Projet Cuve

SAE 5 : Récupération Température et Niveau Cuve

Réalisé par :

OZDEMIR Semih G330

VAUSORT Sébastien G330

GIRARD Louis G330

Tuteur : LUCIDARME Philippe

université angers

iut Angers Cholet

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

UNIVERSITÉ D'ANGERS

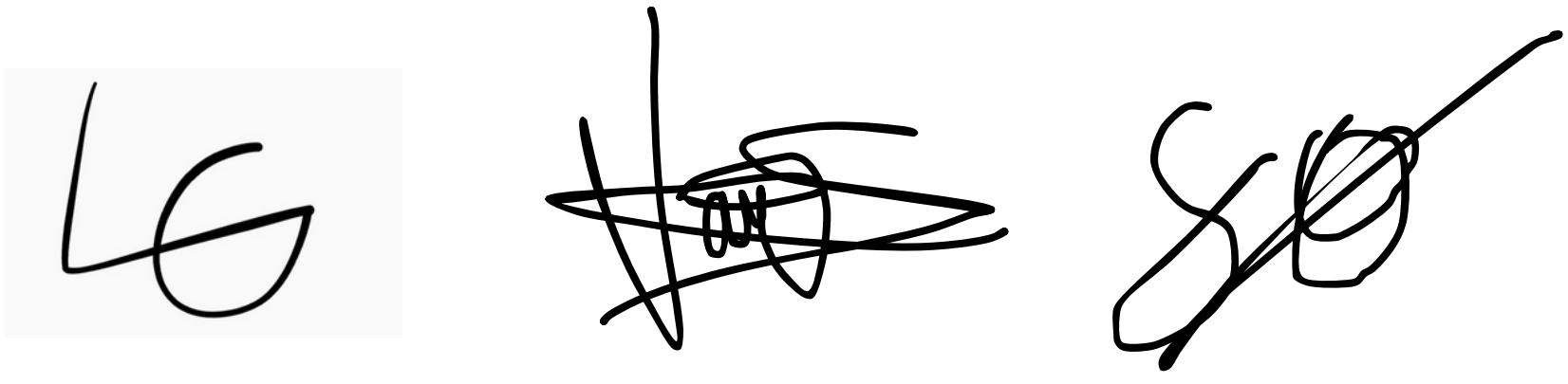
Campus de Belle-Beille (Angers)

4, boulevrard Lavoisier - BP 42018

49016 - ANGERS Cedex

Tél. : 02 44 68 87 00

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.



Remerciements

Nous tenons à exprimer notre profonde gratitude et nos sincères remerciements à toutes les personnes qui ont contribué à la réussite de notre projet au sein de l'IUT d’Angers. Leur soutien inestimable, leur expertise et leurs encouragements ont grandement enrichi notre expérience.

Nous tenons tout d'abord à remercier chaleureusement notre tuteur de projet, M.LUCIDARME, pour sa guidance et ses précieux conseils tout au long de notre projet. Sa patience, son professionnalisme et sa disponibilité ont été d'une aide précieuse dans la réalisation de nos missions et dans notre développement personnel.

Nous souhaitons exprimer notre reconnaissance envers tous nos collègues de travail pour leur soutien et leurs conseils précieux. Leurs expériences partagées, leurs encouragements et leur volonté de nous aider nous ont grandement facilité.

Nous tenons à remercier l'ensemble du personnel de l’IUT particulièrement Mme.DENECHEAU pour son amabilité et sa coopération.

Enfin, nous tenons à remercier nos familles et nos amis pour leur soutien indéfectible tout au long de notre parcours académique et professionnel. Leur encouragement constant et leur soutien moral ont été essentiels pour mener à bien notre projet.

Sommaire

Tables des Figures

Introduction

L’eau recouvre 72% de la surface terrestre, c’est pour cette raison que l’on appelle la Terre planète Bleu. Malgré l’importante quantité d’eau seulement 0,7% peut être consommé par l’Homme, l’eau salée représente 97,2% et les glaces polaires représentent 2,1%.

L’eau disponible n’est donc qu’une infime partie de ce que l’on peut consommer. Il est donc important de préserver et de recycler l’eau douce afin de ne pas connaître de saisons de sécheresse.

Il existe de plus en plus de solutions pour préserver, recycler l’eau comme les usines de traitements d’eaux, les usines de dessalements, mais aussi de solutions plus simples et accessibles pour une partie des citoyens français qui sont les cuves de récupérations d’eau.

Pour des raisons d’esthétique et de gains de place, beaucoup de personnes enterrent les cuves. De plus, elle permet d’être plus écologique en se fournissant sa propre eau, et permet une rentabilité financière au fil des années. Une problématique se présente, il devient donc compliqué de savoir le niveau exact d’eau ainsi que sa température. Une des solutions pour remédier à ce problème est de transformer cette cuve en cuve connectée.

Acheter une cuve connectée représente un certain coût, mais il est possible d’ajouter un module connecté. Notre projet a été de réaliser ce module qui rend la cuve connectée.

I. Cahier des Charges

1. Spécifications Fonctionnelles

a. Mesures du niveau de la cuve

Le capteur de niveau est primordial afin de mesurer le niveau de l'eau dans la cuve, il existe plusieurs sortes de capteurs qui sont capables de réaliser cette fonction. Par ailleurs, nous avons fait le choix de partir sur un capteur plus précisément qui sera présenter dans la partie "Choix technologiques". On trouve ci-dessous le capteur ultrason, celui-ci est le modèle SEN0311.

Le capteur ultrason doit émettre des ondes sonores, ensuite, il mesure le temps nécessaire à leur retour après réflexion sur un objet, puis calcule la distance en utilisant la formule de base de la vitesse du son (d = (v*t)/2). Il génère donc un signal de sortie proportionnel à la distance mesurée, afin d'obtenir la distance de l'obstacle.

Figure 1: Capteur de niveau

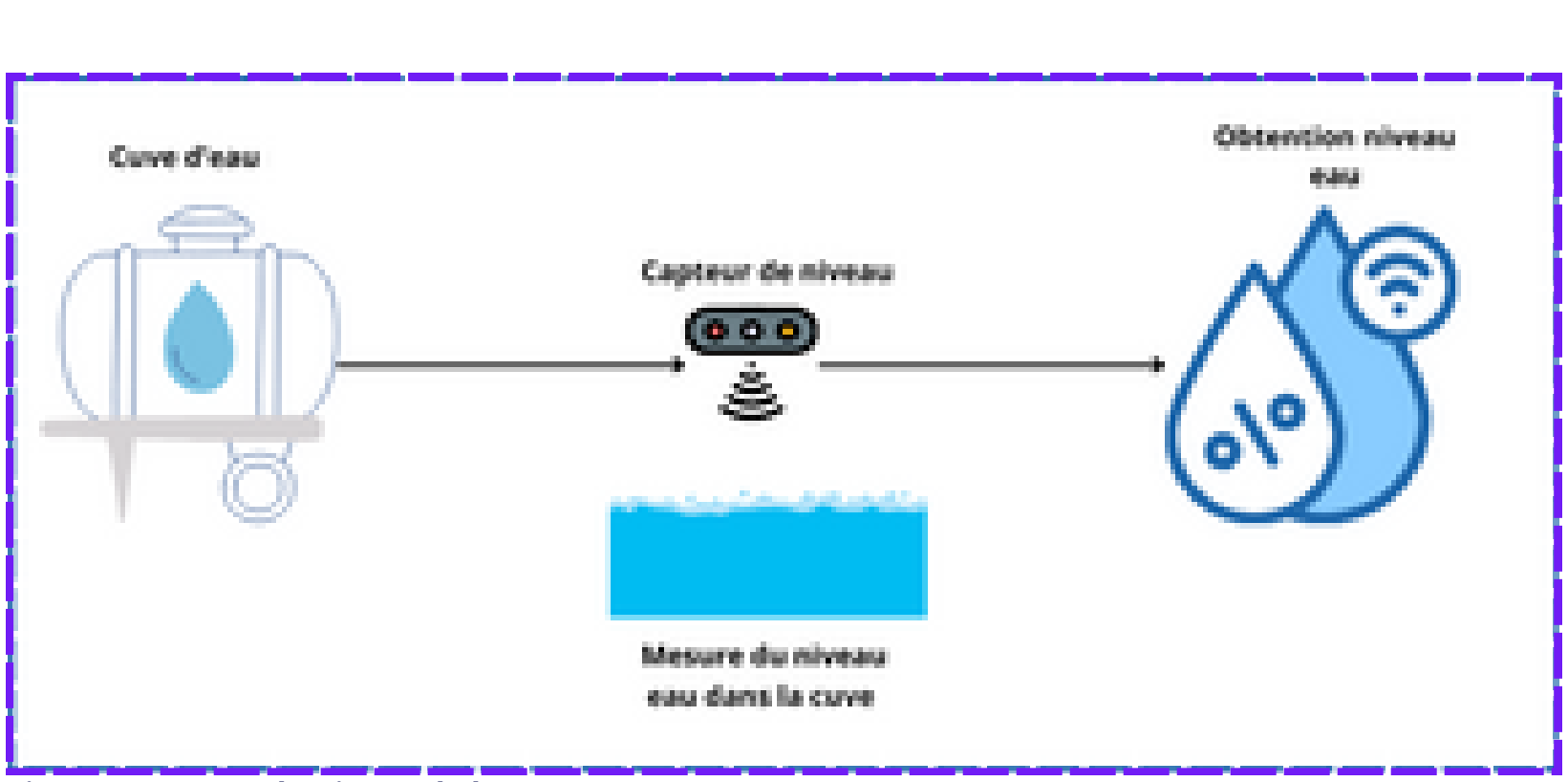


Figure 2: Mesure du niveau de la cuve

b. Mesures de la température

Pour mesurer la température ambiante dans la cuve, nous n'avons pas eu d'autres choix que d'utiliser un capteur de température. Comme pour le capteur ultrason nous avons fait un choix afin de choisir le capteur de température le plus adapté à notre besoin.

Le capteur de température SHT41 utilise une thermistance pour mesurer la résistance électrique, qui varie en fonction de la température. En convertissant ces données, le capteur fournit une sortie numérique précise représentant la température ambiante.

Figure 3 : Capteur de température

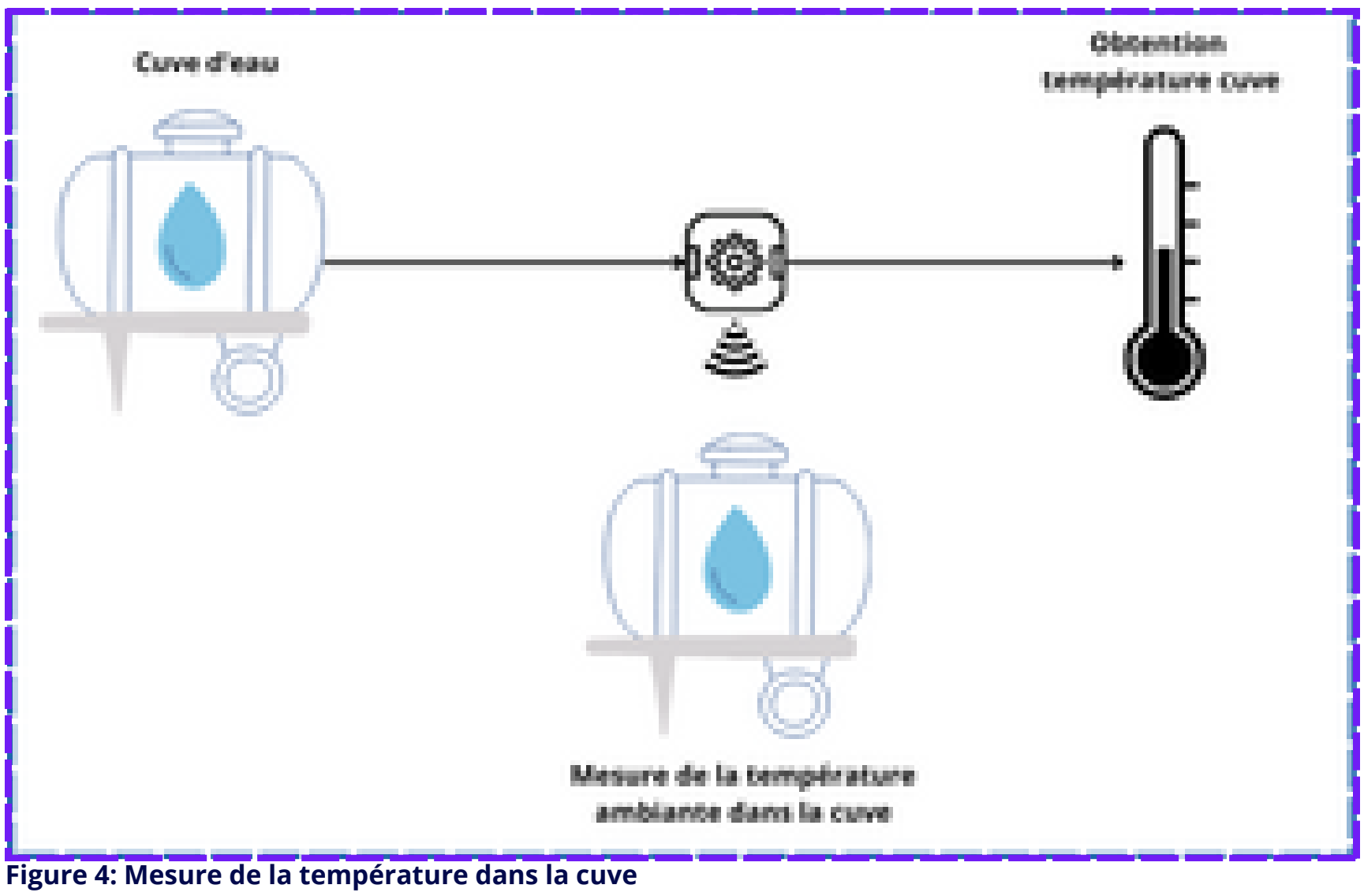


Figure 4: Mesure de la température dans la cuve

2. Spécifications Techniques

a. Contraintes et normes à respecter

Contraintes fonctionnelles :
Le système conçu doit permettre à l'utilisateur de superviser le volume d'eau dans la cuve ainsi que la température et l'humidité ambiante sans avoir à accéder à la cuve.
Le tout doit être utilisable de manière intuitive.

Contraintes mécaniques :
Le système doit résister à diverses conditions météorologiques telles que le froid, l'humidité, la chaleur. Sa taille doit être réduite afin d'être parfaitement intégré dans le regard de la cuve.

Contraintes énergétiques :
La cuve étant enterrée dans un espace extérieur, le système doit s'équiper de sa propre source d'alimentation. Et, afin d'éviter à l'utilisateur d'accéder à l'intérieur de la cuve, le système doit présenter une autonomie importante afin de ne pas changer fréquemment la pile ou la batterie du dispositif (supérieure à un an).

Contraintes matérielles et logicielles :
Le projet visera le développement d'une base de données (Firebase) pour stocker les grandeurs mesurées et le développement d'une interface web qui permet d'afficher les mesures et offre à l'utilisateur la possibilité de piloter le système.

Contraintes temporelles :
Cinquante heures sont consacrées à ce projet. Si besoin, ce dernier pourra être poursuit au semestre 6.

3. Gestion de projet

a. Diagramme de Gantt

Au cours de ce projet, nous avons partagé les différentes tâches entre nous. Nous nous sommes très rapidement rendu compte que le premier diagramme de Gantt que nous avons réalisé n'était pas tout à fait réalisable du fait de la contrainte de temps que nous avions.

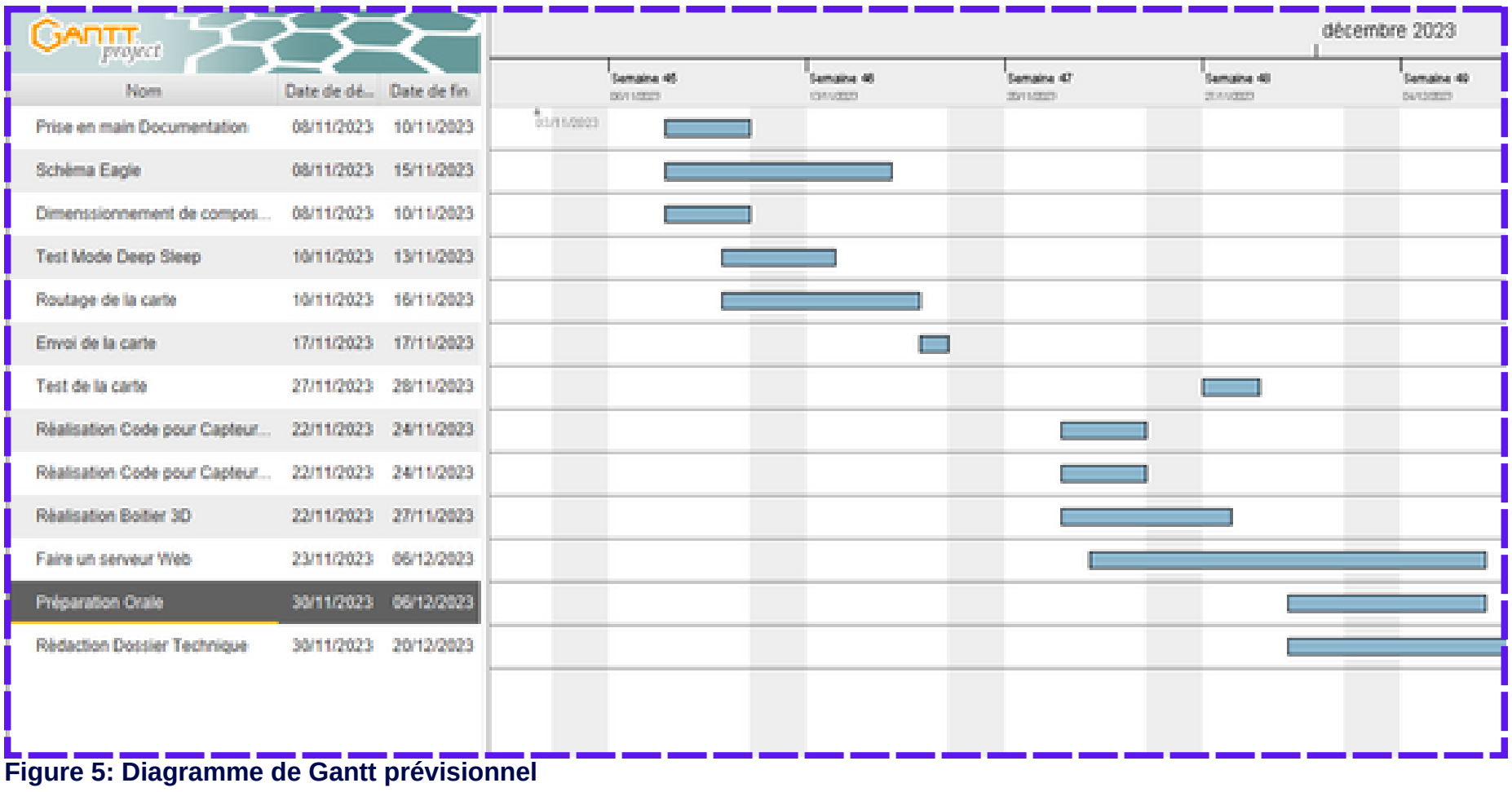


Figure 5: Diagramme de Gantt prévisionnel

Nous avons débuté par prendre en main les différentes documentations principalement celles des capteurs afin de choisir judicieusement les bons composants avec la meilleure précision. En parallèle, nous avons réalisé le schéma Eagle avec le dimensionnement des composants. Une fois que ces tâches ont été réalisées, nous avons écrit le code pour tester le mode deep-sleep afin de pouvoir déterminer l'état de la LED pendant le deep-sleep. Une fois le routage réalisé et la validation de M.LUCIDARME, nous avons envoyé les schémas de la carte au PCB Maker. A son retour nous l'avons testé afin de contrôler s'il n'y a pas de court-circuit, par ailleurs sur cette carte nous n'avons pas trouvé de défaut. En parallèle, nous avons réalisé les différents codes tels que le code pour le capteur de température et le capteur ultrason. Enfin, nous avons développé un serveur Web.

