ActionOn_Off()
{
  if ( digitalRead(pinLed)){
    digitalWrite(pinLed, LOW);
  }
  else{
    digitalWrite(pinLed, HIGH);
  }
  server.sendHeader("Location","/");
  server.send(303);
```

```
}

void pageIntrouvable()
{
  server.send(404,"text/plain","404:page introuvable");
}

void homePage()
{
  server.send(200,"text/html",Home);
}

void WeatherPage()
{
  server.send(200,"text/html",Weather);
}

/*
void JsonPage()
{
  server.send(200,"text/json",Json);
}

char Json[100];
  sprintf(Json, "{\"temp\":%.2f,\"humi\":%.2f,\"dist\":%.2f}", temperature,
humidity, distance);
*/
void setup() {

  Serial.begin(115200);
  delay(1000);
  Serial.println("\n");

  pinMode(pinLed,OUTPUT);
  pinMode(ALIM_CAPT_TEMP, OUTPUT);
  pinMode(ALIM_CAPT_US, OUTPUT);
  pinMode(Rx, INPUT);
  pinMode(Tx, OUTPUT);

  WiFi.persistent(false);
  WiFi.begin(ssid, password);
  Serial.print("Tentative de connexion...");

  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(100);
  }
  Serial.println("/n");
```

```
  Serial.println("Connexion etablie!");
  Serial.print("adresse IP: ");
  Serial.println(WiFi.localIP());

  server.on("/", homePage);
  server.on("/onoff", ActionOn_Off);
  server.on("/WeatherPage", WeatherPage);
  //server.on("/WeatherPage", Json);
  server.onNotFound(pageIntrouvable);
  server.begin();
  Serial.println("Serveur web disponible!");

  digitalWrite(ALIM_CAPT_US, HIGH);
  Serial.println("Début de la mesure distance");
  usSensor = new sen0311(2, Rx, Tx);
  distance = usSensor->getDistance();
  usSensor->flushUs();

  digitalWrite(ALIM_CAPT_TEMP, HIGH);
  Serial.println("Début de la mesure temp/humi");
  Wire.begin(); // Initialize the I2C communication
  SensorTH.begin(Wire, SHT41_I2C_ADDR_44); // Initialize the sensor with the
I2C bus and address

  int16_t error = SensorTH.measureHighPrecision(temperature, humidity); //
Perform a high precision measurement

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %RH");

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" mm");

  if (!(error == 0)) { // Check if the measurement was successful
    Serial.println("Error reading sensor data."); // Print an error message if
the measurement failed
  }
  digitalWrite(ALIM_CAPT_TEMP, LOW);
  digitalWrite(ALIM_CAPT_US, LOW);
}

void loop() {
  server.handleClient();
```

Annexe 5

```
const char Home[] PROGMEM = R"=====(
<!DOCTYPE html>
    <html lang="fr">

        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-
scale=1.0">
            <title>Cuve Connectée/HomePage</title>
        </head>

        <body>
                <h2>Home</h2>
                <h3> ESP-32 Pour tester la connection serveur web </h3>
                <a class="nav-link" href="/onoff"><h3> voyant On/Off </h3></a
                <h3> Pour voir les infos de la cuve </h3>
                <a class="nav-link" href="/WeatherPage"><h3> WeatherPage
</h3></a>
        </body>

    </html>

)=====";

const char Weather[] PROGMEM = R"=====(
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Live Sensor Data</title>
</head>
<body>
    <div>
        <p>Temperature: <span id="responsetemp">Loading...</span></p>
        <p>Humidity: <span id="responsehumi">Loading...</span></p>
        <p>Distance: <span id="responsedist">Loading...</span></p>
    </div>

    <script type="module">
        // Importe la fonction 'ajax' à partir du fichier spécifié
        import { ajax } from 'https://lucidar.me/en/javascript-
modules/files/ajax.min.js';

        // Fonction pour mettre à jour les informations de temp, humi, dist
        function updateSensorData() {
```

```
            ajax('http://http://192.168.43.147/JsonPage')
                .get()
                .then(function(response) {
                    let data = JSON.parse(response);

                    // Met à jour les éléments du DOM avec les nouvelles
données
                    document.getElementById('responsetemp').textContent =
data.temp;
                    document.getElementById('responsehumi').textContent =
data.humi;
                    document.getElementById('responsedist').textContent =
data.dist;
                })
                .catch(function(failure) {
                    console.error('Error:', failure);
                });
        }

        // Met à jour les données toutes les secondes
        setInterval(updateSensorData, 1000);

