Encadré par M. Lucidarme

Projet SAE5

Thermomètre connecté

2023

Gweltaz Burlot

Lucas Olier

Mathis Rudkiewicz

université angers

iut Angers Cholet

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

Sommaire

Remerciement

Nous souhaitons exprimer notre sincère gratitude et mes remerciements à toutes les personnes qui ont contribué à la réalisation de ce projet.

Un merci particulier à Monsieur Lucidarme, notre professeur et superviseur, pour son encadrement, son soutien et ses précieux conseils tout au long de ce projet. Il a su nous guider dans les choix techniques et les réalisations. Son aide a été un pilier dans le succès de notre travail.

Nous tenons également à remercier chaleureusement Géraldine pour sa contribution essentielle à l'envoi de la carte à la fabrication. Sa gestion efficace et son engagement ont grandement facilité cette étape cruciale du projet.

Nos remerciements vont aussi à Pierre-Alexandre, camarade, pour son aide précieuse dans les travaux de soudure. Ses compétences techniques et sa disponibilité ont été d'une grande aide dans la concrétisation de notre projet.

Nous sommes reconnaissants envers chacun pour leur soutien, leur encouragement et leur contribution à ce projet. Leur aide a non seulement facilité ma tâche, mais a également enrichi mon expérience tout au long de cette aventure.

Introduction

Nous évoluons actuellement dans un contexte axé sur l'économie d'énergie et le respect de l'environnement. Dans cette perspective, il devient particulièrement intéressant d'ajuster le chauffage et la climatisation en fonction des conditions météorologiques pour réaliser des économies et adopter un mode de vie plus écologique. C'est dans cette optique que nous vous introduisons à un outil pratique : le thermomètre connecté.

Ce projet vise à développer un système capable de mesurer de manière autonome et périodique la température d'un environnement. Ce thermomètre offre ainsi une solution concrète aux enjeux actuels de l'efficacité énergétique et du développement durable.

À travers ce rapport, nous détaillerons le processus de conception et de réalisation de ce dispositif. De l'intégration d'un microcontrôleur pour la collecte des données, en passant par le stockage des données, jusqu'à la création d'une interface web pour la visualisation.

Le rapport présente non seulement les aspects techniques du projet mais souligne également les défis rencontrés et les solutions adoptées. Nous montrerons aussi de potentielles perspectives d'amélioration de notre thermomètre connecté.

1. Cahier des charges

Introduction et Contexte

Objectif du Projet : Développer un système capable d'acquérir et de transmettre des données de température de manière périodique et autonome. Ce système comprendra un capteur de température, une base de données pour stocker les données, et une interface web pour la visualisation.

Besoins et Exigences

Exigences Fonctionnelles :

- Acquisition périodique des données de température.
- Stockage des données dans Firestore.
- Affichage des données via une interface web.

Exigences Non Fonctionnelles :

- Fiabilité et précision des mesures de température.
- Compatibilité multiplateforme du client web.

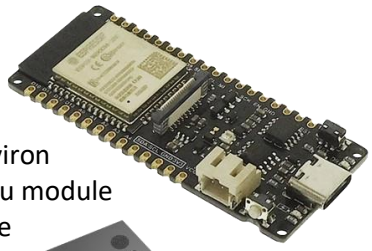
Planification et Étapes

Échéancier : Le projet sera réalisé sur une période de 4 semaines.

Priorité des livrables :

- Acquisition de données de températures et d'humidité
- Stockage sur une base de données
- Visualisation des valeurs sur une interface web

2. Choix technologiques



Pour répondre à ce cahier des charges nous avons choisi de nous diriger vers le capteur de température *SHT41 par Sensirion* ainsi qu'un microcontrôleur ESP 32. Cet ensemble nous permet de réaliser des mesures de température assez fiable (une précision d'environ +/- 0,2°C). Nous pouvons également communiquer en Wi-Fi grâce au module et l'antenne présente sur l'ESP 32 le tout en consommant peu lors de son utilisation (42mA pour l'ESP 32 en fonctionnement au maximum et 1,5 mA pour le capteur).

Concernant la base de données, nous avons choisi *Firebase*, une base de données créée par Google. Cette base de données est gratuite (sous une certaine limite de requête journalière) et facile à prendre en main grâce au nombreuses librairies et aide disponible en ligne. Cette base de données nous permettra donc de stocker les données émises par l'ESP32.

3. Réalisations

A. Conception de la carte Capteur

Dans cette partie nous allons nous intéresser à la conception de la carte qui supportera le capteur.

Pour ce projet nous avons dû concevoir une carte PCB pouvant accueillir le Capteur SHT41. Nous avons décidé de concevoir un « *shield* » pour notre ESP32. Un Shield est une carte modulaire qui, généralement, se branche sur toutes les broches de la carte Arduino (ou ESP32 dans notre cas). De plus, la conception de la carte doit prendre en compte la dissipation générée par l'ESP32. En effet, notre projet consiste à mesurer la température de l'environnement ambiant au capteur or, celui-ci est placé très proche de l'esp 32. Le capteur pourrait donc mesurer uniquement les perturbations générées par le µc plutôt que l'environnement ambiant.

Conception du schéma électrique :

Pour le schéma électrique du capteur, nous avons reproduit le schéma fourni dans la documentation technique : [SHT41-DataSheet] (voir fig.1). Celui-ci est simplement composé de deux résistances de Pull-up sur la communication I2C et d'un condensateur de découplage au niveau de l'alimentation du capteur.
A noter que sur le schéma, nous connectons en réalité a seulement un seul 3v3/GND/SCL/SDA contrairement a ce qu'indique le schéma ci-contre.

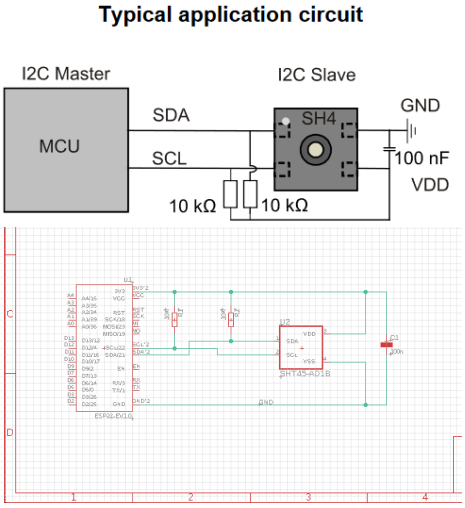


Figure 1 - Schéma de câblage du capteur

Routage de la carte

Le premier routage de la carte avait été conçu sur les deux faces de la carte, il permet d'isoler le capteur de température des 3 autres composants. Grace à cela, le capteur évite les problèmes de chaleur généré par l'ESP32.

Malheureusement cette version possède trois défauts majeurs :

- Le premier est que cette première version nécessite une optimisation des pistes qui pouvait être raccourcie. Par exemple, le condensateur de découplage placé plus près du capteur.
- Le second est que la carte n'aurait pas pu être directement soudé à l'ESP32 si celle-ci comportait des composant sur le côté rouge du routage (faisant face à l'ESP32). Il faut donc que tous les composants soient du coté faisant dos à l'ESP (côté bleu).
- Le dernier défaut est que l'antenne Wi-Fi allait être perturbée par notre carte car celle-ci possède un plan de masse passant juste en dessous de l'antenne (voir fig.2).