Comme dit auparavant, nous avons décidé de ne pas réaliser un de nos objectifs qui était de réaliser un boîtier 3D pour que le module connecté soit protégé de l'humidité. Nous avons préféré de nous concentrer sur les autres missions afin d'avoir un projet finis et complet.

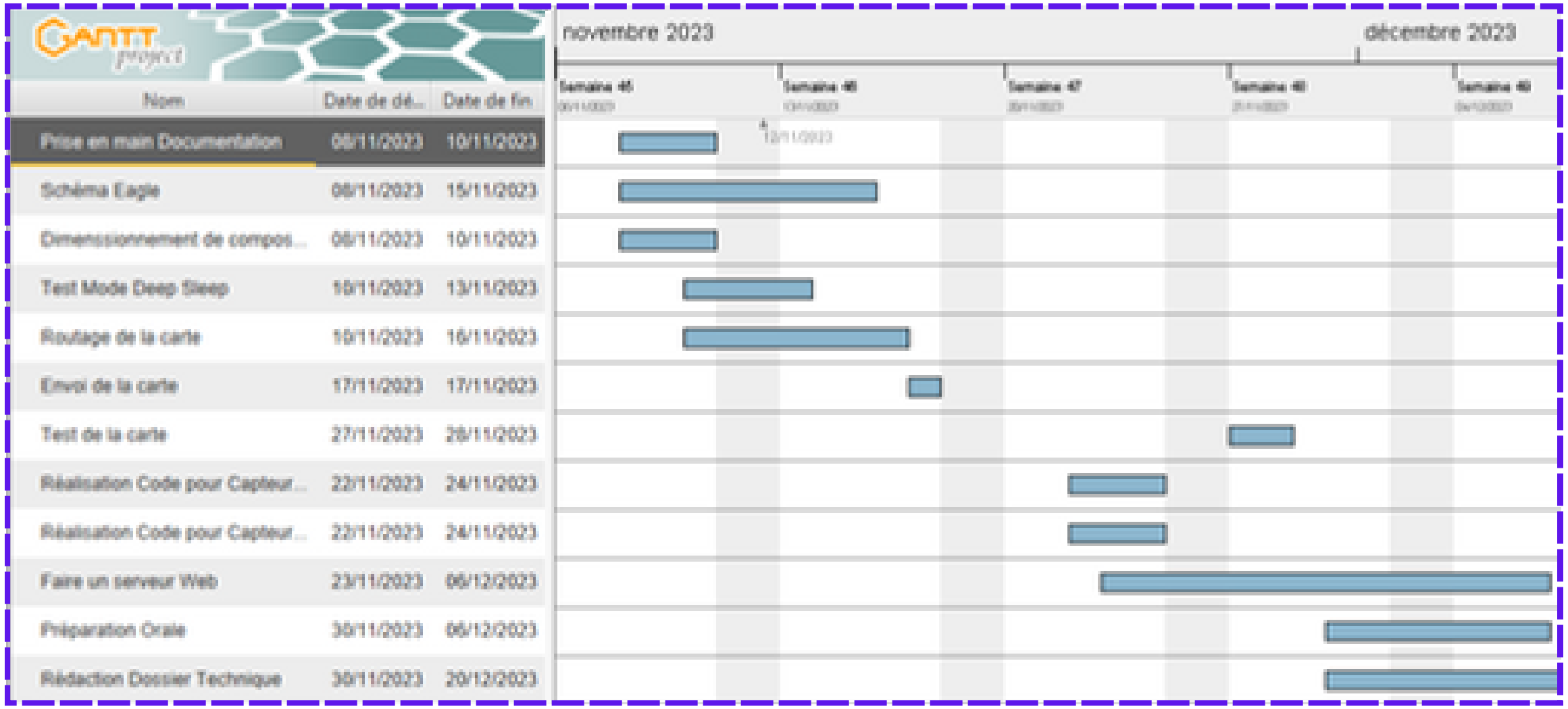


Figure 6: Diagramme de Gantt réalisé

II. Choix Technologiques

1. Choix des capteurs

a. Capteur Ultrason

Figure 7: Capteur US - SEN0311

Capteur ultrason :

Nous avons opté pour le SEN0311 qui présente une consommation réduite (<8mA), peut être alimenté en 3,3V ou 5V, est étanche et peut mesurer des distances allant de 0,03 à 4,5m qui recouvre donc la plage de profondeur de la cuve. Ce capteur peut dialoguer avec le microcontrôleur par liaison série (RX/TX) et possède de librairies qui permettent de le piloter facilement et de récupérer les grandeurs mesurées avec le microcontrôleur.

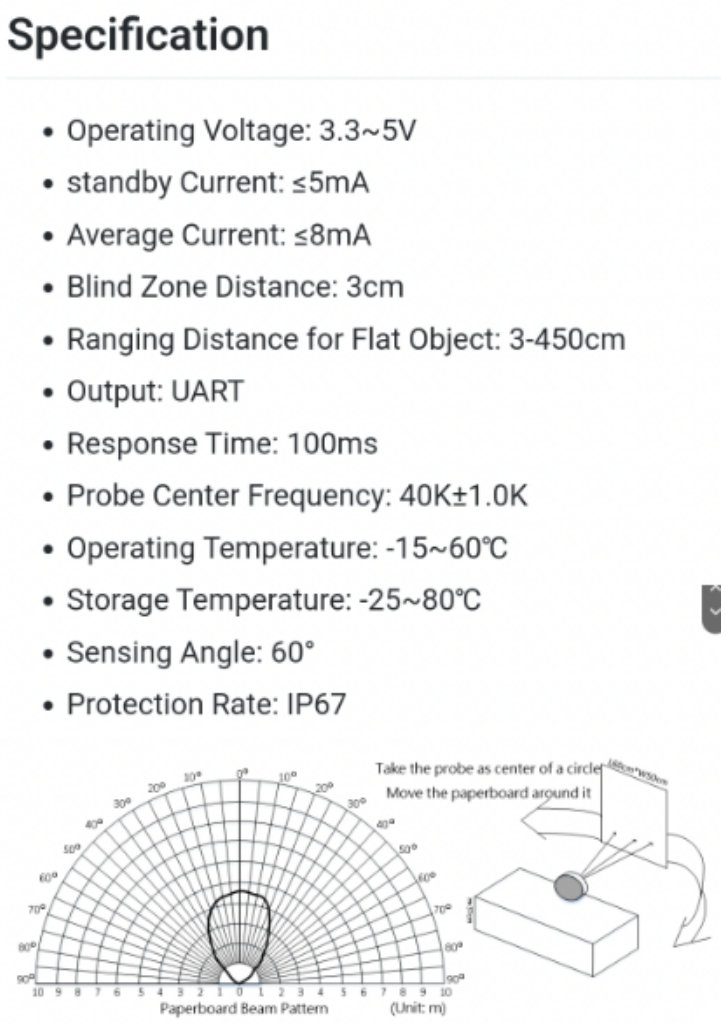


Figure 8: Specifications- SEN0311

b. Capteur de température

Figure 9: Capteur de température - SHT41

Capteur de température :

Le capteur choisi est le SHT41, qui est le capteur le moins énergivore de sa série (0,4 µA) et est relativement précis (+/-1,8% RH pour l'humidité et +/- 0,2°C). Enfin, il présente un faible encombrement ce qui lui permet d'être directement intégré sur la carte et permet également de mesurer l'humidité. Son protocole de communication est l'I2C. Ici encore des librairies existent pour le piloter facilement et acquérir les grandeurs mesurées.

2. Choix des composants

a. Leds

Pour la partie IHM, nous avons choisi d'implanter des LEDs sur la carte à 90° qui auront pour but de témoigner de l'état de la batterie ou d'autres informations, ainsi que des boutons-poussoir pour diverses fonctions.

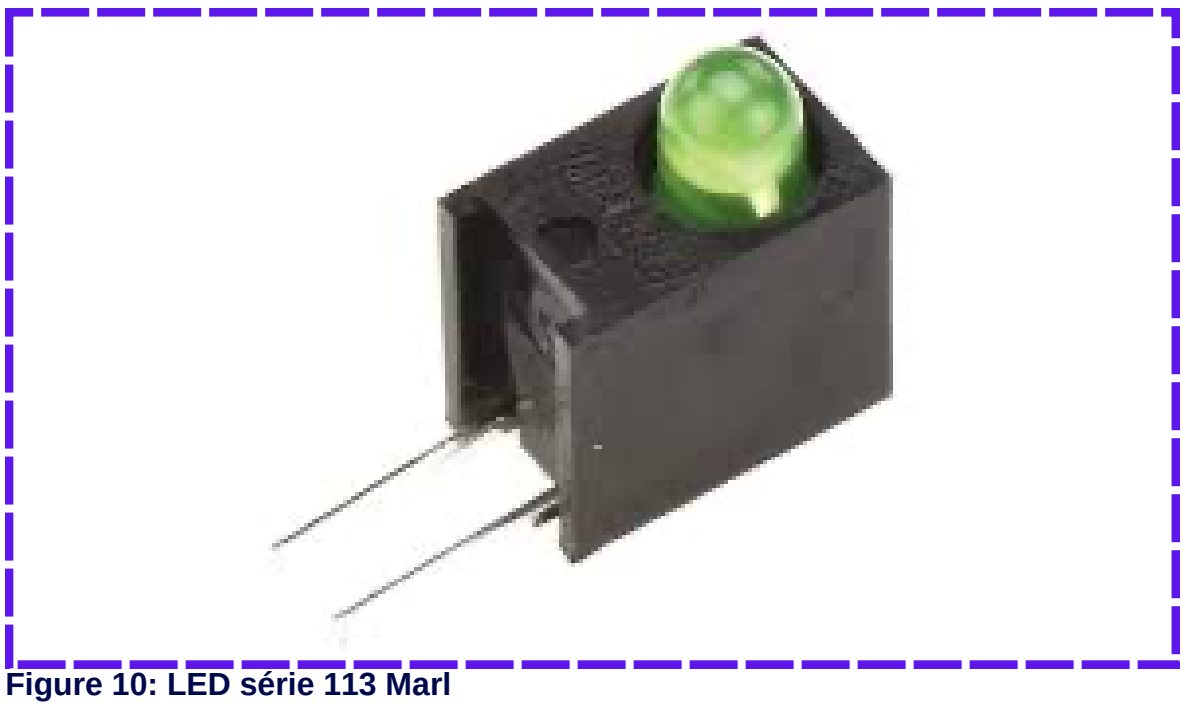


Figure 10: LED série 113 Marl

Nous avons opté pour des LEDs coudées pour permettre une meilleure accessibilité à l'utilisateur qui n'aura pas besoin de retirer le système de la cuve pour visualiser les LEDs et recharger la batterie (connecteur USB coudé également). Des résistances de 150 ohms permettent de limiter le courant dans les LEDs à 20mA.

• $R = Valim - Vled\ Iled$ = 5−2 0.02 = 150Ω.

Parmi les trois LEDs une n'est pas coudée, car son but est de servir de témoin pour les phases de test et n'a donc pas vocation à être utilisée une fois le système intégré dans un boîtier dans la cuve.

b. Boutons-poussoir

Figure 11: Bouton-poussoir

Des boutons-poussoir coudés ont été choisis, là encore pour permettre une meilleure accessibilité. Des condensateurs anti rebonds ont été ajoutés en parallèles et des résistances de pull-down sont de forte valeur pour limiter la consommation. Cependant, parmi ces deux boutons un seul est coudé, le second ayant été ajouté pour les phases de test, il n’avait donc pas besoin d’être coudé.

c. ESP32 Firebeetle

Qu’est-ce qu’un microcontrôleur ?

Un microcontrôleur est un composant électronique intégrant un processeur, de la mémoire et des périphériques d’entrée/sortie. Il est utilisé pour automatiser des tâches spécifiques dans divers appareils électroniques tels que machines industrielles, électroménagers, instruments de mesure, dispositifs de communication, électronique grand public, automobiles, et contribue également à l’internet des objets (IoT). En résumé, il sert de cerveau électronique, permettant le contrôle et l’automatisation de différentes applications.

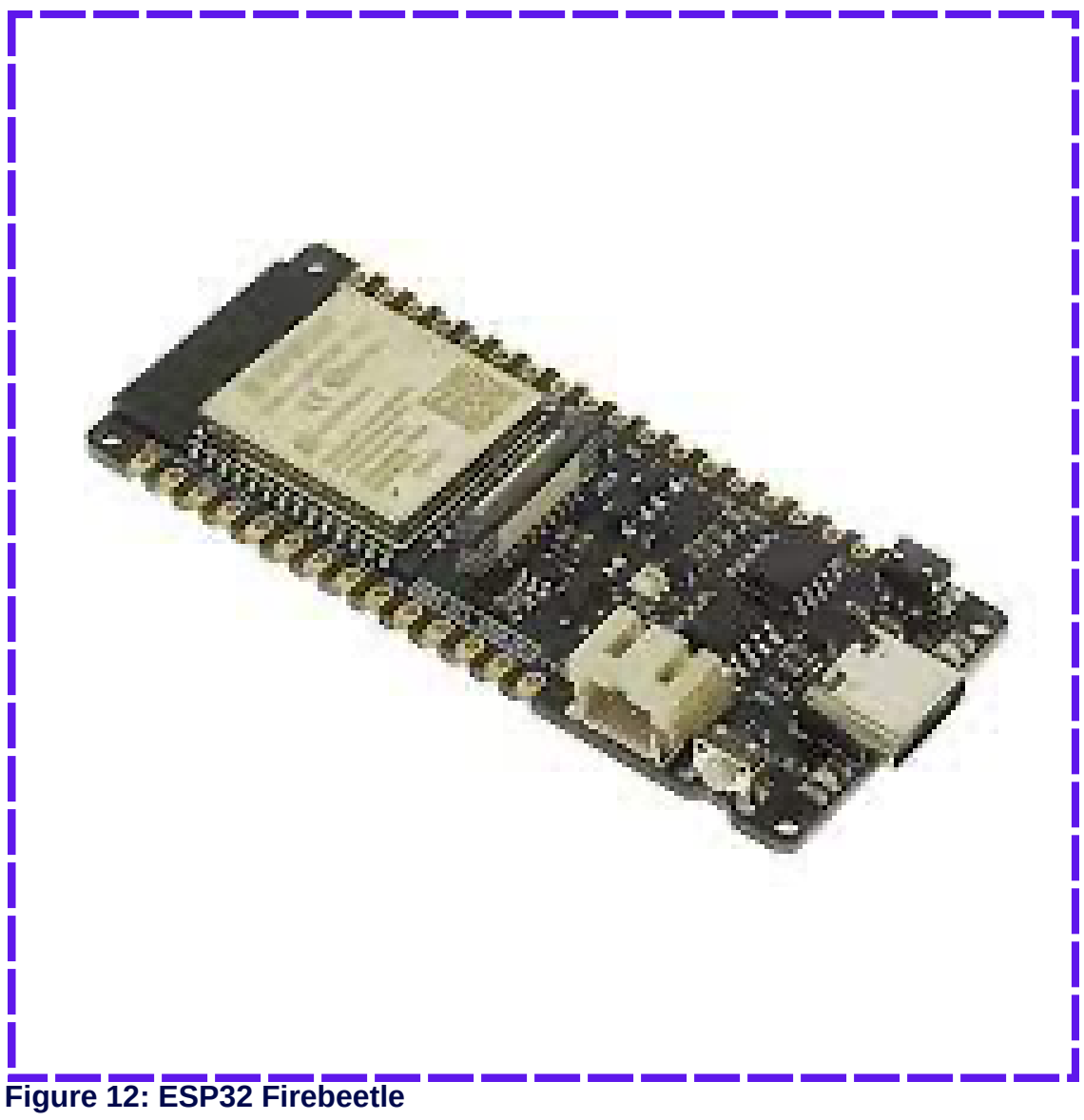


Figure 12: ESP32 Firebeetle

Le Firebeetle DFR0654 a été choisi, car il procède des fonctionnalités telles que le mode deep-sleep, la gestion de la source d’alimentation (batterie ou USB), un module WIFI. De plus, sa consommation est relativement faible (40mA en mode actif et 10µA en mode deep-sleep). Enfin, celui-ci est facilement programmable en C avec l’IDE Arduino ou Visual Studio Code, largement répandus dans le domaine de la programmation.

d. Composants divers

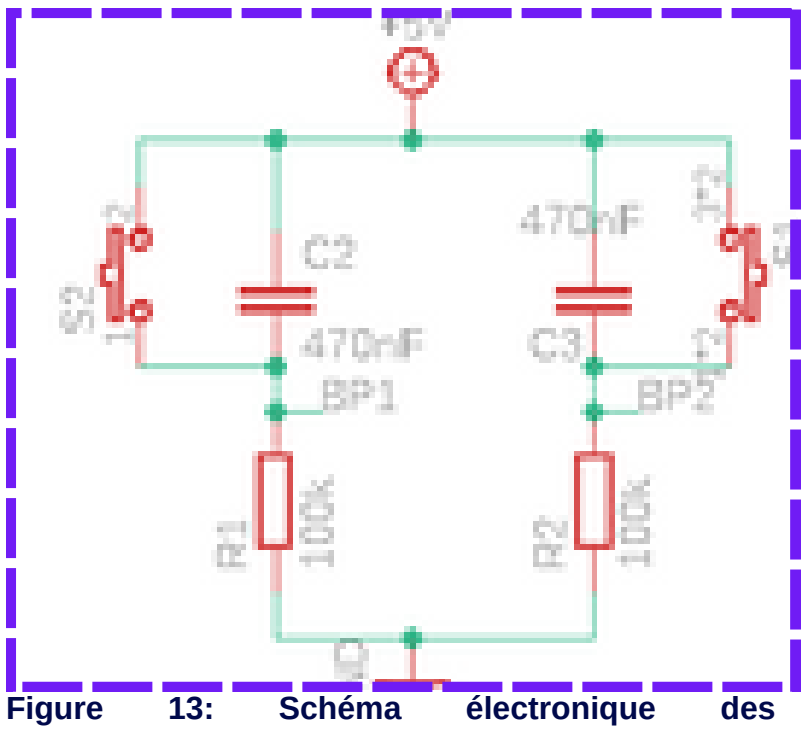


Figure 13: Schéma électronique des condensateurs anti-rebonds

Pour les condensateurs anti-rebonds, nous les avons choisis afin de purifier les signaux générés par l'appui sur les boutons-poussoir, un condensateur anti-rebond de 470nF a été prévu pour chaque bouton.