        // Fonction pour mettre à jour les options (à personnaliser)
        function updateOptions() {
            // Exemple d'options à mettre à jour
            console.log('Updating options...');
            // Ajoutez ici le code pour mettre à jour les options de votre
page
        }

        // Fonction pour initialiser le statut au chargement (à personnaliser)
        function statusOnload() {
            console.log('Status on load...');
            // Ajoutez ici le code pour initialiser le statut de votre page
        }

        // Appelle les fonctions d'initialisation au chargement du DOM
        document.addEventListener('DOMContentLoaded', function() {
            updateOptions();
            statusOnload();
        });
    </script>
</body>
</html>

)=====";
```

Annexe 6

```
#include <WiFi.h>

#include <WebServer.h>
#include "Page.h"
#include <stdio.h>
#include "Arduino.h"
#include <HardwareSerial.h>
#include "SensirionI2cSht4x.h" // Include the Sensirion SHT4x sensor library
#include <Wire.h> // Include the Wire library for I2C communication
#include "sen0311.h" // Include the SEN0311 ultrasonic distance sensor library

// Sensor object declarations
sen0311* usSensor;
SensirionI2cSht4x SensorTH; // Create an instance of the SHT41 sensor class

// Pin and constant definitions
#define pinLed 2  // Onboard LED controlled by pin IO2/D9
#define Rx 17
#define Tx 16
#define ALIM_CAPT_TEMP 14 // Power pin for the temperature sensor
#define ALIM_CAPT_US 12 // Power pin for the ultrasonic sensor
// WiFi connection information
const char *ssid = "Wait A Minute ";
const char *password = "568e32d110b9?";

// Global variables for storing sensor data
int distance;
sen0311 sensor(1, 16, 17);
float temperature, humidity; // Variables for temperature and humidity
char Json[100]; // Buffer to store the JSON string

WebServer server(80); // Create a web server on port 80

// Function to toggle the LED state
void ActionOn_Off() {
  if (digitalRead(pinLed)) {
    digitalWrite(pinLed, LOW);
  } else {
    digitalWrite(pinLed, HIGH);
  }
  server.sendHeader("Location", "/");
  server.send(303);
}

// Function to handle not found pages
void pageIntrouvable() {
```

```
  server.on("/JsonPage", JsonPage);
  server.onNotFound(pageIntrouvable);
  server.begin();



  delay(1000); // Add a delay to avoid too frequent measurements
}

void loop() {

  /*
        récupération valeurs capteurs
*/
  digitalWrite(ALIM_CAPT_US, HIGH);
  distance = usSensor->getDistance();
  usSensor->flushUs();
  digitalWrite(ALIM_CAPT_US, LOW);

  // Initialize and read data from the temperature and humidity sensor
  digitalWrite(ALIM_CAPT_TEMP, HIGH);
  Serial.println("Début de la mesure");
  Wire.begin(); // Initialize I2C communication
  SensorTH.begin(Wire, SHT41_I2C_ADDR_44); // Initialize the sensor with the
I2C bus and address
  // Perform a high precision measurement
  SensorTH.measureHighPrecision(temperature, humidity);

  digitalWrite(ALIM_CAPT_TEMP, LOW);
  digitalWrite(ALIM_CAPT_US, LOW);

  server.handleClient(); // Handle client requestsé


}
```

```
  server.send(404, "text/plain", "404: page introuvable");
}

// Function to display the home page
void homePage() {
  server.send(200, "text/html", Home);
}

// Function to display the weather page
void WeatherPage() {
  server.send(200, "text/html", Weather);
}

// Function to send JSON data
void JsonPage() {
  // Update the JSON string with the current data
  sprintf(Json, "{\"temp\":%.2f,\"humi\":%.2f,\"dist\":%d}", temperature,
humidity, distance);
  server.send(200, "application/json", Json);
}

void setup() {
  Serial.begin(115200); // Initialize serial communication
  delay(1000);

  // Set pin modes
  pinMode(pinLed, OUTPUT);
  pinMode(Rx, INPUT);
  pinMode(Tx, OUTPUT);
  pinMode(ALIM_CAPT_TEMP, OUTPUT);
  pinMode(ALIM_CAPT_US, OUTPUT);
  usSensor = new sen0311(2, Rx, Tx);

  // Connect to WiFi
  //WiFi.persistent(false);
  WiFi.begin(ssid, password);

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }

  Serial.println(WiFi.localIP());

  server.enableCORS();
  // Set up web server routes
  server.on("/", homePage);
  server.on("/onoff", ActionOn_Off);
  server.on("/WeatherPage", WeatherPage);
```