Figure 2 - Antenne Wifi présente sur l'ESP32

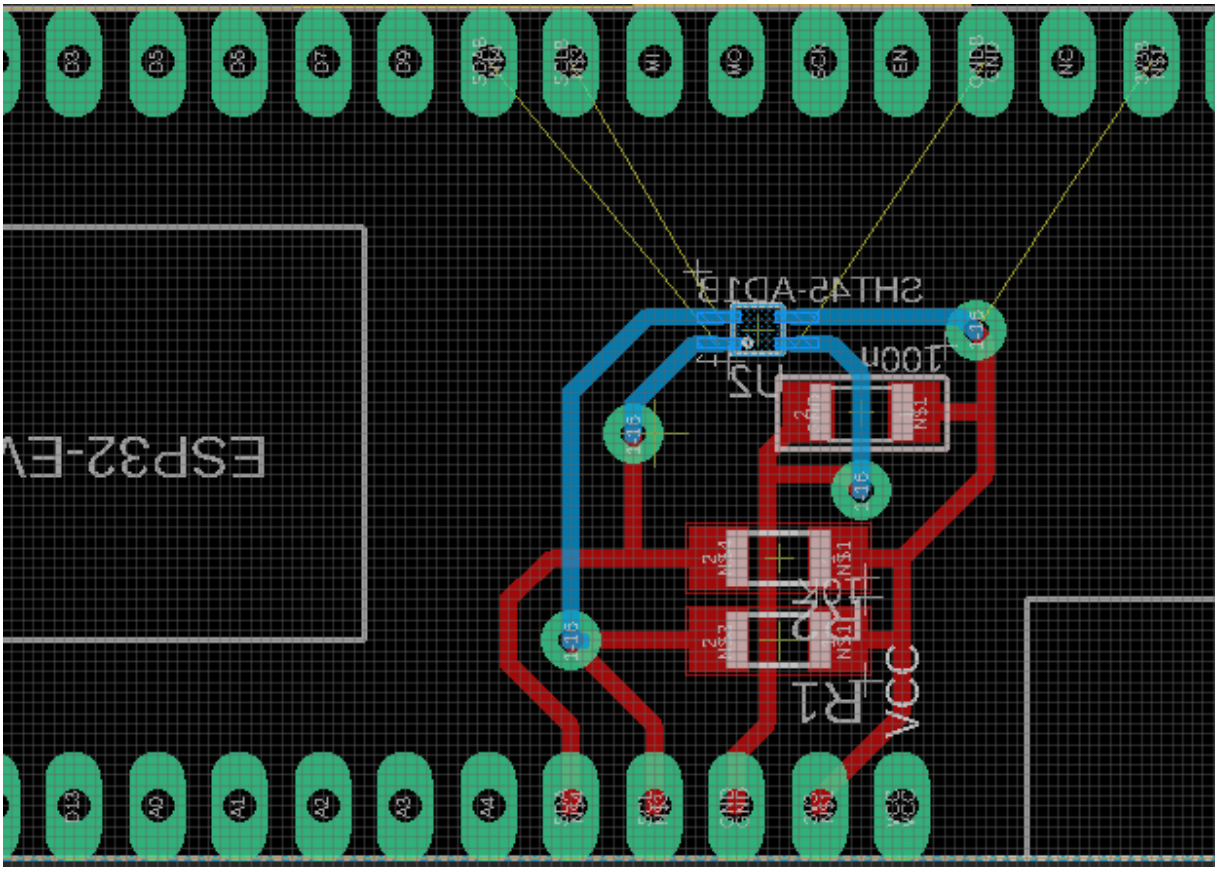


Figure 3 - Première version du routage de la carte

Notre deuxième routage corrige donc tous ces défauts et est celui que nous avons décidés de garder pour la suite du projet.

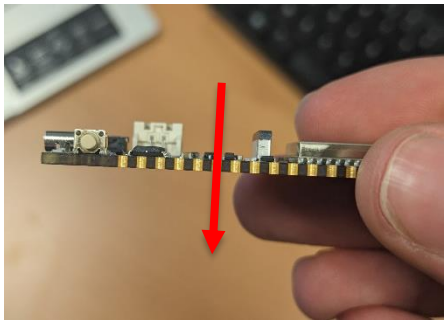


Figure 4 - Partie de l'ESP32 avec le moins de composants

- Celle-ci est conçue sur uniquement une seule face du PCB. Pour éviter les problèmes de température venant de l'ESP 32, nous avons placé le capteur en dessous de l'endroit où il y a le moins de composants (voir flèche rouge fig.4). Ainsi, il y a moins de risques de mesurer des températures erronées.

- Pour éviter les interférences du plan de masse avec l'antenne Wi-Fi, nous avons raccourci le PCB en libérant ainsi l'antenne Wi-Fi.

- Enfin, Deux versions ont été réalisées afin d'essayer au maximum les longueurs de piste en jouant sur les connecteurs doublés. Après plusieurs essais, le fait de jouer avec les connecteurs doublés n'aide pas vraiment le routage, il y aura toujours une piste qui rentrera en conflit avec une seconde (soit les deux pistes de l'I2C ou le 3v3 d'alimentation). Pour contrer ce conflit nous sommes obligés de passer par le composant (voir fig.5).

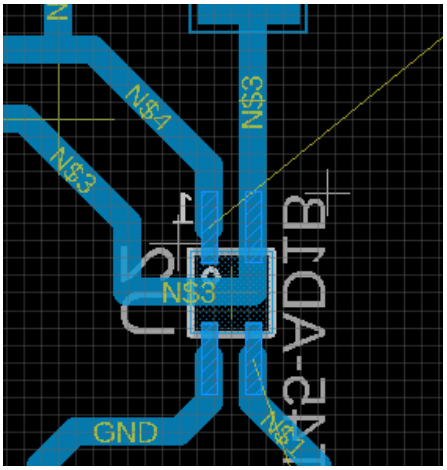


Figure 5 - Technique de routage consistant à passer sous le capteur

Nous avons donc choisi de partir sur la version ci-dessous pour la suite de notre projet :

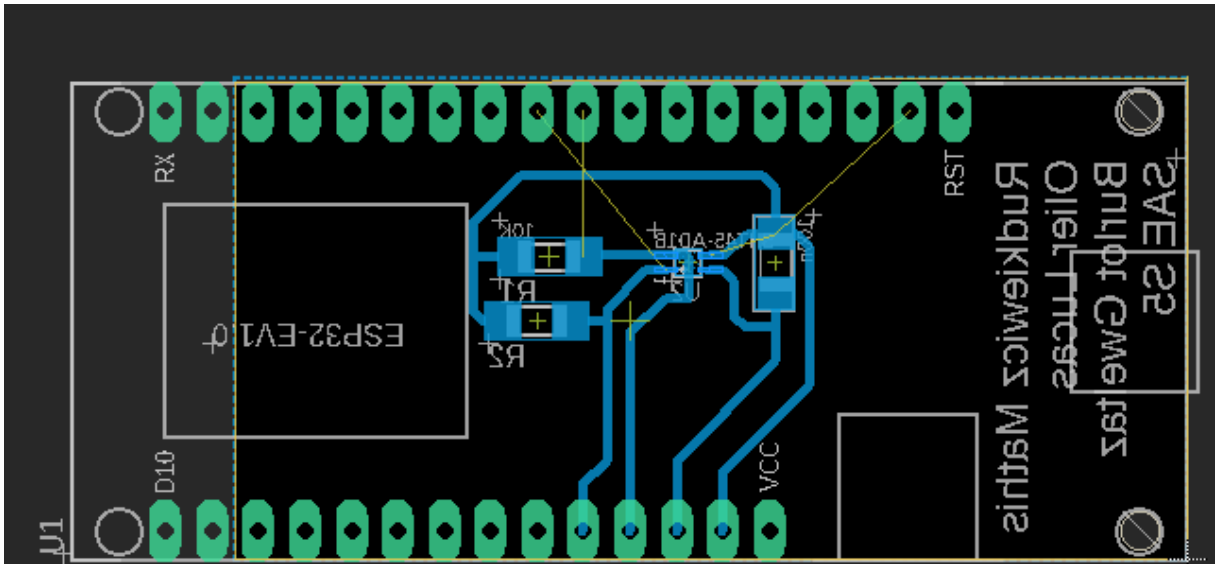


Figure 6 - Version finale du routage

Intégration de la carte à l'ESP32

Une fois la carte PCB réceptionnée, nous avons pu souder les composants a celle-ci. Nous avons rencontré quelques difficultés durant ce processus. Nous voulions souder tous les composants CMS avec de la pâte CMS puis mettant la carte au four. Cependant nous n'étions pas au courant mais le four à l'IUT ne fonctionne pas très bien (une des deux résistances chauffantes ne fonctionne plus) la soudure ne s'est donc pas réalisée correctement (composants en court-circuit car la pâte s'est amollie et s'est étalée au-delà des empreintes). Nous avons donc dû reprendre toutes les soudures sur la carte au fer à souder.