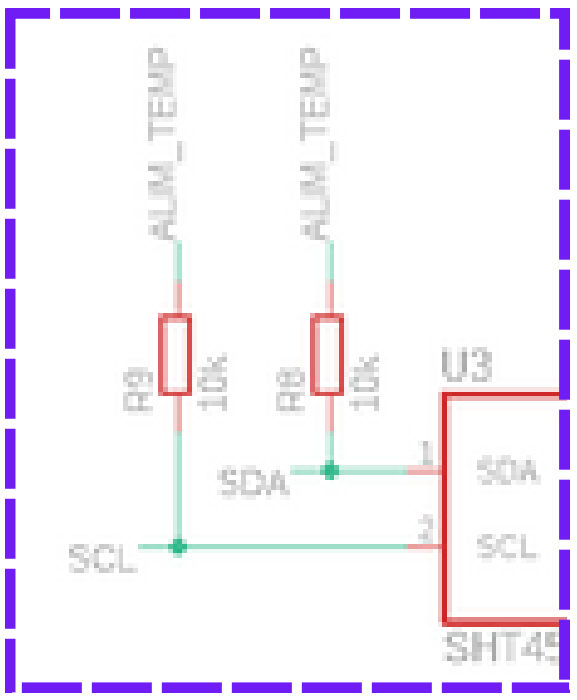


Figure 14: Schéma électronique des résistances de pull-up/pull-down

Pour les résistances de pull-up/pull-down:

Pour la partie I2C comme la datasheet du SHT41 (capteur de température) l'indique, une résistance de pull-up de 10k ohms est nécessaire sur chacune des deux liaisons (SDA et SCL).

Pour les boutons-poussoir une résistance de pull-down de 100k ohms a été ajoutée à chaque bouton-poussoir. Une valeur plutôt élevée permet de limiter la consommation en courant de la carte, un aspect essentiel étant donné que le système est alimenté par une batterie et doit avoir une autonomie d'au moins une année.

3. Plateforme de développement

a. Firebase

Figure 15: Firebase

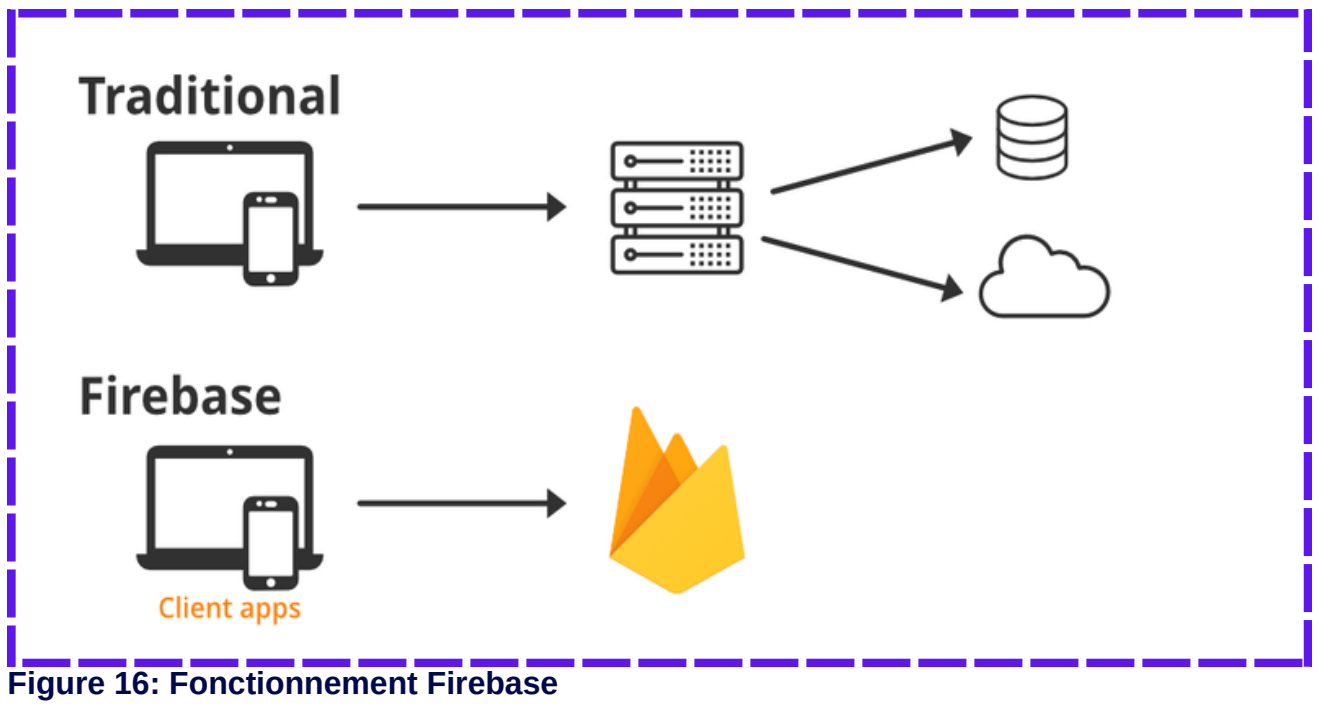


Figure 16: Fonctionnement Firebase

Firebase est une plateforme de développement d'applications mobiles et web proposée par Google. Elle offre une multitude d'outils et de services pour faciliter la création, l'hébergement et la gestion d'applications, incluant l'authentification des utilisateurs, la base de données en temps réel, le stockage de fichiers, les notifications push, l'analyse des données, ainsi que des fonctionnalités pour le déploiement et le suivi des performances des applications. C'est une solution complète qui simplifie le processus de développement et permet aux développeurs de se concentrer sur la création de fonctionnalités plutôt que sur l'infrastructure.

b. Firestore

Figure 17: Firestore

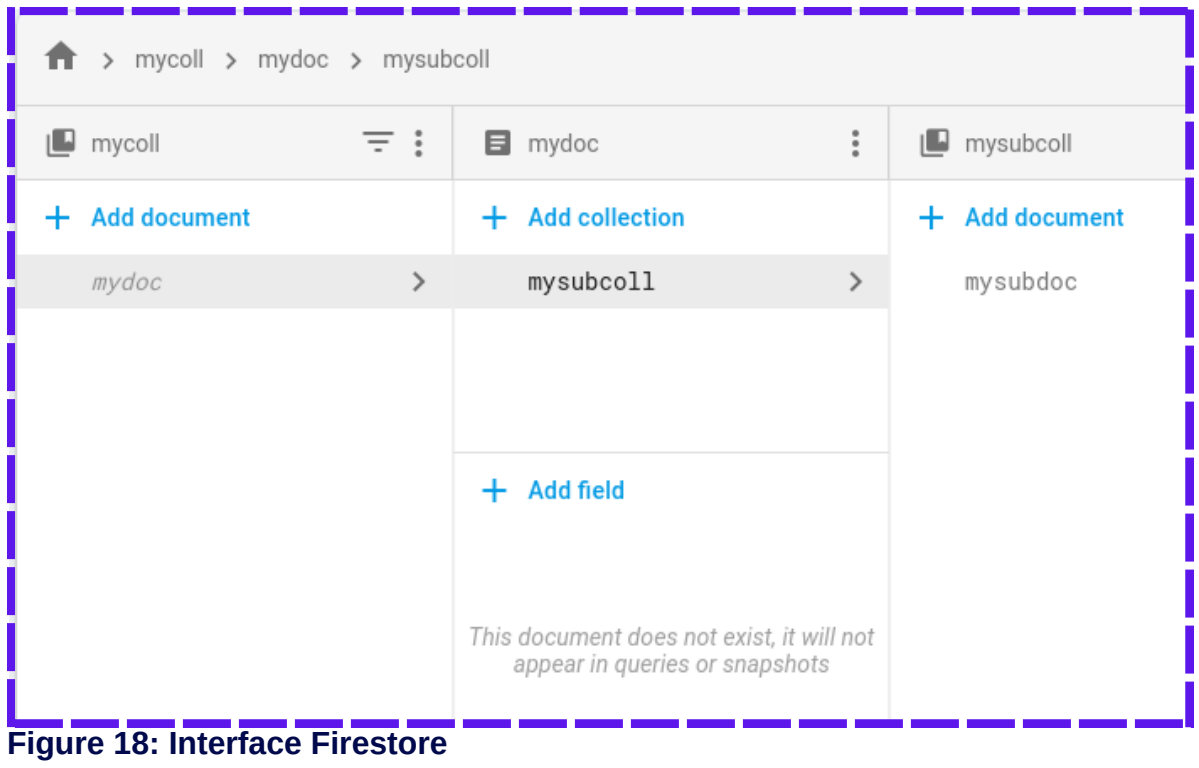


Figure 18: Interface Firestore

Firestore est une base de données NoSQL(Not Only Structured Query Language) en temps réel intégrée à Firebase, la suite de services cloud de Google. Il organise les données en collections et documents, offrant une flexibilité pour gérer des données non structurées. Intégré à Firebase, il bénéficie de fonctionnalités telles que l'authentification des utilisateurs, le stockage de fichiers et les fonctions Cloud. Avec des SDK(Software Development Kit) pour différentes plateformes, des règles de sécurité configurables et une évolutivité automatique, Firestore est une solution prisée pour le développement d'applications nécessitant une gestion dynamique et temps réel des données.

4. Choix des alimentations

a. Alimentation avec batterie

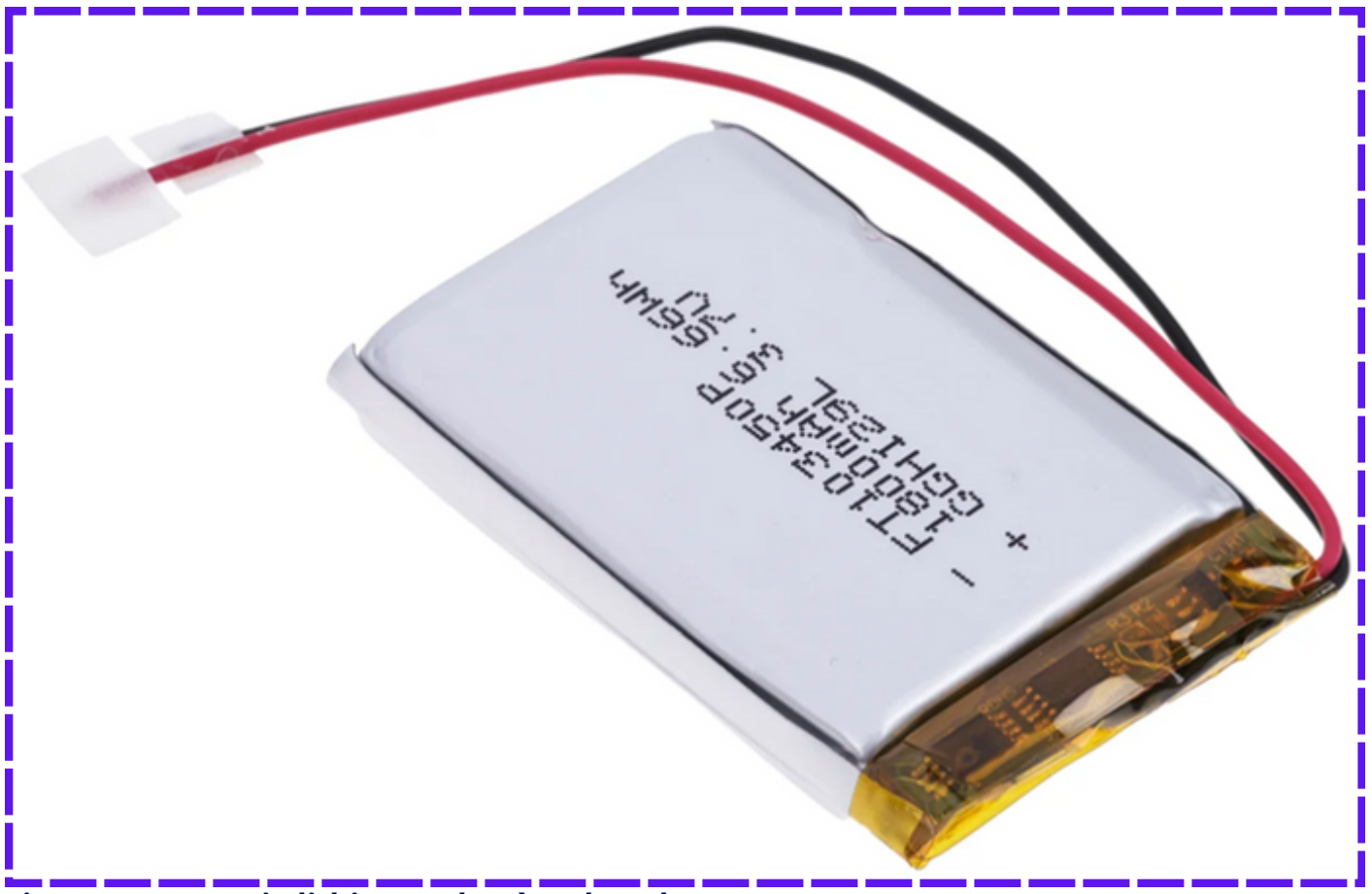


Figure 19: Batterie lithium polymère de 6Ah

Nous avons choisi une batterie lithium polymère de 6Ah. Ainsi, le Firebeetle ayant une consommation de 40mA en mode actif et de 10µA en mode deep-sleep, le capteur ultrason ayant une consommation de 8mA et le capteur de température ayant une consommation de 0,4µA, on peut en déduire le courant consommé par heure (pour une Tactif=20s et Tsleep=30min :

I(pour 30min) = Actif x (0,04 + 0,008 + 4 x 10^^-6) + Tsleep * 10µ =

20sec/3600 x (0,04 + 0,008 + 4 x 10^^-6) + 0,5 x 10µ = 271,669µA

On a donc l’autonomie suivante :

Autonomie = 6Ah/543,3378µA = 11042,85 heures = 1,26 ans = +/- 460jours

b. Alimentation divers

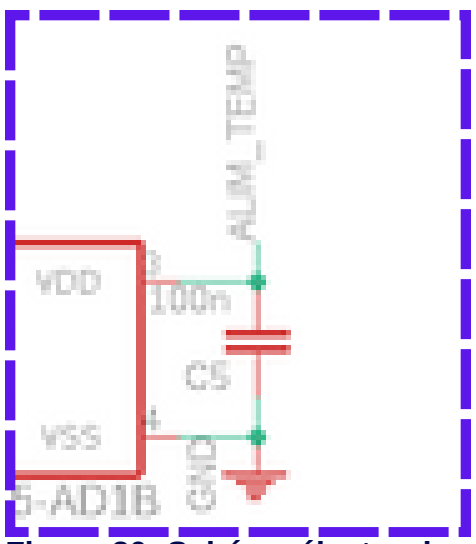


Figure 20: Schéma électronique de l'alimentation du capteur de température

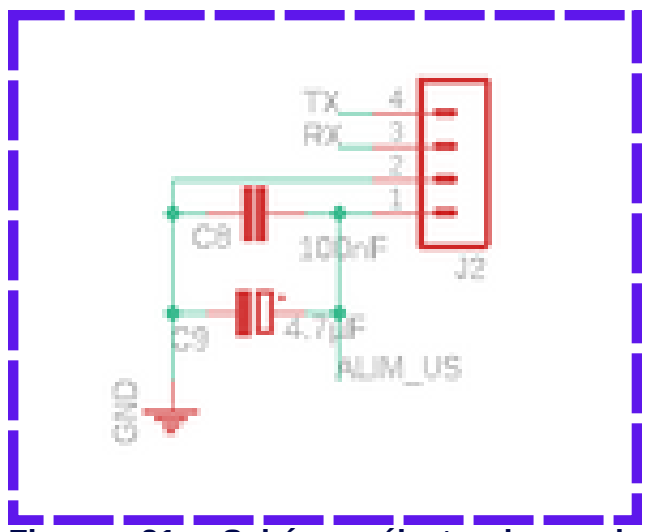


Figure 21: Schéma électronique de l'alimentation du capteur US

Afin de limiter leur consommation en mode deep-sleep, leur borne d'alimentation est reliée à des pins digitaux du Firebeetle.

Pour l'alimentation du capteur ultrason, deux condensateurs de découplages sont prévus : un de 100nF et un second de 4,7µF. Cependant, en réalité, celui de 4,7µF a été prévu par précaution mais n'a pas été soudé. Il peut néanmoins être rajouté à tout moment si besoin en est.

Pour l'alimentation du capteur de température, nous avons ajouté un condensateur de découplage de 100nF du fait que ce soit un circuit intégré.

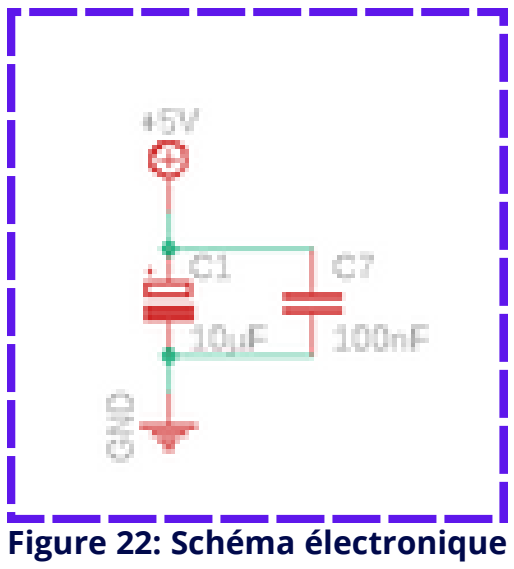


Figure 22: Schéma électronique de l'alimentation de la carte

On a deux condensateurs de découplages en parallèle. Un de 10µF et un second de 100nF. La carte Firebeetle, quant à elle, est déjà composée de condensateurs de découplage pour protéger les circuits intégrés.