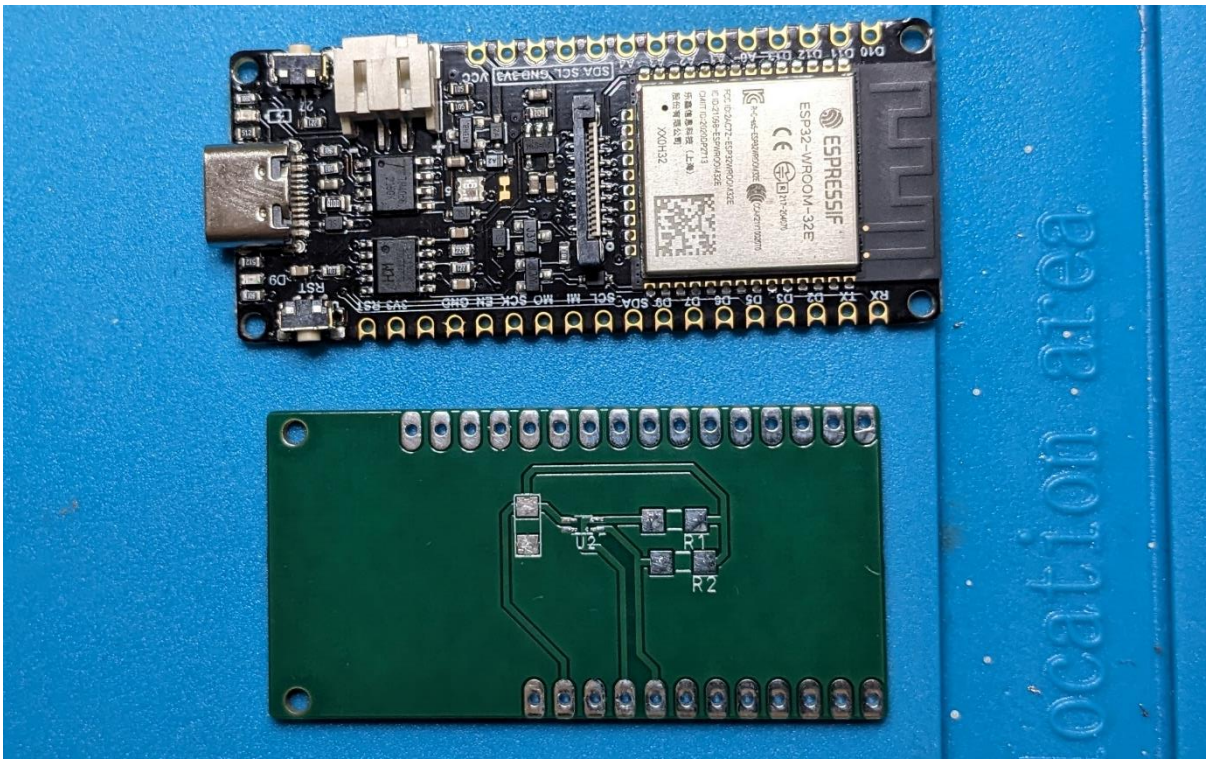


Figure 7 - Carte avant les soudures

Avant de réessayer les soudures, nous avons nettoyé la pâte CMS en y appliquant de l'étain par-dessus. Grâce à l'étain, nous avons pu absorber l'excès de pate. Nous avons ensuite retiré l'étain à l'aide d'une tresse en cuivre pour que le nettoyage soit terminé.

Nous avons eu beaucoup de difficulté à souder le capteur de température en raison de sa taille, nous avons dû nous y prendre à plusieurs reprises en y appliquant le souffleur à air chaud sur les empreintes pour chauffer l'étain présent dessus.

Avant de souder notre carte à l'ESP 32, nous avons fait un test pour s'assurer de la bonne communication entre les deux cartes. Pour cela, nous avons refait le test du driver en reliant l'ESP 32 et notre carte avec des câbles « *Dupont* ». Une fois les deux connectés, nous avons lancé le Driver que nous avons conçu et vérifié que le capteur nous envoyait des données de température et d'humidité.

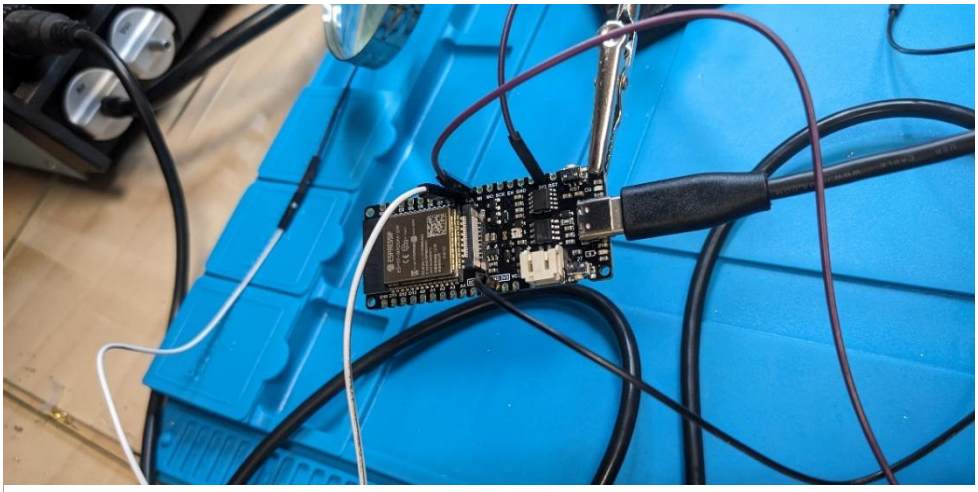


Figure 8 - Premier test avec des branchements "flottants"

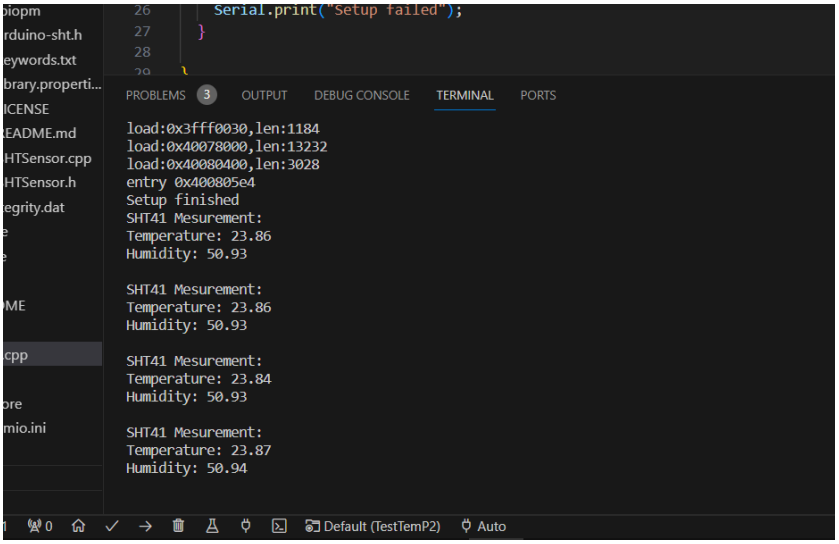


Figure 9 - Réception des valeurs correctes de température et d'humidité lors du test

Une fois que nous nous sommes assurés que la carte fonctionne correctement avec l'ESP 32, nous avons soudés les deux cartes entre elles a l'aide de connecteur que nous avons découpés une fois ceux-ci complétement soudé. Nous avons ensuite réalisé un nouveau test comme le précédent pour s'assurer du fonctionnement.