III. Réalisation

1. Conception du module de mesure de la cuve

a. Schéma électronique

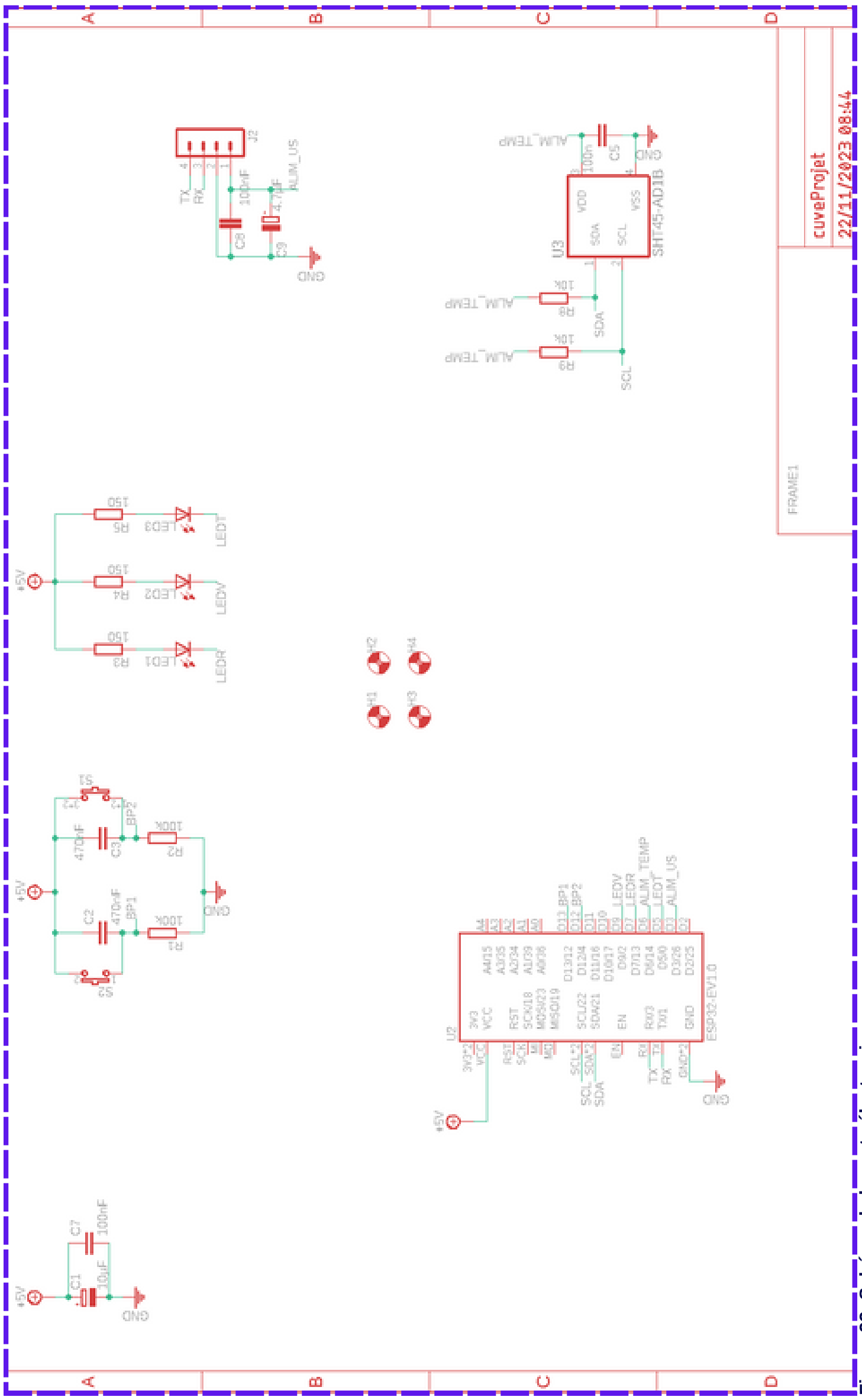


Figure 23: Schéma de la carte électronique

La carte se décompose en plusieurs parties visibles sur le schéma ci-dessous :
Nous avons la partie acquisition :

Cette partie du schéma est constituée de deux capteurs : un capteur de température et un capteur ultrasons. Le capteur ultrasons mesure la hauteur de vide (d'air) dans la cuve afin de calculer le volume d'eau restant comme le montre le schéma ci-dessous.

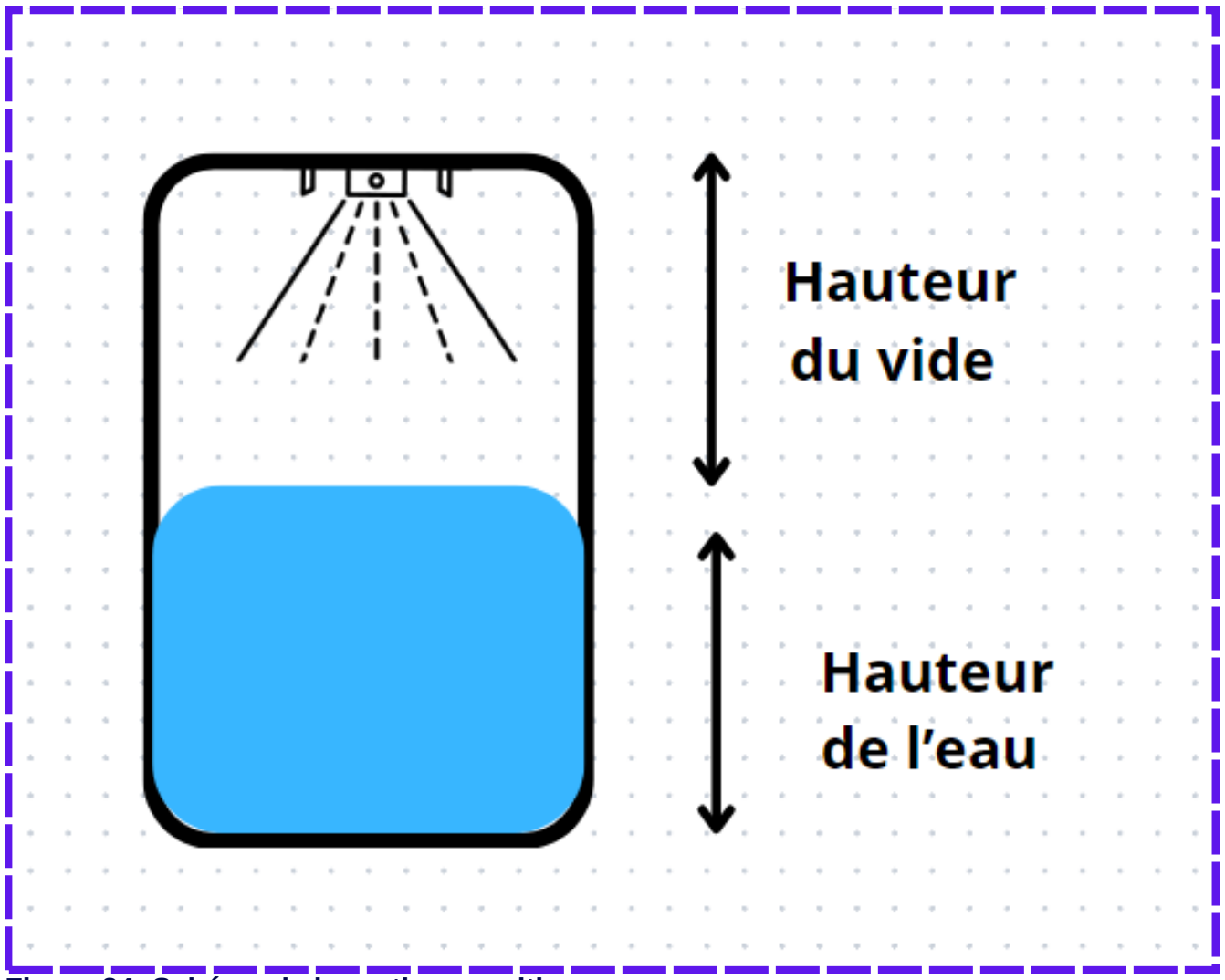


Figure 24: Schéma de la partie acqusition

Un deuxième capteur mesure la température ambiante ainsi que le taux d'humidité. Cela permet ainsi de connaître l'état de l'eau en fonction de la température (liquide, gelée) ainsi que de connaître la température extérieure sans même sortir de chez soi.

Pour la partie Interface Homme/Machine (IHM) :

Trois LEDs et deux boutons ont été intégrés au système, permettant à l'utilisateur d'interagir avec le module. L'utilisateur pourrait ainsi changer de mode en appuyant sur un bouton, ou encore visualiser les transferts de données grâce aux LEDs, ou encore être averti de la pleine charge ou de la charge de la batterie par des LEDs. En étant coudés et placés à proximité du port USB, ces éléments de l'IHM sont facilement accessibles sans avoir à retirer le module de la cuve (pour la recharge de la batterie, par exemple).

Pour la partie logique :

Un Firebeetle équipé d'un ESP32 traite les informations acquises par les capteurs afin de les envoyer sur la base de données Firebase. Il gère également l'IHM, l'alimentation (tension de la batterie) et la charge de la batterie via son port USB. Grâce à son module WIFI, il peut émettre et recevoir des données avec l'interface WEB.

2. Développement logiciel

a. Programmation de l'ESP32

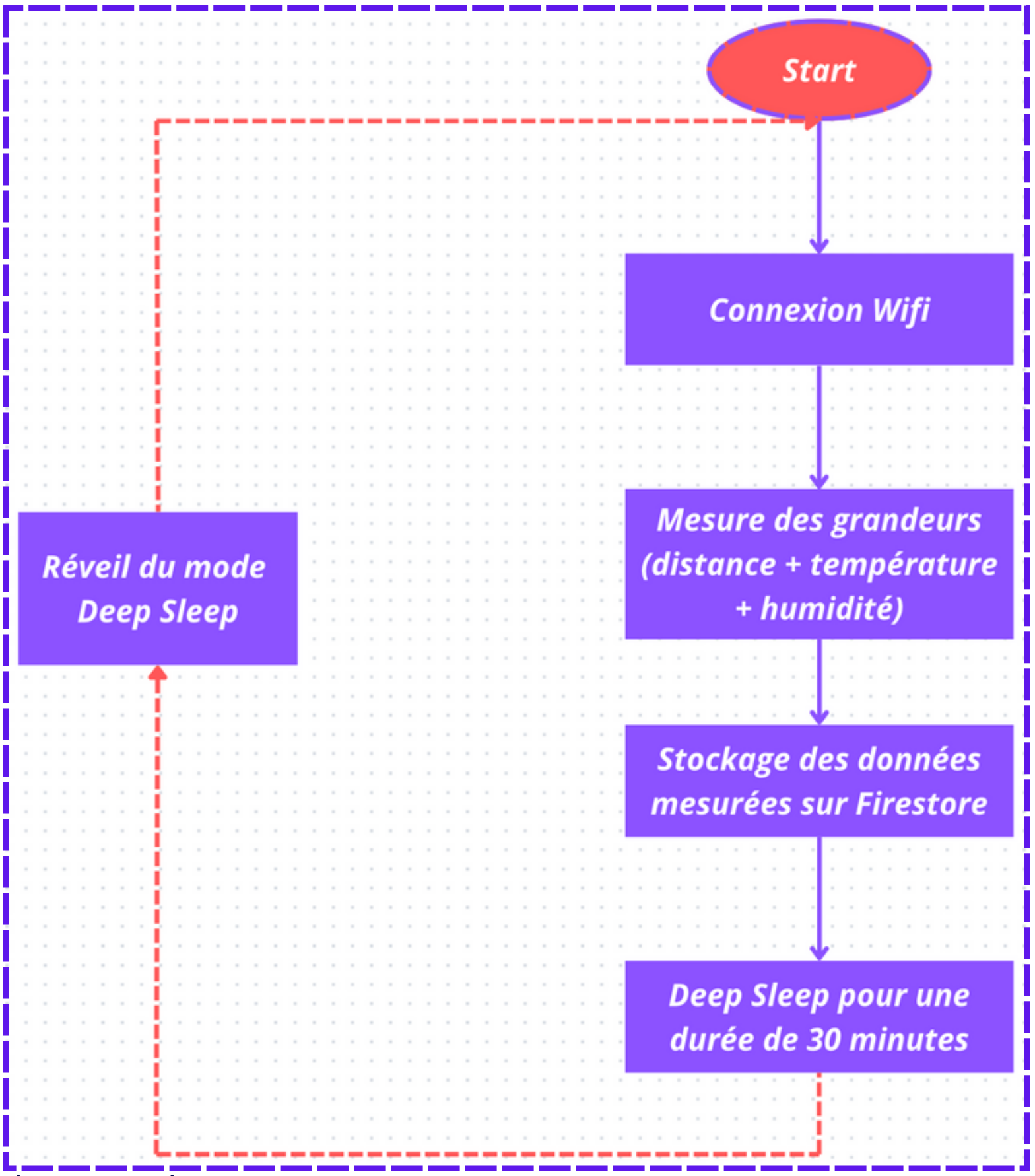


Figure 25: Fonctionnement ESP32

Conformément aux fonctionnalités décrites précédemment, l'ESP est programmé pour effectuer les actions suivantes : mesurer la hauteur du vide dans la cuve afin d'en déduire son volume d'eau, mesurer la température et l'humidité ambiante, transférer les données vers la base de données, le tout en consommant le moins d'énergie possible.

Pour la connexion WIFI :

Dans un premier temps, l'ESP se connecte en WIFI au routeur du domicile, par exemple, dans le but de pouvoir transférer des données vers une base de données par la suite. À cette fin, le nom de l'accès WIFI ainsi que le mot de passe d'accès sont renseignés dans le code. La connexion au wifi se fait par la fonction suivante :

```
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
```

En paramètre de cette fonction, l'identifiant du routeur auquel l'ESP se connecte, et le mot de passe d'accès sont transmis. Ensuite, l'ESP indique à l'utilisateur par l'intermédiaire du moniteur série le statut de la connexion (en cours, connecté ou échec). Cela est illustré par le code suivant :

```
    Serial.print("Connecting 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();
```

Pour la mesure des grandeurs :

Une fois la connexion WIFI effectuée, l'ESP32 va dialoguer avec les capteurs pour réaliser les mesures et récupérer les valeurs des grandeurs mesurées. La liaison avec le capteur ultrason est une liaison série, mais la communication entre le SHT41 et l'ESP se fait par l'intermédiaire d'un bus I2C. Pour piloter chacun des deux capteurs, des bibliothèques propres à ces derniers sont utilisées, permettant de simplifier le code. Dans un premier temps, les objets « Capteur » pour le capteur ultrason et « captemp » pour le SHT41 sont créés.

```
// création de l'objet capteur
sen0311 Capteur (1,3,1);

// creation de l'objet sensor du capt temperature
SensirionI2CSht4x captemp;
```

Les capteurs sont ensuite alimentés en imposant un niveau logique haut sur les broches correspondantes.

```
pinMode(26,OUTPUT); // config des pins pour l'alimentation des capteurs
pinMode(14,OUTPUT);

digitalWrite(14,HIGH); // mise à 1 des pins pour l'alimentation des capteurs
digitalWrite(26,HIGH);
```

Ensuite, les mesures sont effectuées et stockées dans les variables « i » pour la distance, « aTemperature » et « aHumidity ».

```
i = Capteur.getDistance();  // mesure de la distance en mm et stockage d
error = captemp.measureLowestPrecision(aTemperature, aHumidity); // mesu
                                //temperature et stockage dans les  variable
if (error != NO_ERROR)
{
    Serial.print("Error trying to execute measureLowestPrecision(): ");
    errorToString(error, errorMessage, sizeof errorMessage);
    Serial.println(errorMessage);
    return;
}

Serial.println("mesure prise");
```

Dans le même temps, l'utilisateur est informé via le moniteur série du succès ou de l'échec de la mesure. Avant d'entrer en mode Deep-Sleep, l'alimentation des capteurs est coupée pour limiter la consommation.

```
digitalWrite(14,LOW); // mise à 0 des pins pour éteindre des capteurs
digitalWrite(26,LOW);
```

Pour le traitement des mesures transmises par les capteurs :

Lorsque les mesures sont effectuées et transmises à l'ESP par les capteurs, le microcontrôleur va les traiter. En effet, les données transmises ne sont pas exploitables telles quelles. Les bibliothèques évoquées précédemment permettent de récupérer les données des capteurs et de les mettre en forme. Ainsi, grâce à ces bibliothèques, nous obtenons des grandeurs dans les unités souhaitées (°C pour la température, cm pour la distance et % pour l'humidité).

Pour le stockage des données :

Afin de les rendre accessibles à tout moment par l'utilisateur, les données sont stockées par l'ESP dans une base de données Firestore. Cela évite de stocker les données dans le microcontrôleur, qui a une capacité de stockage limitée, et qui nécessiterait un mode actif permanent de l'ESP32, limitant ainsi son autonomie. Pour chaque groupe de trois mesures transmises (température + distance + humidité), l'ESP crée un nouveau document sur Firestore nommé "mesure n" où n est le numéro de la mesure. L'ESP commence d'abord par se connecter au projet Firebase avec la clé API, une adresse email et un mot de passe comme l'illustre le code ci-dessous.

```
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;
```

Une configuration est ensuite nécessaire avec la ligne suivante :

```
Firebase.begin(&config, &auth);
```

Le nouveau chemin où seront stockées les mesures est nommé (documentPath).

```
  String documentPath = "Cuve/mesure";
  documentPath += String(count);
```

Les trois grandeurs sont ajoutées au futur document avec leur intitulé, leur type :

```
// ajout des grandeurs mesurées dans le document
content.set("fields/hauteur/doubleValue", i/10.0);
content.set("fields/temperature/doubleValue", String(aTemperature));
content.set("fields/humidite/doubleValue", String(aHumidity));
```

Enfin, le nouveau document est créé avec le code suivant en prenant tous les paramètres préalables.

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

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

Par la même occasion, l'utilisateur est informé par le biais du moniteur série du succès ou de l'échec de la création du nouveau document dans Firebase.

Pour l'entrée en mode Deep-Sleep :

Dans une démarche d'économie d'énergie pour préserver la batterie et avoir une autonomie maximale, l'ESP entre en mode Deep-Sleep après avoir effectué les trois mesures (température + distance + humidité). La durée de ce mode peut être configurée comme voulu dans notre code (configurée à une minute pour la phase expérimentale). À la fin de cette durée de Deep-Sleep, le microcontrôleur se réveille pour entrer à nouveau en mode actif et démarrer le cycle connexion WIFI/mesures/stockage/deep-sleep.

Avant d'entrer en mode Deep-Sleep, l'ESP coupe l'alimentation des deux capteurs en imposant un niveau bas sur chacune des deux broches correspondantes puis les maintient dans l'état souhaité grâce à la fonction gpio_hold_en().

```
digitalWrite(14,LOW); // mise à 0 des pins pour éteindre des capteurs
digitalWrite(26,LOW);
gpio_hold_en(GPIO_NUM_26); // Permet de conserver l'état de la broche 25 (attentiln la fonction atte
gpio_hold_en(GPIO_NUM_14);
```

Une fois ceci fait, l'ESP configure la durée de veille avec la ligne suivante :

```
// Set deep sleep duration
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
```

Et enfin, il entre en mode Deep-Sleep.

```
// Switch to deep sleep mode
esp_deep_sleep_start();
```

A son réveil, il lui faudra libérer de nouveau les broches bloquées pour le mode Deep-Sleep pour pouvoir les réutiliser naturellement et ainsi pouvoir alimenter les capteurs.

```
gpio_hold_dis(GPIO_NUM_26);
gpio_hold_dis(GPIO_NUM_14);
```

b. Programmation de l’espace Firebase

Firebase est une plateforme de développement d'applications mobiles et web proposée par Google. Il fournit un ensemble d'outils et de services cloud pour simplifier et accélérer le processus de développement d'applications, offrant diverses fonctionnalités clé pour les développeurs. Concernant la programmation de l’espace Firebase, nous avons choisi d’utiliser la base de données Firestore plutôt que Real Time DataBase (base de données en temps réel aussi disponible sur Firebase) simplement, car nous n’avions pas besoin dans notre utilisation de la base de données d’avoir accès à des données en temps réel. En effet, le niveau d’eau dans une cuve est une donnée qui change à une fréquence relativement petite. De plus, la problématique de la consommation d’énergie est vite apparue, une de nos solutions a été de réaliser des mesures à intervalle de temps assez long (30 minutes). Nous pouvons donc éteindre notre système le plus possible et ainsi optimiser la durée de vie de notre système. La base de données en temps réel n’a donc que peu d’intérêt dans notre utilisation. Nous avons donc priorisé l’utilisation de Firestore.
Notre base de données sert à stocker les données envoyées par l’ESP32. L’idée est donc de faire en sorte que les données soient stockées dans Firestore. Mais aussi que ces données soient accessible afin de les afficher sur une page Web par exemple.

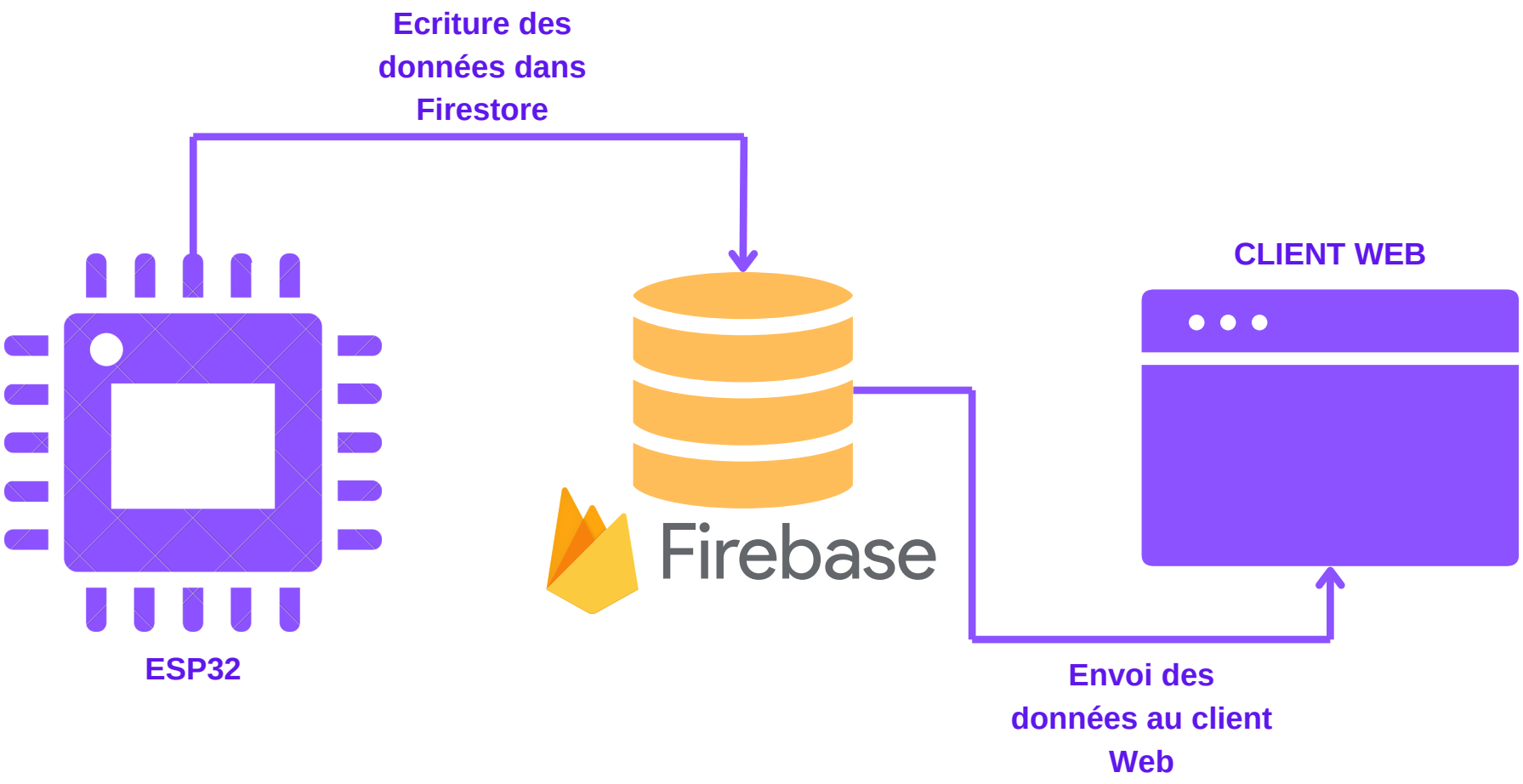


Figure 26: Fonctionnement Firestore

Firestore fonctionne assez simplement. Une base de données Firestore se créer facilement et rapidement. Il suffit de suivre les étapes de Firebase. Lorsque la base de données est créée, on peut ajouter des documents dans la base de données à la main (depuis l’espace Firestore) mais on peut aussi créer un utilisateur à partir duquel on pourra envoyer des données à Firestore. C’est ce que nous avons fait pour l’ESP32. L’ESP32 utilise un identifiant et un mot de passe qui possède les droits d’écrire dans la base de données. C’est de cette manière que nous allons écrire dans Firestore.

c. Développement de la page Web de visualisation des mesures

L'objectif de la page web est de permettre à l'utilisateur d'avoir accès aux données facilement. Nous avons donc réalisé une page Web capable d'afficher les données principales que sont la température, l'humidité, la hauteur de l'eau et le remplissage de la cuve. L'idée était d'afficher chacune des données sur une première page et de rajouter une cuve dans laquelle le niveau de l'eau serait modifié en fonction du niveau de remplissage dans la cuve. Les valeurs des capteurs sont donc affichées de cette façon sur la page web :

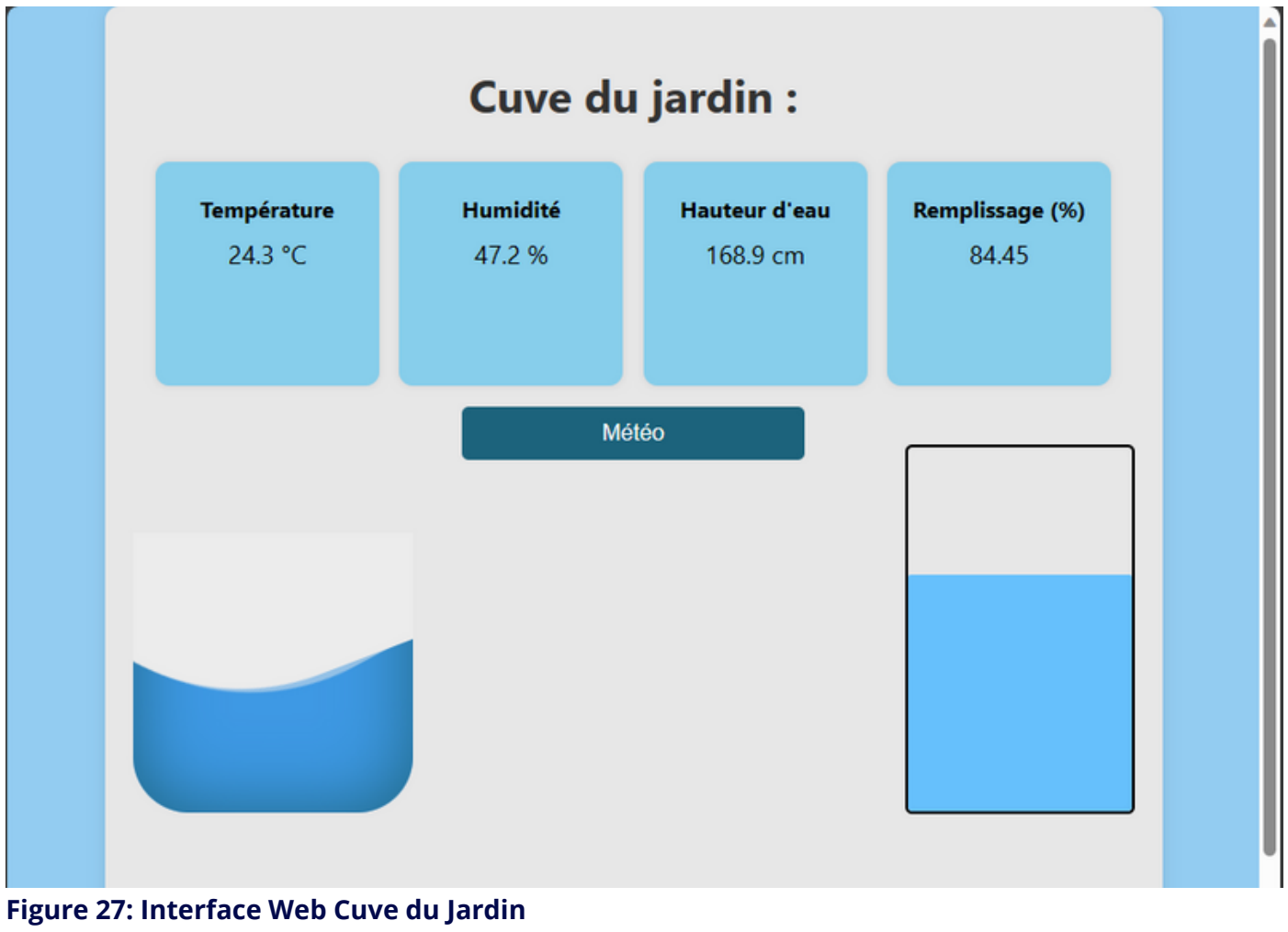


Figure 27: Interface Web Cuve du Jardin

```
<div class="container">
<h1>Cuve du jardin :</h1>
```

```
<button id="weatherButton">Météo</button>
```

Il fait aussi le lien entre tous les fichiers de la page (CSS et JS). Par exemple, ici, on appelle le fichier CSS qui met en forme notre fichier.

```
<link rel="stylesheet" href="dataacquire.css">
```

Le fichier CSS à lui, pour objectif, de définir la présentation de la page web. La ou le HTML ne définit que le contenu de la page web, le CSS, lui, permet contrôle l'apparence de la page web. Le code CSS définissant le style de la page, il peut donc définir beaucoup de choses. Comme par exemple les polices d'écriture, la couleur des polices, et tout ce qui se rapporte au texte. Mais le CSS peut aussi définir les couleurs et les formes de "fond" de la page web. Voici ce que nous avons fait pour ajouter au titre un style :

```css
h1{
  color : #333;
  font-size: 2em;
}
```

Ici, on défini, la couleur du texte compris dans la balise <h1> (ici le titre) et on défini la taille de la police à 2em. Voici maintenant un exemple de notre code CSS qui permet de définir la couleur, la taille et la position des carrés bleus dans lesquels les valeurs de notre capteur sont affichées :

```css
.data-square {
  width: 150px;
  height: 150px;
  background-color: #f9f9f9;
  border-radius: 10px;
  display: inline-block ;
  margin: 5px;
  padding: 5px;
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
```

Pour conclure sur le fonctionnement de la page Web, il faut savoir que tout ce qui touche à l'interactivité de la page est géré par le JavaScript. Le JavaScript permet dans notre cas dans un premier temps de connecter notre page web à Firebase en utilisant les librairies de Firebase. Voici ce que nous avons écrit pour cette partie dans notre code.

```javascript
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'
import { firebaseConfig } from './authentifications.js';


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


  // Initialise Firestore obtient la référence du service
  const db = getFirestore(app);

  // Affichage des valeurs pour le débogag
  console.log("db", db);
  console.log("app", app);

  // Récupération d'un document spécifique depuis Firestore
  const docRef = doc(db, "Cuve", "mesure1");
  const docSnap = await getDoc(docRef);
```

Avec le code JavaScript, on arrive à récupérer des données depuis Firestore et à leur donner un nom afin de les réutiliser sur la page web. Nous réalisons aussi avec notre code JavaScript les animations pour l'appui sur le bouton qui nous relance sur une autre page Web qui permettra ensuite d'acquérir la météo. Le code JavaScript fera aussi tous les calculs nécessaires de nos données acquises depuis Firestore afin de les mettre dans le format voulu pour la page web.

Dans un souci de sécurité, pour ce qui est de la connexion à Firestore, nous avons préféré mettre les informations confidentielles (clé API, ID du projet, etc) de notre page dans un autre fichier JavaScript.

Nous avons, par la suite, souhaité créer un bouton “météo” qui dirige l’utilisateur sur une nouvelle page Web. Le but de celle-ci est d’acquérir les données météo de la position de la cuve en rentrant une ville. Cette page affichera les données météorologiques telles que la température, l’humidité et les conditions (pluie, neige, soleil, ...). En fonction des conditions, la page web affichera à l’utilisateur un message. Par exemple, s’il pleut, il y aura écrit : “ Attention, il pleut, la cuve va se remplir”. Pour cette nouvelle page web, il y a évidemment un autre fichier HTML, un autre CSS et un autre JavaScript.

Figure 28: Interface Web météo

Voici le fonctionnement global de notre page Web :

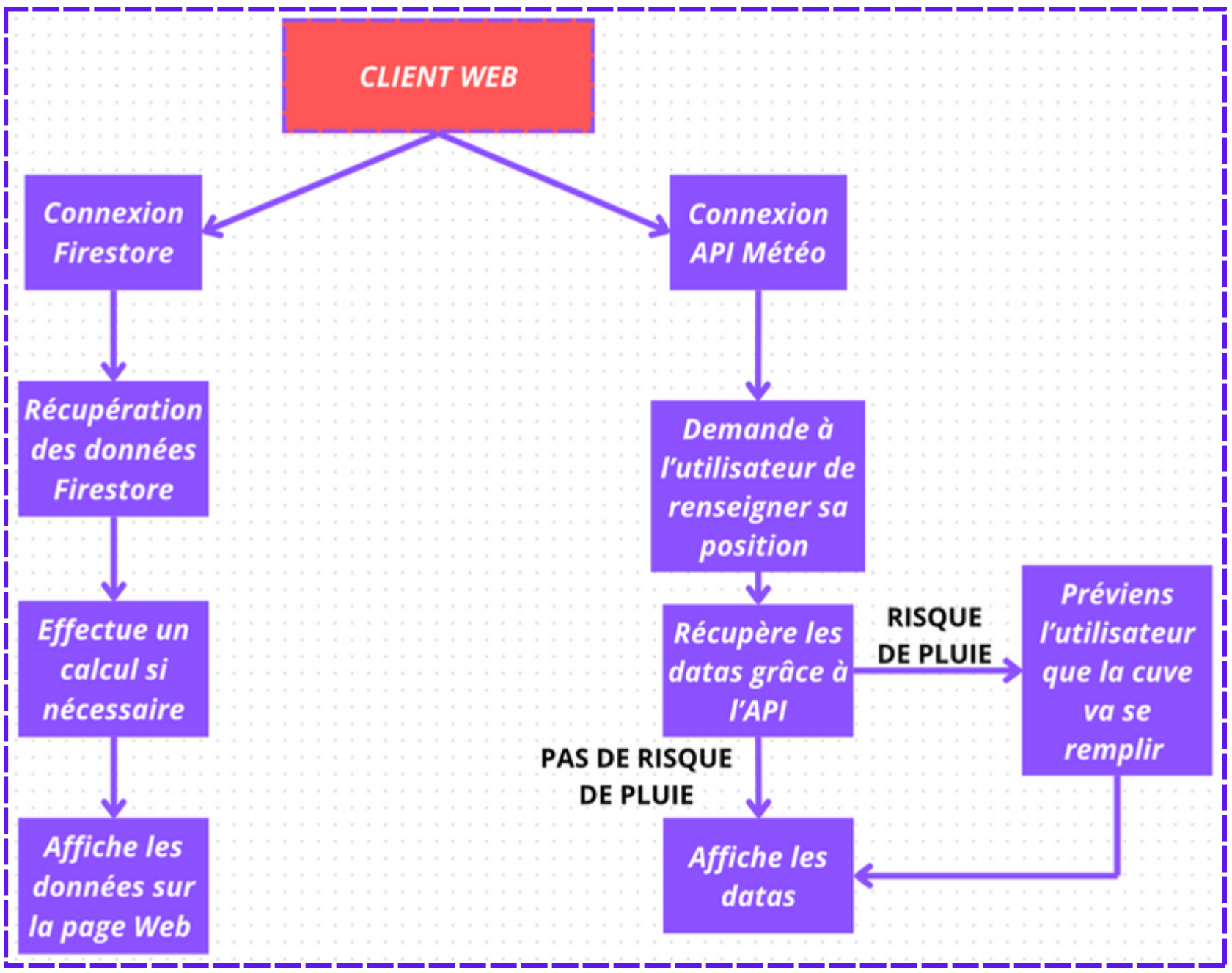


Figure 29: Fonctionnement de la page Web

3. Résultats

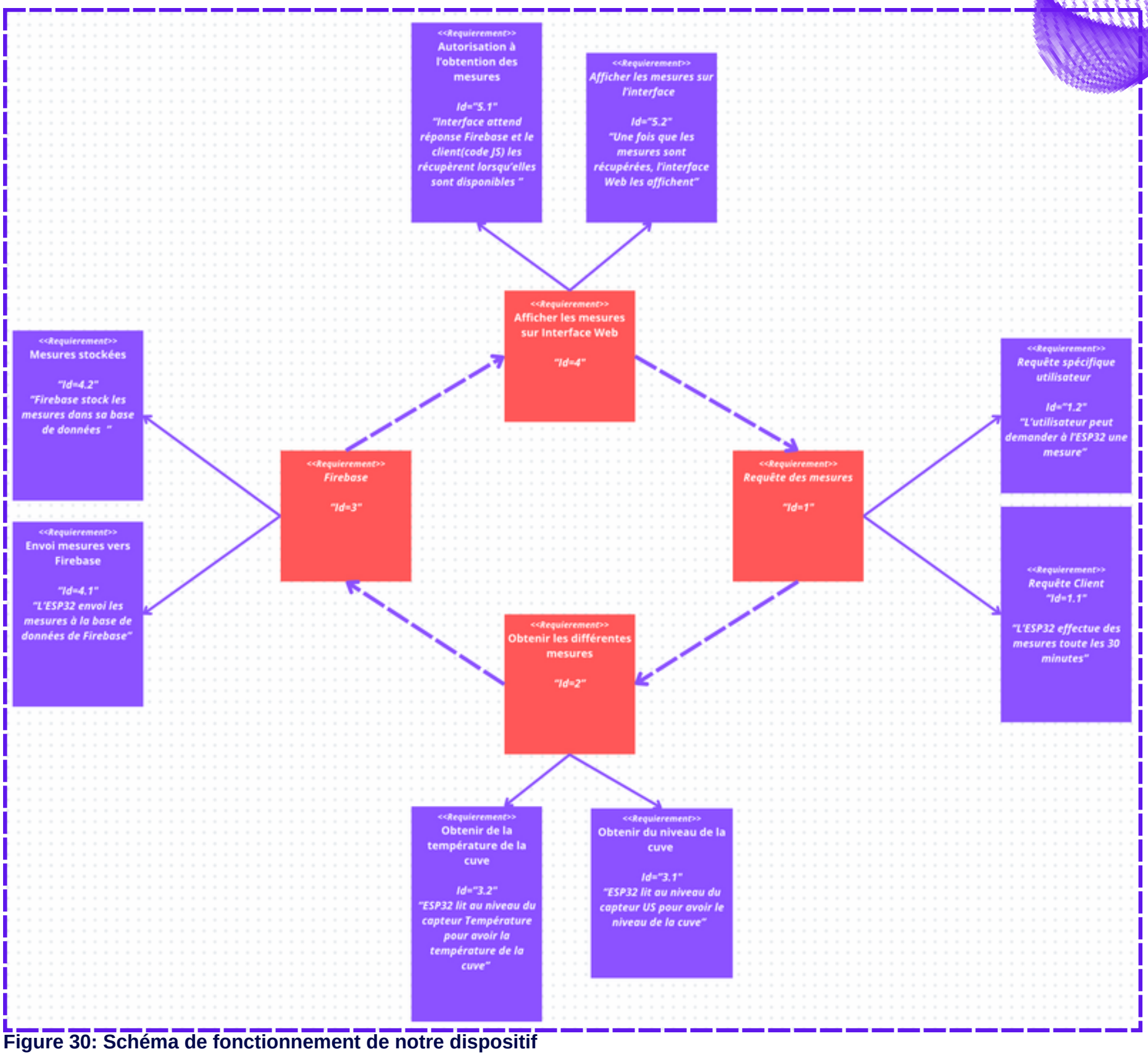


Figure 30: Schéma de fonctionnement de notre dispositif

Voici le schéma de fonctionnement de notre dispositif. Tout d'abord, il y a la requête des mesures permettant de réaliser des mesures toutes les 30 minutes. Ensuite, il y a l'obtention des différentes mesures, telles que le niveau de l'eau dans la cuve avec le capteur US et la température ambiante dans la cuve avec le capteur de température.