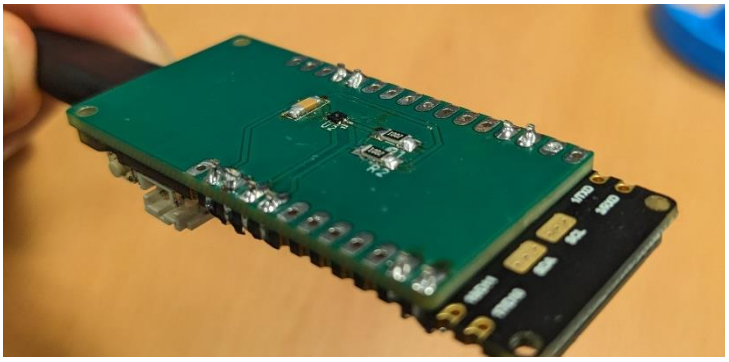


Figure 10 - Carte finale soudée à l'ESP32

B. Programmation du capteur (SHT41)

Dans cette section, nous vous détaillerons le processus d'acquisition des données à partir du capteur de température et d'humidité SHT41. Il est important de noter que le capteur SHT41 est configuré pour fonctionner avec le protocole I2C.

Initialisation :

Pour la programmation nous avons utilisé la bibliothèque *SHTSensor.h*. Cette bibliothèque nous permet non seulement de spécifier que nous travaillons avec le modèle SHT41, mais aussi d'assurer que nous utilisons les bonnes adresses et fonctions du capteur, soit 0x44 ou 0x45. Par la suite, l'ESP32, agissant comme maître, établit une communication via le protocole I2C.

```
#include "SHTSensor.h" // bibliothèque du capteur (pour les SHT)

SHTSensor sht41(SHTSensor::SHT4X); // On utilise la version SHT4X( blibliothèque)
```

Figure 11 - Initialisation de la librairie et du capteur SHT

L'initialisation du capteur est confirmée automatique dans la console par l'affichage du message "*Setup finished*", tandis qu'un échec entraîne l'apparition du message "*Setup failed*", indiquant la nécessité de vérifier la configuration.

Programme principal :

Au cœur du programme principal, la lecture des données est effectuée en vérifiant si la méthode `readSample()`, du capteur SHT41, parvient à collecter les informations. En cas de succès, la température et l'humidité sont récupérées et affichées. Nous utilisons `Serial.print` pour afficher "Temperature : " suivi de la valeur de la température obtenue avec la fonction `getTemperature()`, affichant deux chiffres après la virgule pour plus de précisions. Un processus similaire est suivi pour l'humidité, où "Humidity : " est affiché avant la valeur obtenue par `getHumidity()`.

Pour contrôler la fréquence des mesures, un délai est introduit, ce qui nous permet de définir l'intervalle entre chaque mesure, par exemple, une mesure toutes les secondes. Si la lecture échoue, nous assurons la gestion des erreurs en affichant le message "Error", ce qui signale un problème dans la lecture des données et nécessite une investigation plus approfondie.

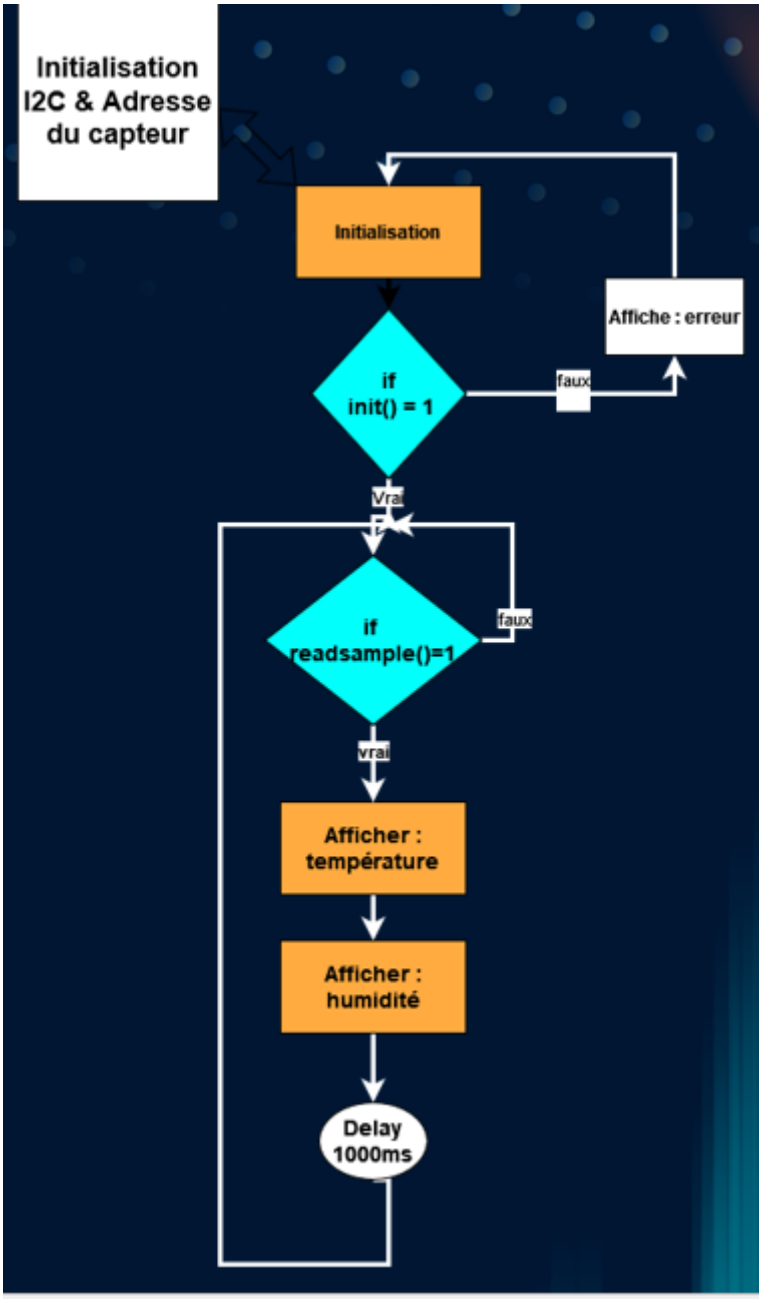


Figure 12 - Diagramme du programme permettant d'utiliser le capteur SHT

C. Firestore Database

Dans la continuité de notre projet de thermomètre connecté, nous avons opté pour l'utilisation de Firestore. C'est une base de données NoSQL hébergée par Google et est donc facile d'utilisation. Cette solution a été choisie pour sa capacité à stocker et synchroniser les données en temps réel entre nos utilisateurs. Firestore nous offre la flexibilité nécessaire pour évoluer avec le projet, une intégration aisée avec notre microcontrôleur ESP32, et une compatibilité avec notre client web. Ce dernier sert à visualiser les valeurs stockées dans la base de données ainsi qu'à afficher la météo en direct.

a. ESP32

L'ESP32 Firebeetle joue un rôle crucial dans notre dispositif il dirige à la fois le capteur comme vu précédemment mais est aussi la passerelle entre ce dernier et la base de données. Pour pouvoir stocker les données dans Firestore, on est passé par la librairie : *Firebase-ESP-Client* (source : *Github*).

Nous avons commencé par étudier attentivement le code exemple fourni avec cette librairie, ce qui nous a permis de comprendre les mécanismes fondamentaux pour créer et gérer des documents dans Firestore. Cette étape d'analyse a été cruciale pour acquérir une compréhension approfondie des opérations *Create*, *Read*, *Update* et *Delete*.

Après avoir assimilé les concepts de base, nous nous sommes attachés à personnaliser et à étendre le code exemple pour répondre aux exigences spécifiques de notre projet. Ce processus a inclus plusieurs étapes clés :

- **Création d'un document avec des valeurs arbitraires :** Nous avons modifié le code pour qu'il puisse envoyer des données avec l'architecture qui nous convenaient le mieux. C'est-à-dire, créer la collection, les documents et les variables avec les noms adéquat (voir fig.11) pour pouvoir les réutiliser.