Pour la partie Firebase, nous avons d'abord l'envoi des mesures vers Firebase, l'ESP32 envoyant les mesures à la base de données de Firebase. Ces mesures sont ensuite stockées dans la base de données. Enfin, nous affichons les mesures sur l'interface web. Pour cela, il y a d'abord, l'autorisation d'obtention des mesures, ensuite, nous pouvons afficher les mesures sur la page.

4. Tests et validations

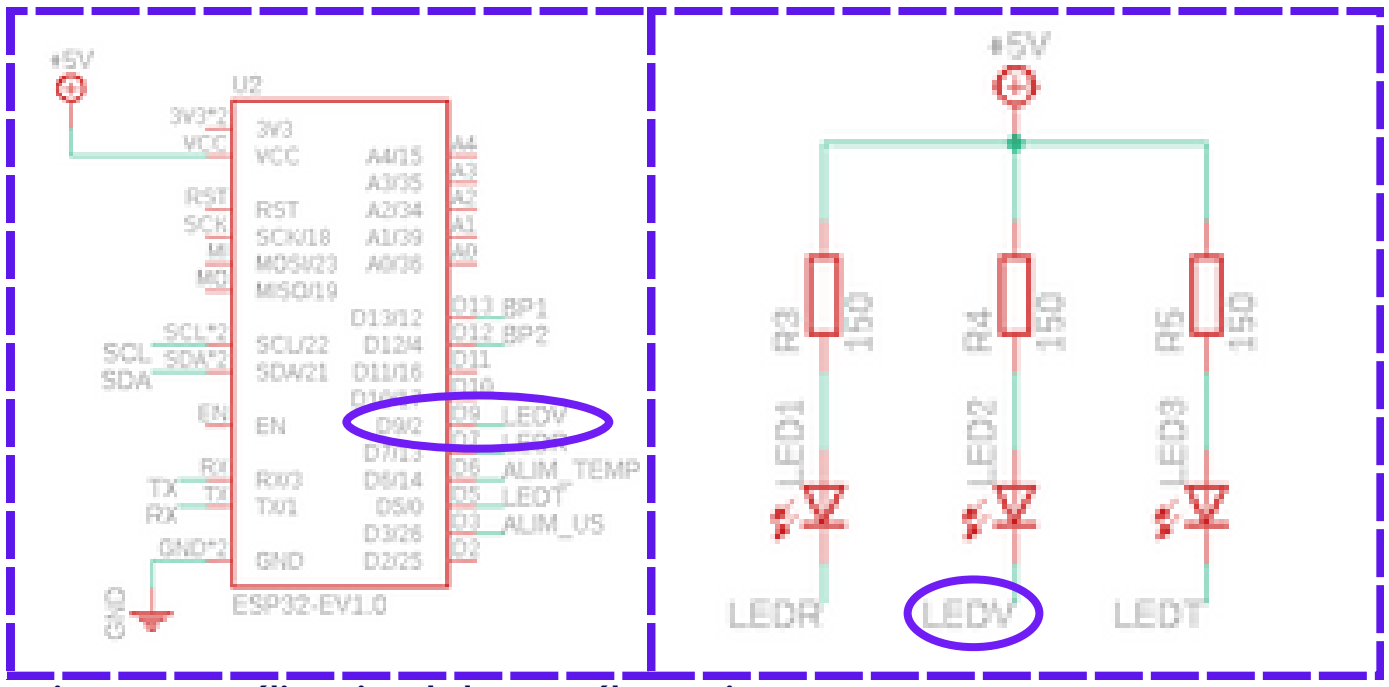


Figure 31: Amélioration de la carte électronique

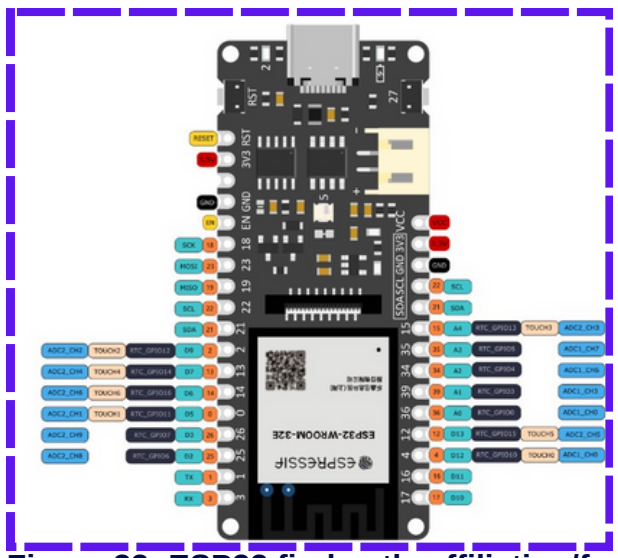


Figure 32: ESP32 firebeetle affiliation/fonctionnalités des pins

Lors des tests de la carte, nous nous sommes rendu compte que nous avions commis une erreur, lorsque nous voulions compiler le code que nous avions développé. Nous recevions un message d'erreur nous informant que la connexion Wi-Fi ne parvenait pas à s'établir. Nous avons donc effectué tous les tests physiques sur la carte, tels que les tests de continuité par exemple. Une fois que nous avons effectué ces tests, nous nous sommes posé la question de l'origine possible du problème sur la carte, sachant que le code était correct. Nous avons donc enquêté sur les broches de l'ESP32.

GPIO 2	2/D9	Used as input or output	ADC2_CH2		For controlling onboard LED by outputting digital signal

Figure 33: Rôle de la pin D9 sur la datasheet de l'ESP32

On constate sur la datasheet de l'ESP32 que la broche D9 est utilisée par l'ESP32, ce qui crée un conflit sur notre carte, car, comme indiqué sur le schéma électrique, nous avions attribué la broche D9 à la LED V. Une fois que nous nous sommes rendu compte de ce souci, nous avons déconnecté la broche D9, ce qui a permis de supprimer le défaut et rendre notre carte fonctionnelle.

Conclusion

Notre projet est aujourd'hui fonctionnel. Il est capable de fonctionner sur batterie, de mesurer des données avec les différents capteurs et de les envoyer sur un serveur distant. L'idée du projet était de réaliser un capteur capable de mesurer des données dans la cuve tout en consommant le moins possible d'énergie afin d'avoir une durée de vie maximale du système sans recharge. Aujourd'hui, le système que nous avons mis en place permet cela même s'il présente toujours quelques failles (boîtier 3D manquant par exemple).

Ce projet a été très intéressant d'une part pour son aspect technique qui, nous a permis d'en apprendre beaucoup sur les ESP32, les serveurs et bases de données ainsi que les pages web. Mais aussi d'une autre part pour son aspect écologique qui est très important aujourd'hui. En effet, ce projet a directement été mis en contexte dans une démarche écologique et écoresponsable qui sont, aujourd'hui, des valeurs très importantes. Le système a pour but premier de fonctionner dans une cuve de récupération d'eau de pluie tout en prenant en compte des contraintes énergétiques dans son fonctionnement.

Pour finir, nous avons su, dans le peu de temps imparti, créer une équipe soudée et capable de se faire confiance. En étant 3 dans un groupe, il n'est pas toujours évident de travailler sur le même projet. Ce que nous avons mis en place est assez simple mais très efficace. L'idée était de se répartir le travail. Chacun travaille sur un sujet plus ou moins différent des autres en se fixant des moments de mise en commun pour être au courant de l'avancement d'un camarade et de l'aider si besoin. Ce mode de travail a très bien fonctionné dans le cadre de notre projet.

C'est donc pourquoi, pour le semestre 6, nous souhaitons continuer ce projet. Nous avons envisagé des pistes d'améliorations que vous retrouverez à la page suivante.

Perspectives

Figure 34: Perspectives

Pour les perspectives du semestre 6, nous voulons réaliser un boîtier 3D afin de pouvoir protéger notre module physiquement, de l'eau de l'humidité.

L'ajout d'un TimeStamp serait pour nous est idéal, car nous pourrions connaître les dates de chaque mesure. Cela permettrait de créer un tableau reprenant les mesures d'une année afin de les analyser.

L'ajout d'un mode adaptatif en fonction des saisons serait très utile. Pour, par exemple, pendant les périodes d'été, avertir de périodes sèches et pour les périodes pluvieuses prévenir l'utilisateur que la cuve risque de se remplir.

La programmation des LEDs pourrait nous permettre d'informer l'utilisateur de l'état de la batterie, on pourrait ajouter une fonction utile pour l'utilisateur avec l'appui sur le bouton-poussoir

Il nous est impératif pour le semestre 6 de développer la partie sécurité pour l'accès à l'interface Firestore afin d'éviter le maximum de risque et d'avoir une base de données sécurisée.

Nous aimerions développer la partie requête pour que l'utilisateur puisse faire une demande de mesures lorsqu'il en a le besoin en plus des mesures réalisées toutes les 30 minutes.

Enfin, la page Web Client peut être améliorée en rajoutant des graphiques par exemple.

Dans un monde où la surveillance précise des ressources est essentielle, ce dossier de projet présente une solution basée sur la technologie de l'ESP32 et des capteurs ultrasons, de température et d'humidité.

L'objectif ?
Acquérir et surveiller avec précision le niveau d'eau, élément vital, dans une cuve, offrant ainsi une gestion intelligente des ressources.

Ce système autonome, alimenté par batterie, déploie ses capteurs pour collecter les données cruciales. Ces informations sont soigneusement acheminées vers Firestore, une base de données cloud, garantissant une accessibilité et une sécurité optimales.

Mais ce n'est pas tout. Une page web dédiée a été méticuleusement conçue pour permettre aux utilisateurs d'accéder facilement et instantanément aux données enregistrées. Cette interface intuitive offre une visibilité claire et précise des informations importantes sur les niveaux d'eau dans la cuve.

Ce projet représente une fusion harmonieuse entre l'utilisation de capteurs, la connectivité cloud et une interface utilisateur.
Une solution intéressante pour ceux qui cherchent à surveiller et à gérer efficacement les ressources essentielles de manière intelligente et proactive.

In a world where precise resource monitoring is crucial, this project report introduces a solution based on ESP32 technology and ultrasonic, temperature, and humidity sensors.

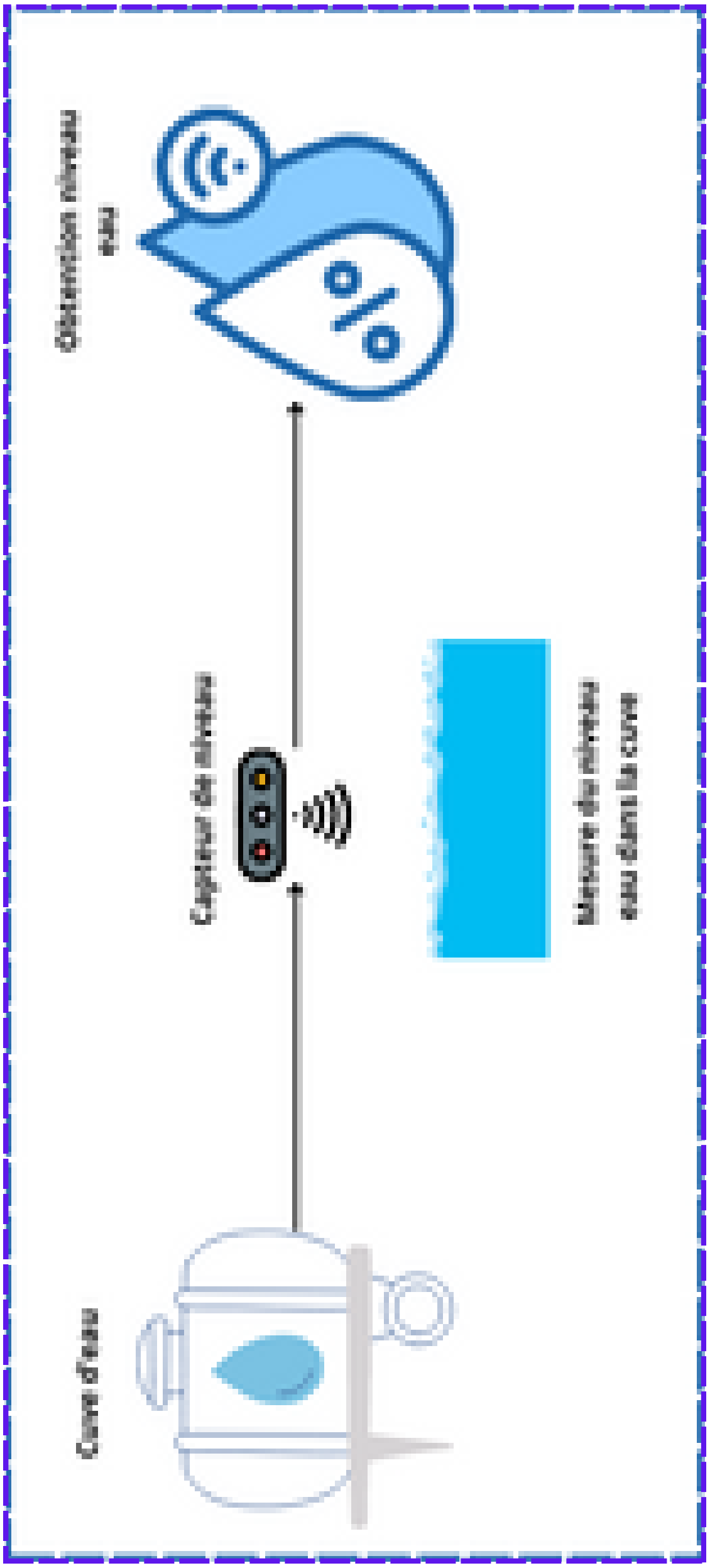
The aim?
To accurately acquire and monitor the rain water level in a tank, which is a vital ressource, offering intelligent resource management.

This self-sufficient system, powered by a battery, deploys its sensors to gather critical data. These insights are meticulously routed to Firestore, a cloud-based database, ensuring optimal accessibility and security.

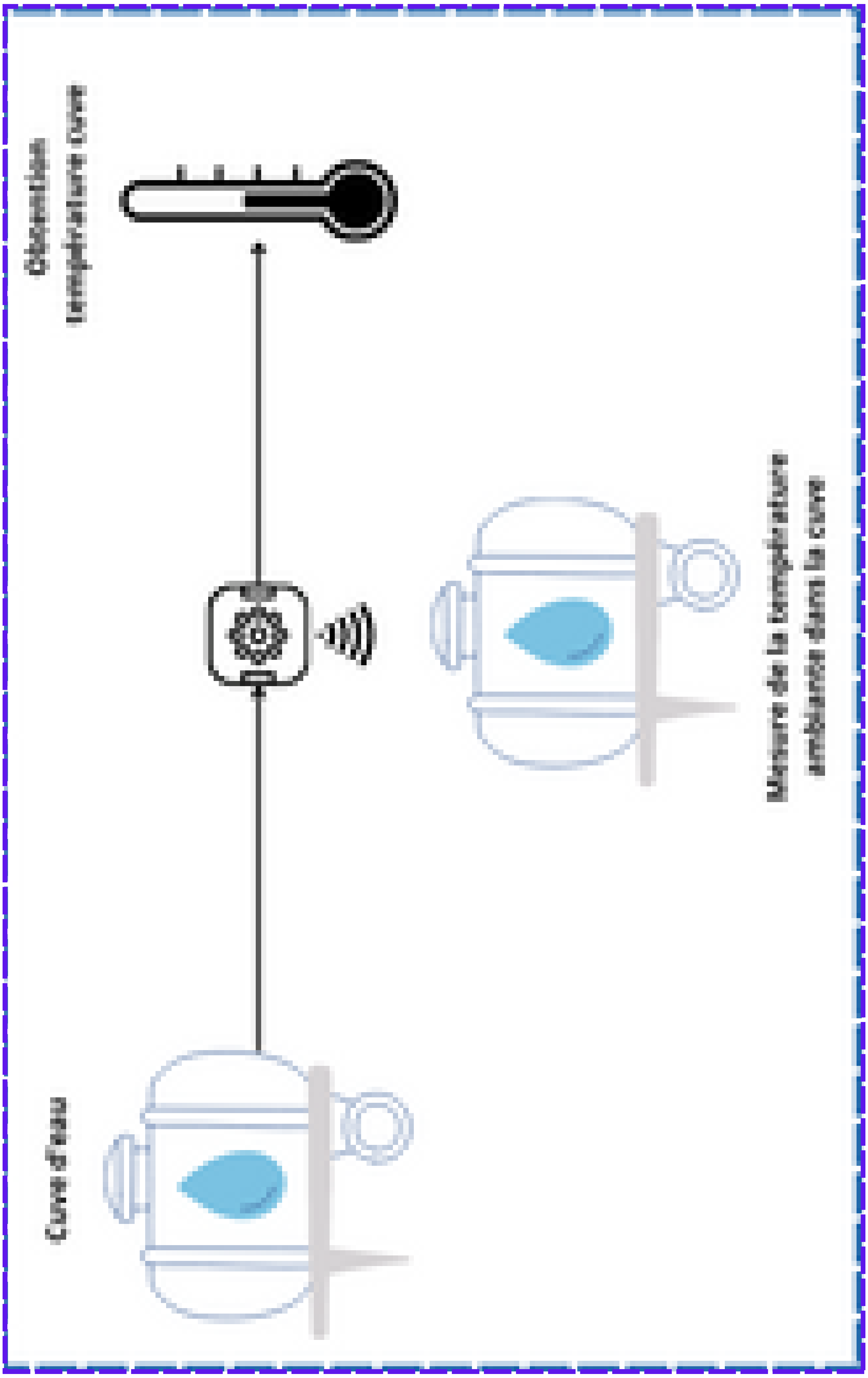
But that's not all. A dedicated web page has been meticulously designed to allow users easy and instant access to the recorded data. This intuitive interface provides clear and precise visibility of important information regarding water levels in the tank.