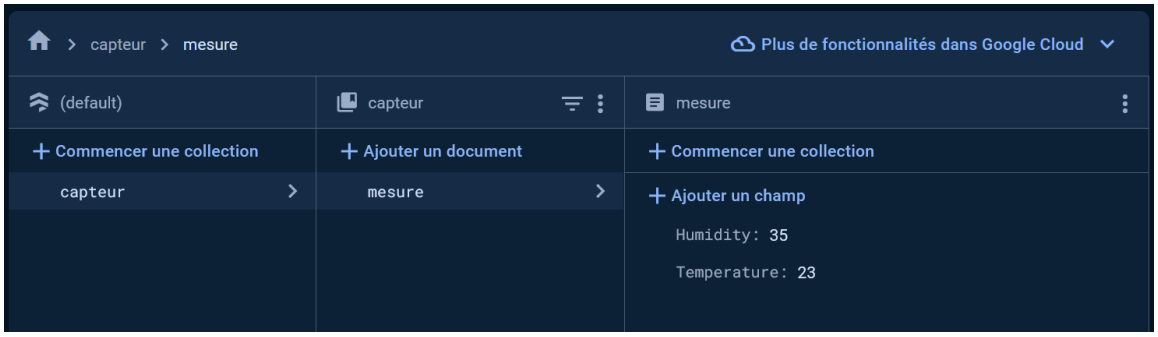


Figure 13 - Architecture personnalisée des documents Firestore

- **Adaptation au contexte du projet :** Dans un deuxième temps, nous avons ajouté le code de lecture du capteur pour avoir les bonnes données de température et d'humidité, les inclure dans la création du document, et les envoyer de à notre base de données Firestore. De plus, nous avons ajouté l'opération *Delete*. Donc l'ESP32 supprime le document s'il existe et en créer un nouveau avec le même nom.

En résumé, l'adaptation et l'extension du code exemple de la librairie Firestore ont été des étapes fondamentales de notre projet, nous permettant de mettre en place une base de données efficace. La dernière étape a été de pouvoir visualiser ces données sur un client web.

b. Client Web

Nous avons choisi de développer notre client web pour avoir une manière plus intuitive de consulter la température que de se connecter au site Firebase.

Pour le design global de notre application, nous voulions un arrangement simple, que tout soit bien lisible et surtout une construction axée sur la verticalité. L'objectif est de créer un outil facile d'utilisation, permettant à chacun de vérifier rapidement à la température de sa maison directement depuis son téléphone. L'interface se devait donc s'adapter à ce format. Cette dernière est conçue pour

être intuitive et épurée, avec deux sections principales :

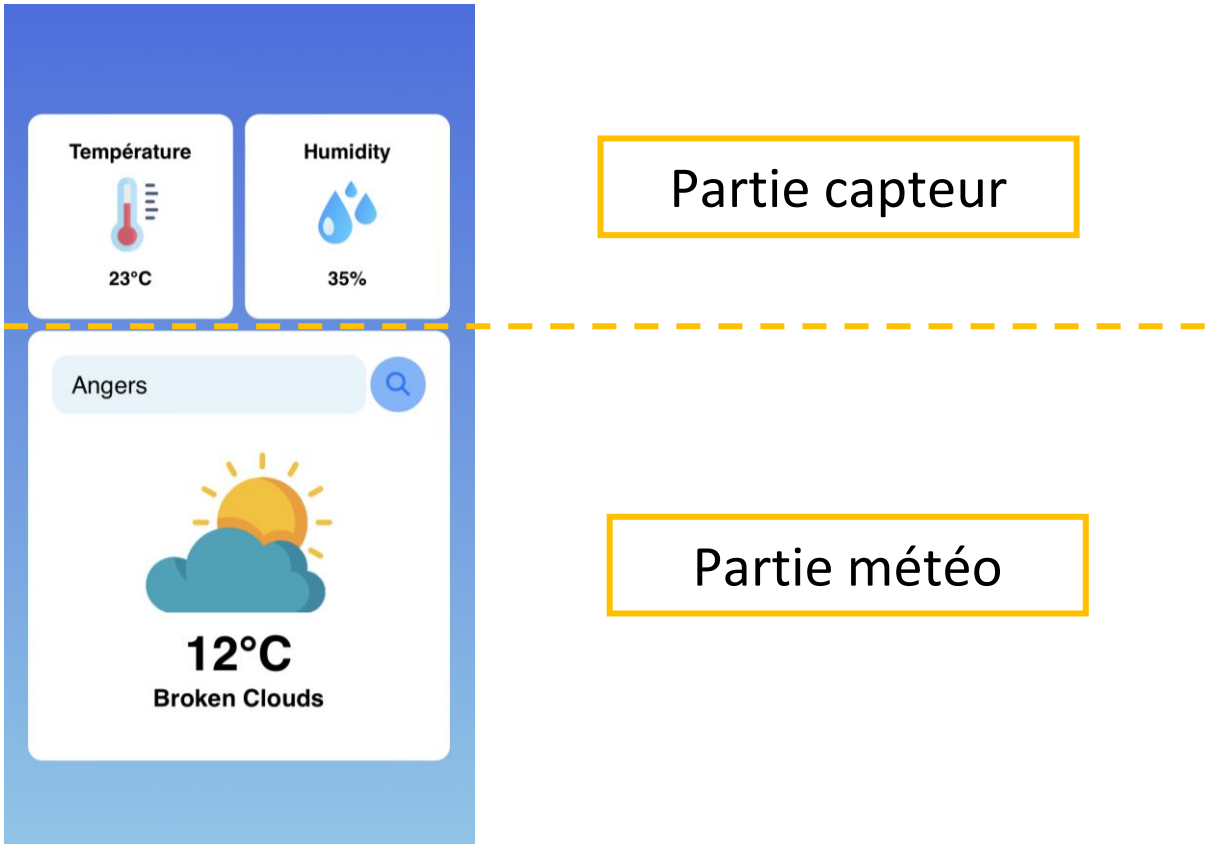


Figure 14 - Interface de l'application

Partie capteur :

Pour la récupération des valeurs depuis Firestore, notre client web utilise un script JavaScript pour établir une connexion avec la base de données Firestore puis lire les documents.

Établissement de la connexion : Le script initie d'abord une connexion avec Firestore en utilisant les identifiants et clés d'accès. Cette étape permet d'accéder à la base de données créée pour notre projet.

Lecture des documents : Une fois la connexion établie, le script exécute des requêtes pour lire les données stockées dans les documents de Firestore.

Traitement et affichage des données : Après avoir récupéré les données, le script les traite pour les convertir dans un format adapté à notre client web. Cette étape utilise Les Id en HTML pour intégrer les valeurs dans les balises souhaitées, ainsi les données sont transmises au front-end de notre client web, où elles sont intégrées dans l'interface utilisateur.

Partie météo en direct :

Pour avoir une application plus complète, nous avons ajouté une section où l'on peut consulter la météo en direct, n'importe où dans le monde. Nous avons, pour cela, utilisé un API qui nous permet de pourvoir accéder aux données météorologiques en direct.

Qu'est-ce qu'un API ?

Un API, ou Interface de Programmation d'Application (Application Programming Interface), est un ensemble de règles et de protocoles qui permettent à différents logiciels de communiquer entre eux. Dans notre cas, elle va interagir avec un service web. Pour être plus précis, un API permet à un programme informatique d'accéder à des fonctionnalités ou des données spécifiques, dans notre cas ce sont des informations météorologiques.

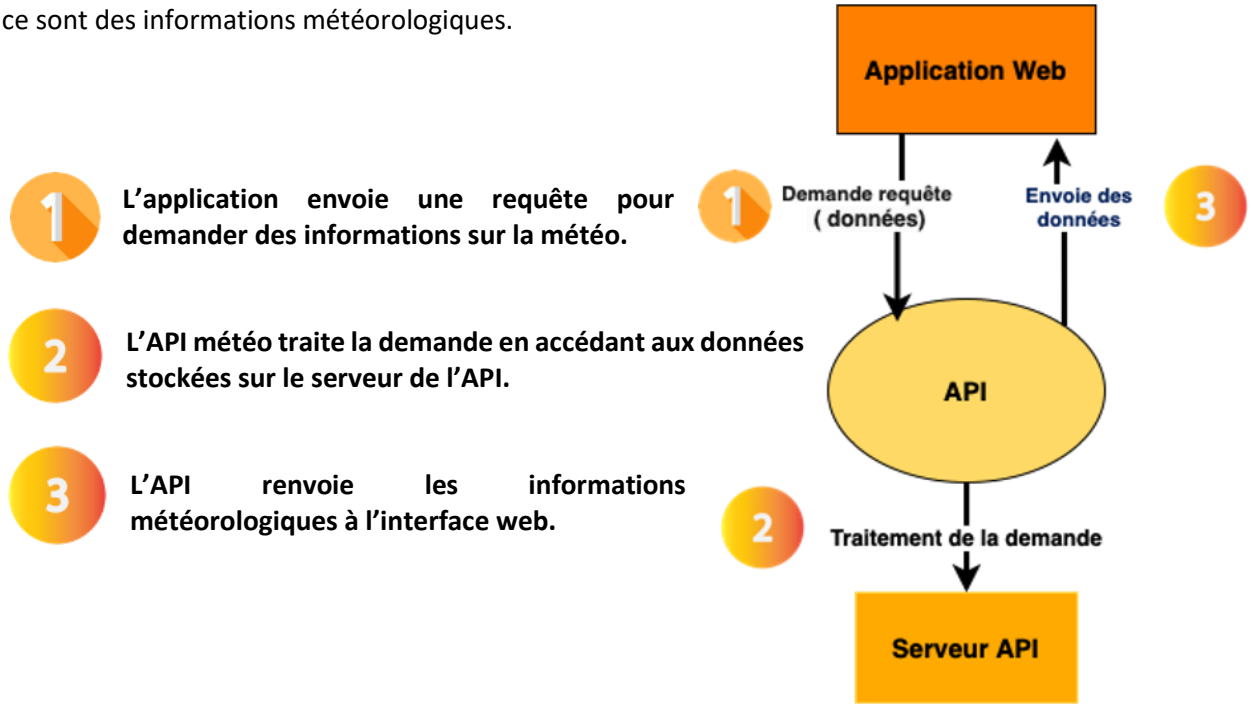


Figure 15 - Schéma d'explication d'un API

Pour notre projet nous avons utilisé l'API *OpenWeatherMap* (service qui fournit des données météorologiques, notamment des prévisions, des conditions actuelles et d'autres informations liées à la météo).

Explication du programme de l'API

Dans cette section, nous explorons le fonctionnement du programme conçu pour récupérer les données météorologiques de l'API *OpenWeatherMap* (voir fig.14). Le programme suit une série d'étapes logiques et structurées, que nous allons détailler ci-dessous :

Déclaration de la fonction asynchrone `checkWeather` :

Cette fonction est asynchrone, ce qui signifie qu'elle peut effectuer des opérations en arrière-plan tout en continuant à exécuter le reste du programme. Elle prend en paramètre la ville pour laquelle nous souhaitons obtenir la météo.

Création de la constante `api_key` :

Ici, nous définissons une constante, qui contient notre clé d'accès personnelle à l'API *OpenWeatherMap*, une clé qui nous est donné à la création de notre compte sur le site de l'API.

Construction de l'URL pour la requête API :

Cette étape consiste à assembler l'URL nécessaire pour interroger l'API. L'URL est construite en combinant la clé API et le nom de la ville spécifiée (le paramètre `city`).

Récupération des données météorologiques :

Nous déclarons une autre constante dans laquelle nous utilisons la fonction `fetch` pour envoyer une requête à l'URL préparée précédemment et attendons que la réponse soit disponible. Une fois la réponse obtenue, nous la convertissons au format JSON pour extraire les données météorologiques utiles.

```
async function checkWeather(city){
    const api_key = "d5004811908edd56905b81c3a3639902";
    const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${api_key}`;

    const weather_data = await fetch(`${url}`).then(response => response.json());
```

Figure 16 - Programme de l'utilisation de l'API

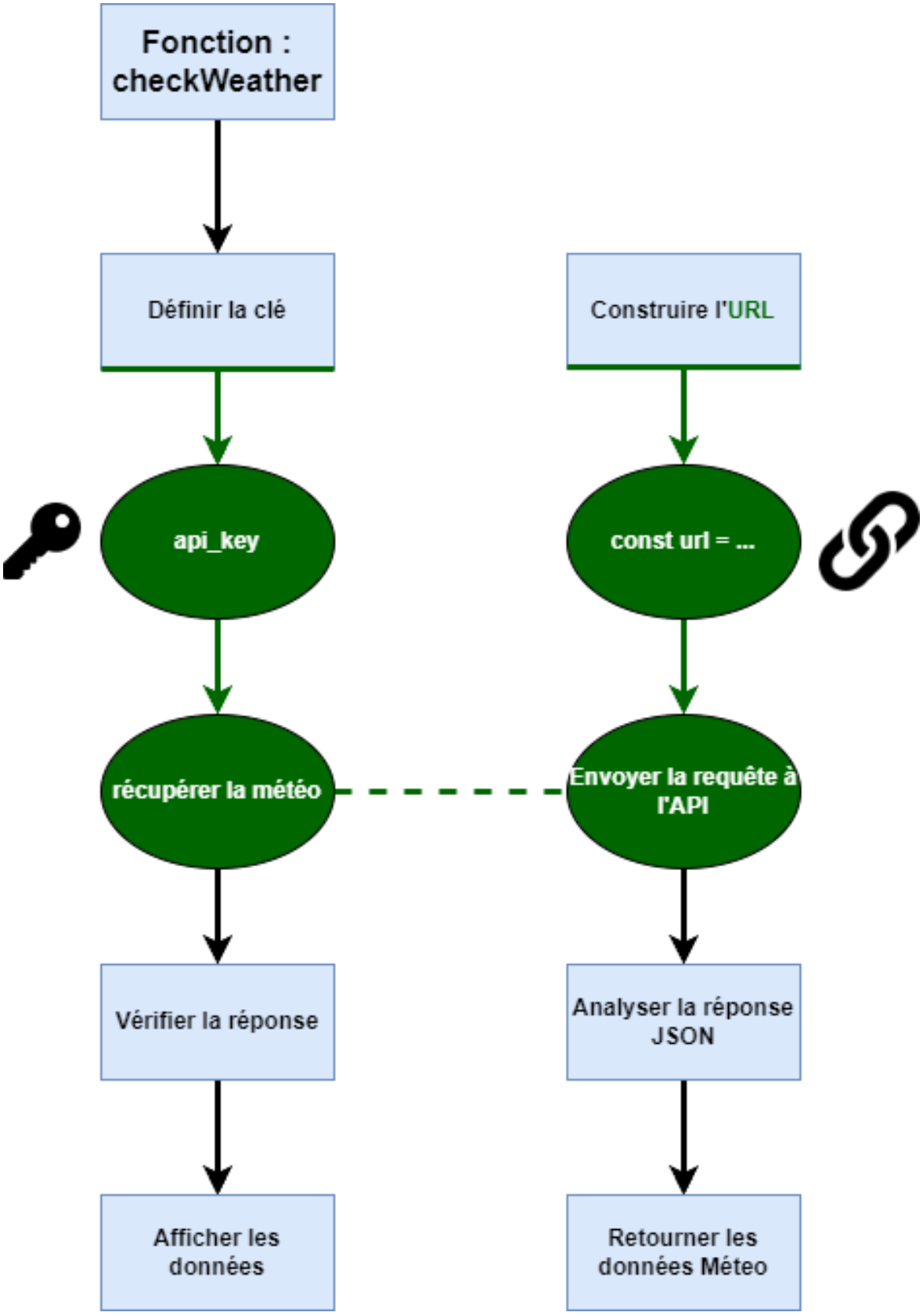


Figure 17 - diagramme explicatif du code pour l'utilisation de l'API

4. Perspectives d'améliorations

Notre projet, bien qu'efficace dans sa forme actuelle, offre plusieurs opportunités d'amélioration pour accroître sa performance, sa sécurité et son esthétique. Voici les améliorations que nous envisageons :

Sécurité Firestore

Actuellement, notre accès à Firestore est sécurisé, mais il existe des opportunités d'améliorer davantage cette sécurité, par exemple le renforcement des règles Firestore. Nous prévoyons de revoir les règles d'accès dans l'onglet "Rules" de Firestore. Cela inclura des contrôles plus stricts sur qui peut lire ou écrire dans la base de données, et sous quelles conditions.