This project represents a seamless integration of sensor utilization, cloud connectivity, and user interface. A compelling solution for those seeking to effectively and proactively monitor and manage essential resources.

Annexes

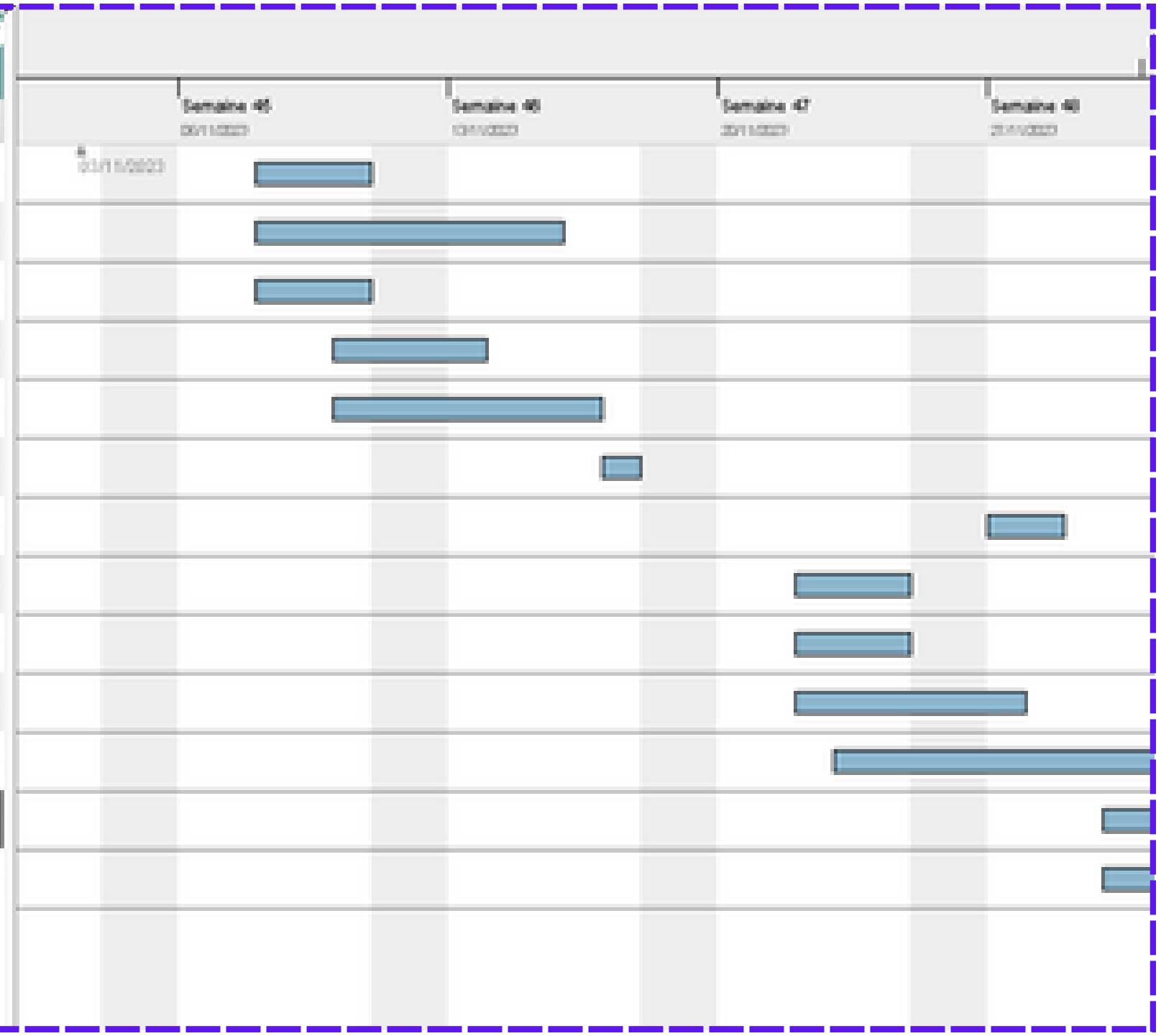

Détection niveau eau
Capteur de niveau
Mesure du niveau eau dans la cuve
Cuve d'eau

Annexes

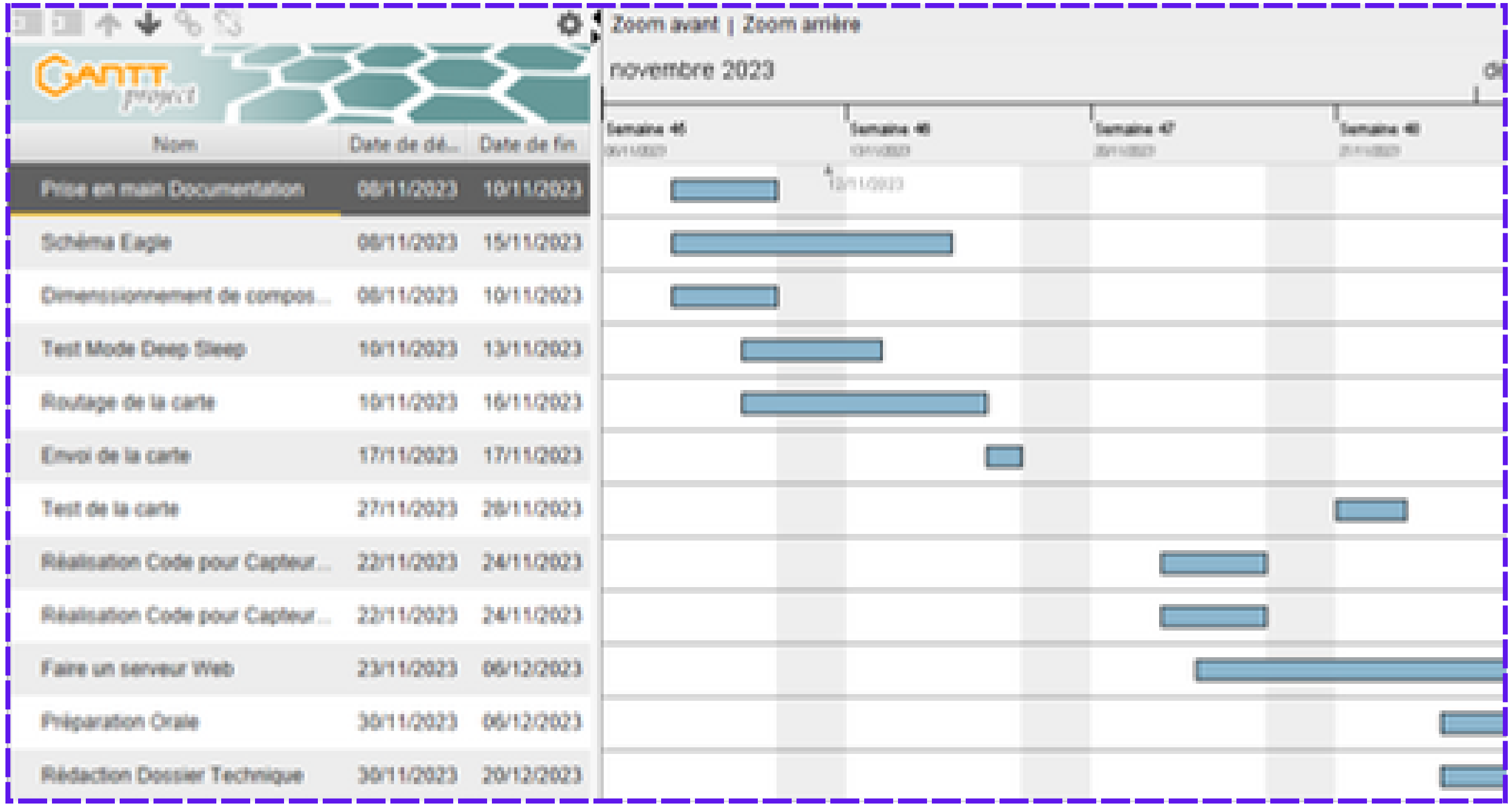

Cuve d'eau
Mesure de la température ambiante dans la cuve
Obtention température Cuve

Annexes

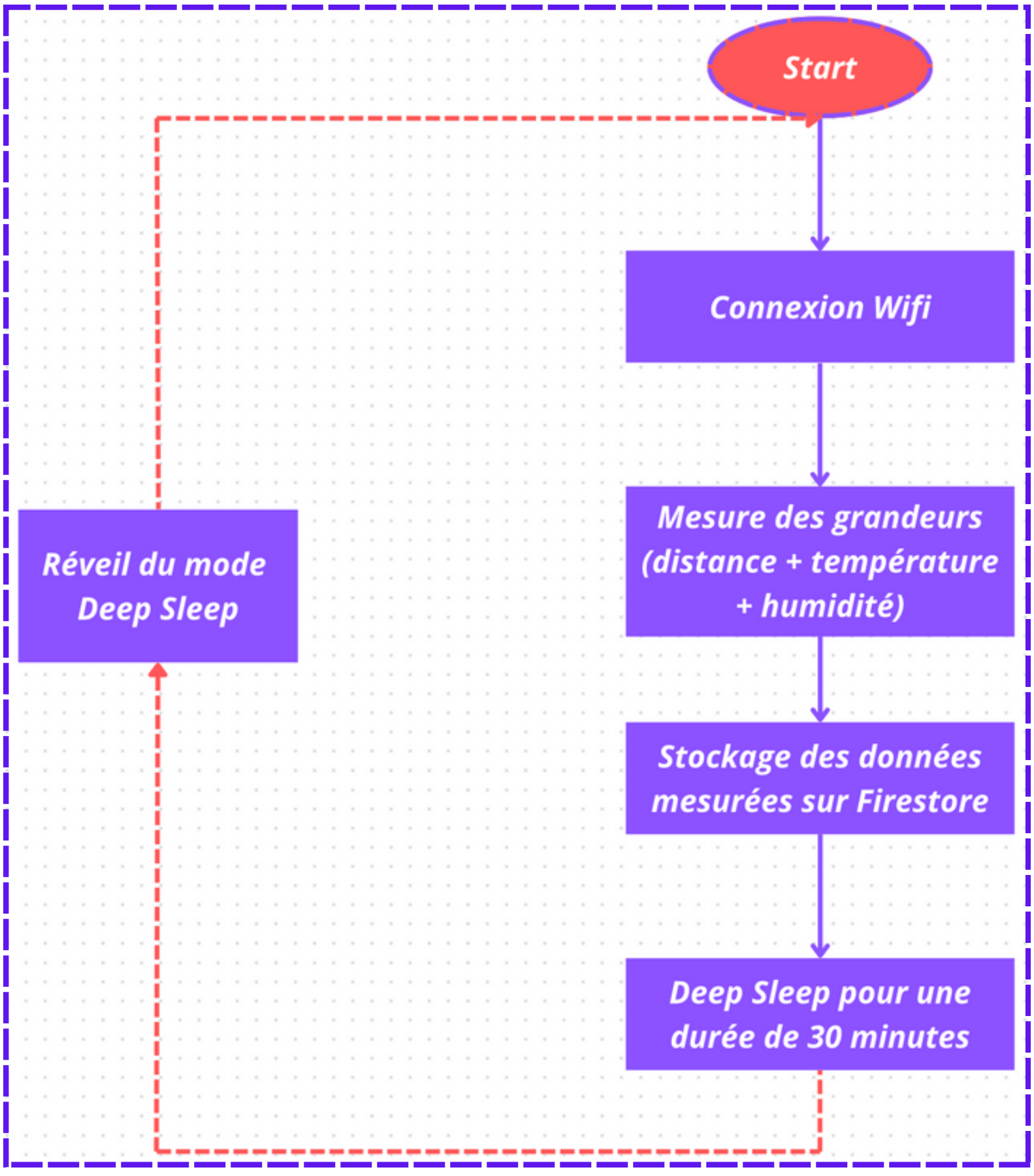
Nom	Date de dé...	Date de fin
Prise en main Documentation	08/11/2023	10/11/2023
Schéma Eagle	08/11/2023	15/11/2023
Dimenssionnement de compos...	08/11/2023	10/11/2023
Test Mode Deep Steep	10/11/2023	13/11/2023
Routage de la carte	10/11/2023	16/11/2023
Envoi de la carte	17/11/2023	17/11/2023
Test de la carte	27/11/2023	28/11/2023
Réalisation Code pour Capteur...	22/11/2023	24/11/2023
Réalisation Code pour Capteur...	22/11/2023	24/11/2023
Réalisation Boitier 3D	22/11/2023	27/11/2023
Faire un serveur Web	23/11/2023	06/12/2023
Préparation Orale	30/11/2023	08/12/2023
Rédaction Dossier Technique	30/11/2023	20/12/2023



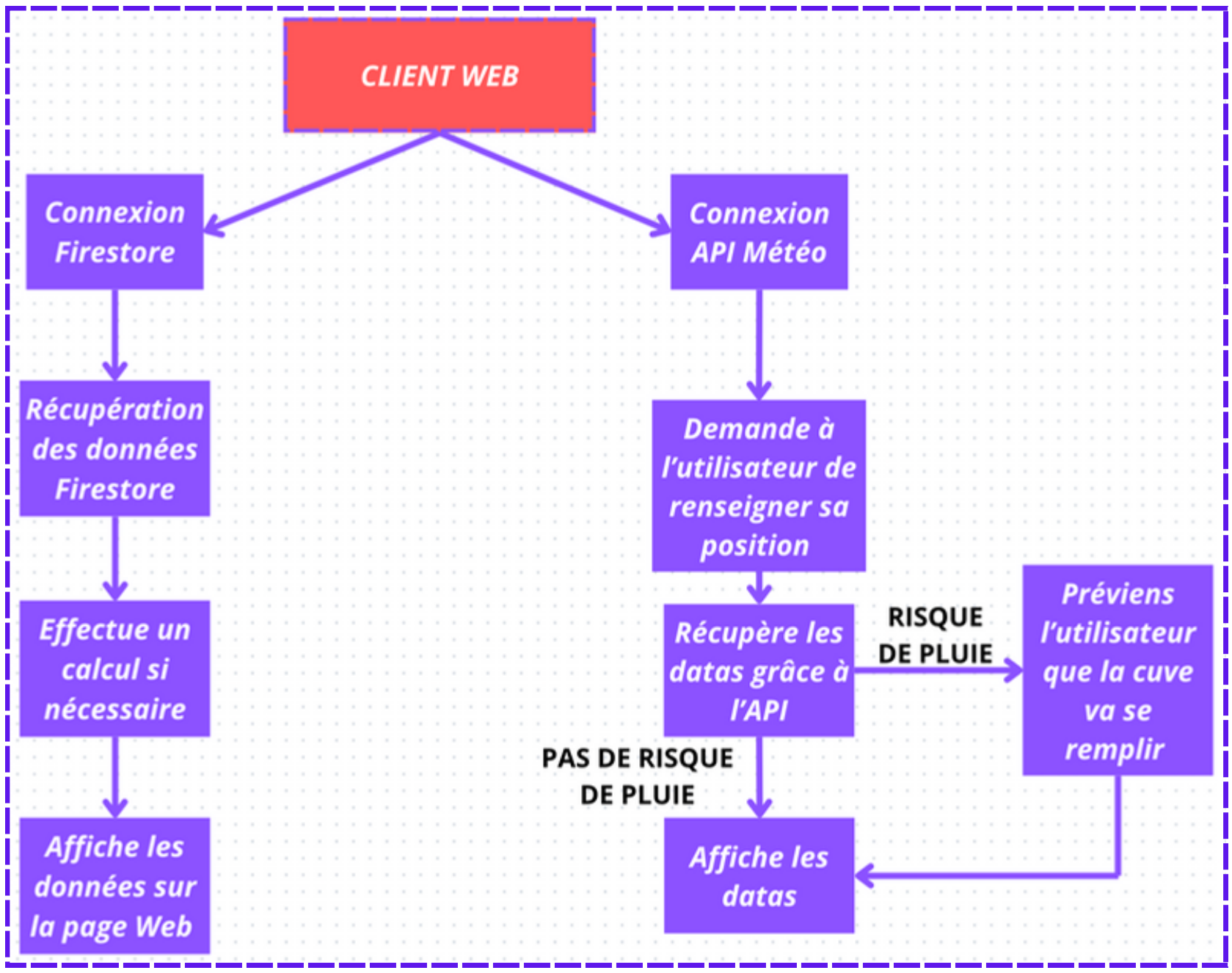
Annexes


GANTT project
Nom
Date de dé...
Date de fin
Prise en main Documentation 08/11/2023 10/11/2023
Schéma Eagle 08/11/2023 15/11/2023
Dimensionnement de compos... 08/11/2023 10/11/2023
Test Mode Deep Sleep 10/11/2023 13/11/2023
Routage de la carte 10/11/2023 16/11/2023
Envoi de la carte 17/11/2023 17/11/2023
Test de la carte 27/11/2023 28/11/2023
Réalisation Code pour Capteur... 22/11/2023 24/11/2023
Réalisation Code pour Capteur... 22/11/2023 24/11/2023
Faire un serveur Web 23/11/2023 06/12/2023
Préparation Orale 30/11/2023 06/12/2023
Rédaction Dossier Technique 30/11/2023 20/12/2023
Zoom avant | Zoom arrière
novembre 2023
Semaine 46
Semaine 47
Semaine 48