Boitier

Nous envisageons de concevoir et de réaliser un boîtier 3D sur mesure pour notre thermomètre
connecté. Ce boîtier offrira non seulement une protection physique pour les composants, mais ajoutera également une touche esthétique. Le choix des matériaux et le design du boîtier seront étudiés pour optimiser la durabilité, l'esthétique et la fonctionnalité.

Consommation Énergétique

Une amélioration majeure concernera la gestion de la consommation énergétique. Nous envisageons d'intégrer une fonction de mise en veille ("*sleep*") entre la création de deux documents. Cette fonction permettra de réduire considérablement la consommation énergétique de l'ESP32, prolongeant ainsi l'autonomie de l'appareil lorsqu'il fonctionne sur batterie.

Ces améliorations visent à renforcer la sécurité de nos données, à améliorer l'aspect esthétique et pratique de notre dispositif, et à optimiser sa consommation énergétique. Elles sont essentielles pour rendre notre projet non seulement plus efficace, mais aussi plus attractif et durable.

Conclusion

En conclusion, ce projet de thermomètre connecté représente une avancée significative dans notre compréhension et notre application de la technologie IoT (Internet des Objets). En surmontant les défis techniques, nous avons développé un système capable de mesurer, de stocker et de visualiser des données de température, tout en offrant des informations météorologiques en direct.

L'expérience acquise dans la programmation de l'ESP32, la gestion des données avec Firestore et le développement d'un client web fut importante. Ces compétences reflètent non seulement notre capacité à travailler avec des technologies avancées, mais aussi notre aptitude à concevoir et implémenter des solutions intégrées répondant à des besoins réels.

Les perspectives d'amélioration identifiées, telles que le renforcement de la sécurité de Firestore, la conception d'un boîtier 3D pour le dispositif, et l'optimisation de la consommation énergétique, tracent la voie pour l'évolution future du projet. Ces améliorations contribueront à la réalisation d'un produit non seulement fonctionnel mais aussi sécurisé, esthétiquement plaisant et en lien avec les enjeux environnementaux actuels.

Ce projet a démontré que, grâce à une collaboration efficace et une approche méthodique, il est possible de transformer une idée conceptuelle en un prototype fonctionnel et prometteur. Il pose les fondations pour de futurs développements dans le domaine de l'IoT et ouvre des perspectives passionnantes pour des applications similaires.

En somme, nous sommes fiers de ce que nous avons accompli et enthousiastes à l'idée de poursuivre notre projet .

Annexes

Programme du Capteur

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

#include "SHTSensor.h"

// put function declarations here:
SHTSensor sht41(SHTSensor::SHT4X);


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

  Wire.begin();           // Initialise la liaison I2C avec l'ESP 32 comme
maitre
  Serial.begin(115200);   // Initialise la liaison série à 115200 bauds
  delay(2000);            // let serial console settle

  //Autodetect Sensor Type at initialisation
  if (sht41.init())
  {
    //Lorsque la fonction init() est terminée avec succès, elle renvoie 1
    Serial.print("Setup finished");
  }
  else
  {
    //Renvoi un message d'erreur lorsque le capteur a rencontré un problème à
l'initialisation
    Serial.print("Setup failed");
  }

}

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

  Serial.print("SHT41 Mesurement:");

  if (sht41.readSample()){
    //Si la fonction readSample() = 1, On mesure l'humidité et la température
    Serial.print("Temperature: ");
    Serial.print(sht41.getTemperature(), 2);//Affichage de la température
    Serial.println("Humidity: ");
    Serial.print(sht41.getHumidity(), 2);   //Affichage de l'humidité
    delay(1000);        // Délai d'attente entre 2 mesures
```

```
  }
  else{
    //Envoi d'un message d'erreur lorsque la mesure à rencontré un problème
    Serial.print("Error");
  }

}

// put function definitions here:
```

Programme de l'envoi des données dans Firestore

```
/**
 * Created by K. Suwatchai (Mobizt)
 *
 * Email: k_suwatchai@hotmail.com
 *
 * Github: https://github.com/mobizt/Firebase-ESP-Client
 *
 * Copyright (c) 2023 mobizt
 *
 */

// This example shows how to create a document in a document collection. This
operation required Email/password, custom or OAUth2.0 authentication.

#include <Arduino.h>
#if defined(ESP32) || defined(ARDUINO_RASPBERRY_PI_PICO_W)
#include <WiFi.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#elif __has_include(<WiFiNINA.h>)
#include <WiFiNINA.h>
#elif __has_include(<WiFi101.h>)
#include <WiFi101.h>
#elif __has_include(<WiFiS3.h>)
#include <WiFiS3.h>
#endif

#include <Firebase_ESP_Client.h>

#include "SHTSensor.h"

// put function declarations here:
SHTSensor sht41(SHTSensor::SHT4X);

// Provide the token generation process info.
#include <addons/TokenHelper.h>

/* 1. Define the WiFi credentials */
#define WIFI_SSID "wifirobot"
#define WIFI_PASSWORD "robot2004LARIS"

/* 2. Define the API Key */
#define API_KEY "AIzaSyBZJn1bZEzJoL-cUIEIhVkrSTDFyuHtnfA"

/* 3. Define the project ID */
```

```
#define FIREBASE_PROJECT_ID "sae-s5-2023"

/* 4. Define the user Email and password that alreadey registerd or added in
your project */
#include "userMail.h"

// Define Firebase Data object
FirebaseData fbdo;

FirebaseAuth auth;
FirebaseConfig config;

unsigned long dataMillis = 0;
int count = 0;

#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
WiFiMulti multi;
#endif

// The Firestore payload upload callback function

void fcsUploadCallback(CFS_UploadStatusInfo info)
{
    if (info.status == firebase_cfs_upload_status_init)
    {
        Serial.printf("\nUploading data (%d)...\n", info.size);
    }
    else if (info.status == firebase_cfs_upload_status_upload)
    {
        Serial.printf("Uploaded %d%s\n", (int)info.progress, "%");
    }
    else if (info.status == firebase_cfs_upload_status_complete)
    {
        Serial.println("Upload completed ");
    }
    else if (info.status == firebase_cfs_upload_status_process_response)
    {
        Serial.print("Processing the response... ");
    }
    else if (info.status == firebase_cfs_upload_status_error)
    {
        Serial.printf("Upload failed, %s\n", info.errorMsg.c_str());
    }
}

void setup()
{
```

```
    Serial.begin(9600);

#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
    multi.addAP(WIFI_SSID, WIFI_PASSWORD);
    multi.run();
#else
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
#endif

    Serial.print("\nConnecting to Wi-Fi");
    unsigned long ms = millis();
    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.print(".");
        delay(300);
#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
        if (millis() - ms > 10000)
            break;
#endif
    }
    Serial.println();
    Serial.print("Connected with IP: ");
    Serial.println(WiFi.localIP());
    Serial.println();

    Serial.printf("Firebase Client v%s\n\n", FIREBASE_CLIENT_VERSION);

    /* Assign the api key (required) */
    config.api_key = API_KEY;

    /* Assign the user sign in credentials */
    auth.user.email = USER_EMAIL;
    auth.user.password = USER_PASSWORD;

    // The WiFi credentials are required for Pico W
    // due to it does not have reconnect feature.
#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
    config.wifi.clearAP();
    config.wifi.addAP(WIFI_SSID, WIFI_PASSWORD);
#endif

    /* Assign the callback function for the long running token generation task
*/
    config.token_status_callback = tokenStatusCallback; // see
addons/TokenHelper.h

    // Comment or pass false value when WiFi reconnection will control by your
code or third party library e.g. WiFiManager
```

```
    Firebase.reconnectNetwork(true);

    // Since v4.4.x, BearSSL engine was used, the SSL buffer need to be set.
    // Large data transmission may require larger RX buffer, otherwise
connection issue or data read time out can be occurred.
    fbdo.setBSSLBufferSize(4096 /* Rx buffer size in bytes from 512 - 16384
*/, 1024 /* Tx buffer size in bytes from 512 - 16384 */);

    // Limit the size of response payload to be collected in FirebaseData
    fbdo.setResponseSize(2048);

    Firebase.begin(&config, &auth);

    //=============================================================================
    //============================ SETUP CAPTEUR ==================================
    //=============================================================================

    Wire.begin(); // Initialise la liaison I2C avec l'ESP 32 comme maitre
    delay(2000);  // let serial console settle

    Serial.print("\nConnecting to Sensor");

    // Autodetect Sensor Type at initialisation
    if (sht41.init())
    {
        // Lorsque la fonction init() est terminée avec succès, elle renvoie 1
        Serial.print("\nSetup finished");
    }
    else
    {
        // Renvoi un message d'erreur lorsque le capteur a rencontré un
problème à l'initialisation
        Serial.print("Setup failed");
    }
}

void loop()
{

    int tempvalue;
    int humvalue;

    // Firebase.ready() should be called repeatedly to handle authentication
tasks.
    if (sht41.readSample())
    { // Si la fonction readSample() = 1, On mesure l'humidité et la
température
```

```
        if (Firebase.ready() && (millis() - dataMillis > 10000 || dataMillis
== 0))
        {

            Serial.print("SHT41 Mesurement:");

            dataMillis = millis();

            // For the usage of FirebaseJson, see
examples/FirebaseJson/BasicUsage/Create_Edit_Parse/Create_Edit_Parse.ino
            FirebaseJson content;

            String documentPath = "capteur/mesure";

            Serial.print("\nTemperature: ");
            tempvalue = sht41.getTemperature() - 6;
            Serial.print(tempvalue, 2); // Affichage de la température

            Serial.println("\nHumidity: ");
            humvalue = sht41.getHumidity();
            Serial.print(humvalue, 2); // Affichage de l'humidité

            content.set("fields/Temperature/doubleValue", tempvalue);
            content.set("fields/Humidity/doubleValue", humvalue);

            Serial.print("Delete a document... ");

            if (Firebase.Firestore.deleteDocument(&fbdo, FIREBASE_PROJECT_ID,
"" /* databaseId can be (default) or empty */, documentPath.c_str()))
                Serial.printf("ok\n%s\n\n", fbdo.payload().c_str());
            else
                Serial.println(fbdo.errorReason());


            Serial.print("Create a document... ");

            if (Firebase.Firestore.createDocument(&fbdo, FIREBASE_PROJECT_ID,
"", documentPath.c_str(), content.raw()))
                Serial.printf("ok\n%s\n\n", fbdo.payload().c_str());
            else
                Serial.println(fbdo.errorReason());
        }
    }
    else
    {
        // Envoi d'un message d'erreur lorsque la mesure à rencontré un
problème
        Serial.print("Error");
```

```
    }
}
```

Programme récupération des données Firestore

```
import { initializeApp } from
"https://www.gstatic.com/firebasejs/10.7.0/firebase-app.js";
import {
  getFirestore,
  doc,
  getDoc,
} from "https://www.gstatic.com/firebasejs/10.7.0/firebase-firestore.js";

// Paramètres du projet Firebase
let firebaseConfig = {
  apiKey: "AIzaSyBZJn1bZEzJoL-cUIEIhVkrSTDFyuHtnfA",
  authDomain: "sae-s5-2023.firebaseapp.com",
  projectId: "sae-s5-2023",
  storageBucket: "sae-s5-2023.appspot.com",
  messagingSenderId: "671913288675",
  appId: "1:671913288675:web:512c32227f22166bc136d6",
};

// Initialisation Firebase
const app = initializeApp(firebaseConfig);

// Initialize Cloud Firestore and get a reference to the service
const db = getFirestore(app);

console.log("db", db);
console.log("app", app);

const docRef = doc(db, "capteur", "mesure"); // Nom de la collection et du
document où il faut récupérer les valeurs
const docSnap = await getDoc(docRef); // Récupération de documents

if (docSnap.exists()) { // Si le document existe au nom indiqué
  console.log("Document data:", docSnap.data());
  const tempValue = docSnap.data().Temperature; // Création de variable avec
les valeurs stocké dans firestore
  const humValue = docSnap.data().Humidity; // Création de variable avec les
valeurs stocké dans firestore

  // Utilisation des valeurs dans le site web avec les ID
  document.getElementById("tempID").innerText = tempValue + "°C";
  document.getElementById("humID").innerText = humValue + "%";

  console.log("Temperature:", tempValue); // Ecriture des valeurs dans la
console
  console.log("Humidity:", humValue); // Ecriture des valeurs dans la console
} else {
```

```
  console.log("No such document!"); // S'il n'y a pas de document
}
```

Programme de L'API météo

```
const inputBox = document.querySelector('.input-box');
const searchBtn = document.getElementById('searchBtn');
const weather_img = document.querySelector('.weather-img');
const temperature = document.querySelector('.temperature');
const description = document.querySelector('.description');

// ================== SCRIPT METEO ==================

async function checkWeather(city){
    const api_key = "d5004811908edd56905b81c3a3639902";
    const url =
`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${api_key}`;

    const weather_data = await fetch(`${url}`).then(response =>
response.json());


    //console.log(weather_data);
    temperature.innerHTML = `${Math.round(weather_data.main.temp -
273.15)}°C`; // Affichage des degrés en Celsius
    description.innerHTML = `${weather_data.weather[0].description}`;

    switch(weather_data.weather[0].main){ // Changement d'image en fonction du
temps
        case 'Clouds':
            weather_img.src = "/assets/cloud.png";
            break;
        case 'Clear':
            weather_img.src = "/assets/clear.png";
            break;
        case 'Rain':
            weather_img.src = "/assets/rain.png";
            break;
        case 'Mist':
            weather_img.src = "/assets/mist.png";
            break;
        case 'Snow':
            weather_img.src = "/assets/snow.png";
            break;
        case 'thunderstorm':
            weather_img.src = "/assests/thunder.png";
            break;
    }
}
```

```
inputBox.addEventListener('keydown', (event) => { // fonctionnement bouton
entrer pour rechercher
    if (event.code === 'Enter') {
        event.preventDefault();
        checkWeather(inputBox.value);
    }
});

searchBtn.addEventListener('click',()=>{ // fonctionnement bouton loupe pour
rechercher
    checkWeather(inputBox.value);

});
```

Résumé

Le projet de capteur de température est un projet de SAE centré autour de l'envoi de données vers un serveur Web. Son but est d'envoyer les données de température d'une pièce vers une base de données à l'aide d'un microcontrôleur ESP32. Une fois ces données enregistrées celles-ci sont affichés sur un client Web afin d'être accessible de partout.

The temperature sensor project is a third-year student project focused on sending data to a web server. His goal is to send temperature data from room to a database with a micro controller ESP32. Once these data are stored in the database, they are displayed on a Web client and are accessible from everywhere.