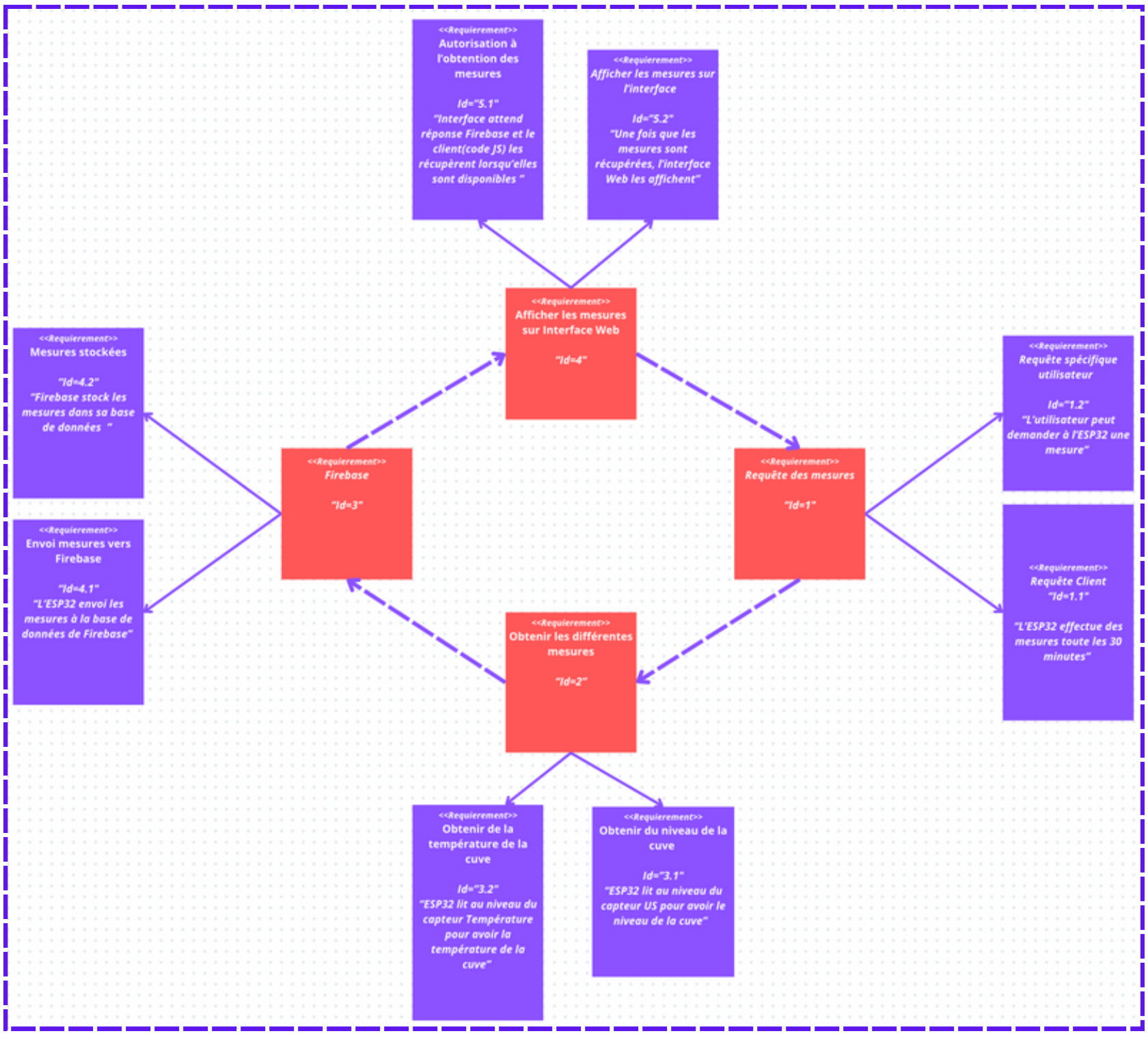
Annexes


Start
Connexion Wifi
Mesure des grandeurs (distance + température + humidité)
Stockage des données mesurées sur Firestore
Deep Sleep pour une durée de 30 minutes
Réveil du mode Deep Sleep

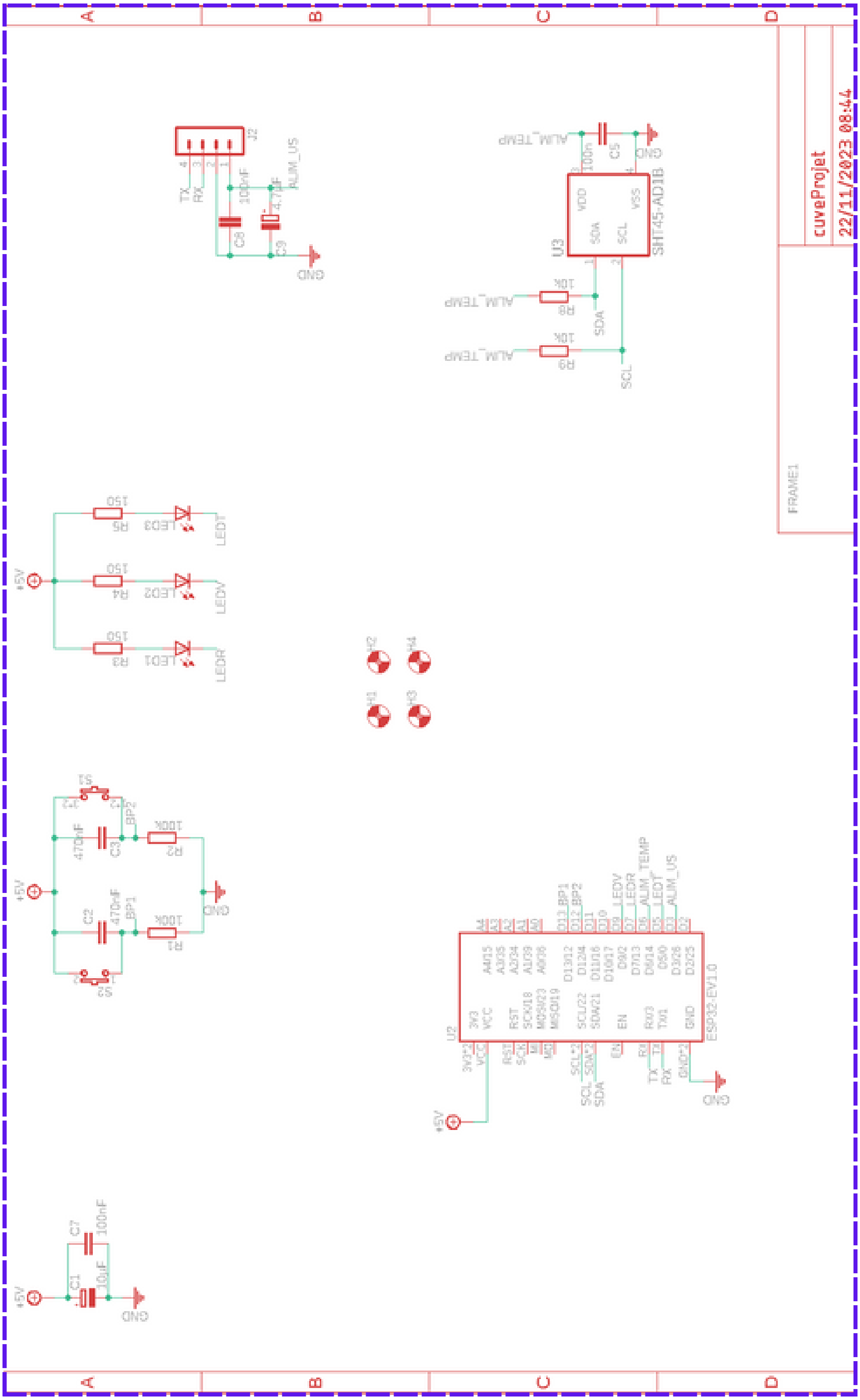
Annexes


CLIENT WEB
Connexion Firestore
Connexion API Météo
Récupération des données Firestore
Demande à l'utilisateur de renseigner sa position
Effectue un calcul si nécessaire
Récupère les datas grâce à l'API
RISQUE DE PLUIE
Préviens l'utilisateur que la cuve va se remplir
PAS DE RISQUE DE PLUIE
Affiche les données sur la page Web
Affiche les datas

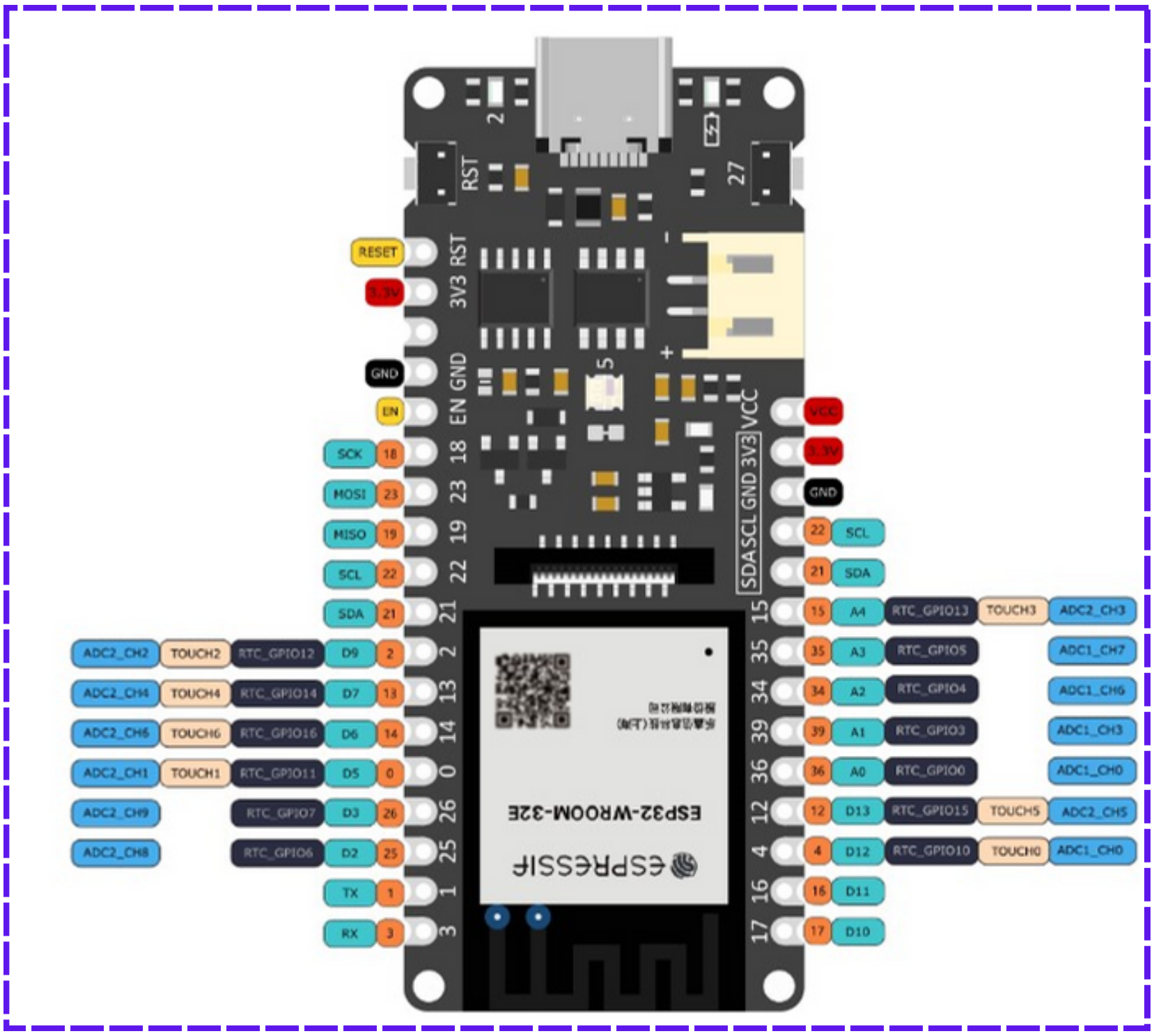
Annexes


<<Requierement>>
Autorisation à l'obtention des mesures
Id="5.1"
"Interface attend réponse Firebase et le client(code JS) les récupèrent lorsqu'elles sont disponibles "
<<Requierement>>
Afficher les mesures sur l'interface
Id="5.2"
"Une fois que les mesures sont récupérées, l'interface Web les affichent"
<<Requierement>>
Afficher les mesures sur Interface Web
"Id=4"
<<Requierement>>
Mesures stockées
"Id=4.2"
"Firebase stock les mesures dans sa base de données "
<<Requierement>>
Firebase
"Id=3"
<<Requierement>>
Requête des mesures
"Id=1"
<<Requierement>>
Requête spécifique utilisateur
Id="1.2"
"L'utilisateur peut demander à l'ESP32 une mesure"
<<Requierement>>
Requête Client
"Id=1.1"
"L'ESP32 effectue des mesures toute les 30 minutes"
<<Requierement>>
Envoi mesures vers Firebase
"Id=4.1"
"L'ESP32 envoi les mesures à la base de données de Firebase"
<<Requierement>>
Obtenir les différentes mesures
"Id=2"
<<Requierement>>
Obtenir de la température de la cuve
Id="3.2"
"ESP32 lit au niveau du capteur Température pour avoir la température de la cuve"
<<Requierement>>
Obtenir du niveau de la cuve
Id="3.1"
"ESP32 lit au niveau du capteur US pour avoir le niveau de la cuve"

Annexes

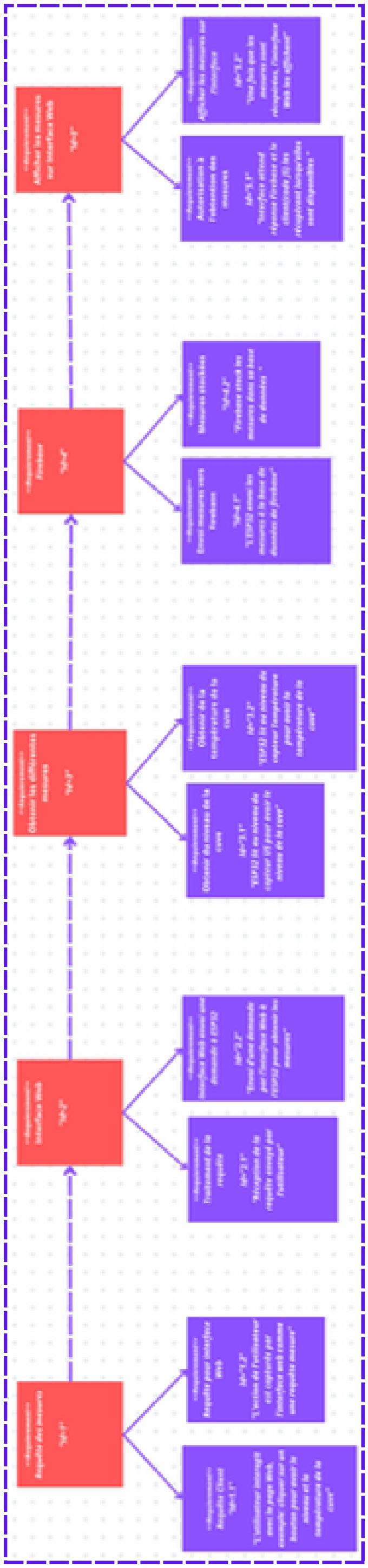

A
B
C
D
cuveProjet
22/11/2023 08:44
FRAME1
J2
TX
RX
100nF
4.7uF
ALIM_US
C8
C9
GND
ALIM_TEMP
100n
C5
GND
U3
SHT45-AD1B
VDD
VSS
SDA
SCL
10K
R8
R9
SDA
SCL
150
R5
R4
R3
LED3
LED2
LED1
LEDT
LEDV
LEDR
+5V
H2
H4
H1
H3
S1
S2
47nF
C3
C2
BP2
BP1
100K
R2
R1
C7
100nF
C1
10µF
U2
ESP32-EV1.0
LEDV
LEDR
ALIM_TEMP
LEDT
ALIM_US

Annexes

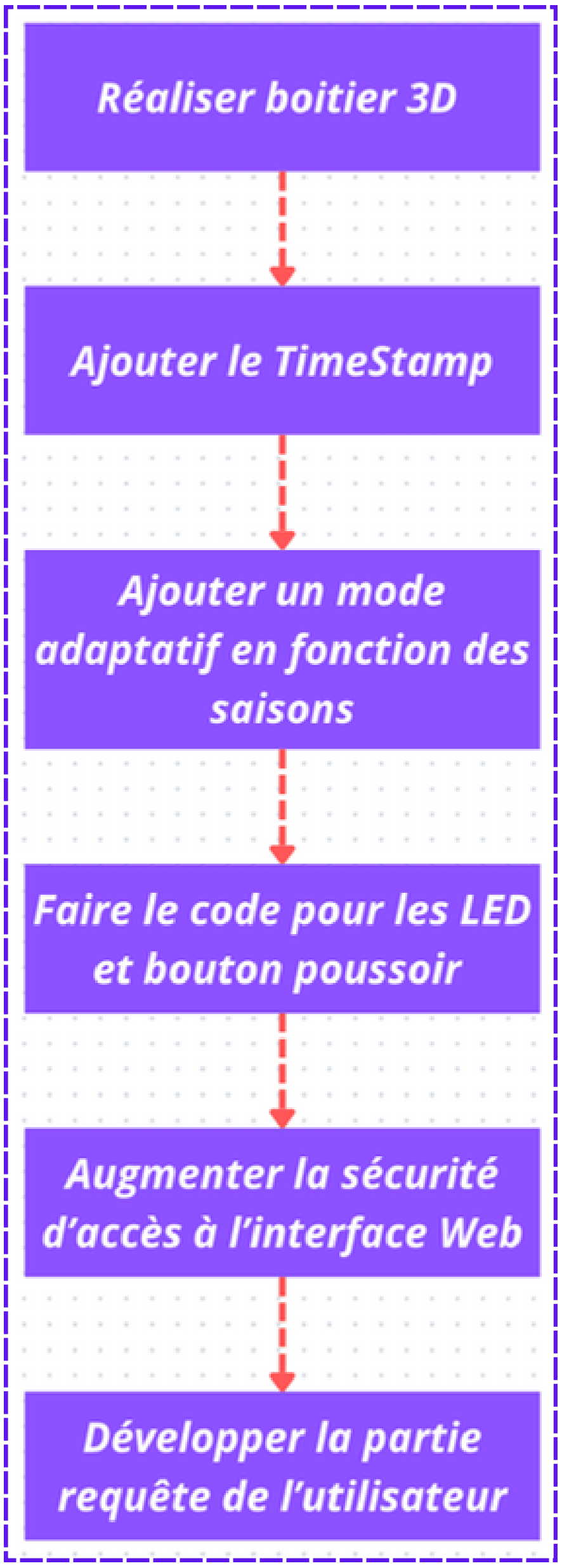

RESET
3.3V
GND
EN
SCK 18
MOSI 23
MISO 19
SCL 22
SDA 21
ADC2_CH2 TOUCH2 RTC_GPIO12 D9 2
ADC2_CH4 TOUCH4 RTC_GPIO14 D7 13
ADC2_CH6 TOUCH6 RTC_GPIO16 D6 14
ADC2_CH1 TOUCH1 RTC_GPIO11 D5 0
ADC2_CH9 RTC_GPIO7 D3 26
ADC2_CH8 RTC_GPIO6 D2 25
TX 1
RX 3
2 RST 3V3 GND EN 18 23 19 22 21
27 5
SDA SCL GND 3V3 VCC
VCC
3.3V
GND
22 SCL
21 SDA
15 A4 RTC_GPIO13 TOUCH3 ADC2_CH3
35 A3 RTC_GPIO5 ADC1_CH7
34 A2 RTC_GPIO4 ADC1_CH6
39 A1 RTC_GPIO3 ADC1_CH3
36 A0 RTC_GPIO0 ADC1_CH0
12 D13 RTC_GPIO15 TOUCH5 ADC2_CH5
4 D12 RTC_GPIO10 TOUCH0 ADC1_CH0
16 D11
17 D10
ESP32-WROOM-32E
ESPRESSIF

GPIO 2	2/D9	Used as input or output	ADC2_CH2		For controlling onboard LED by outputting digital signal

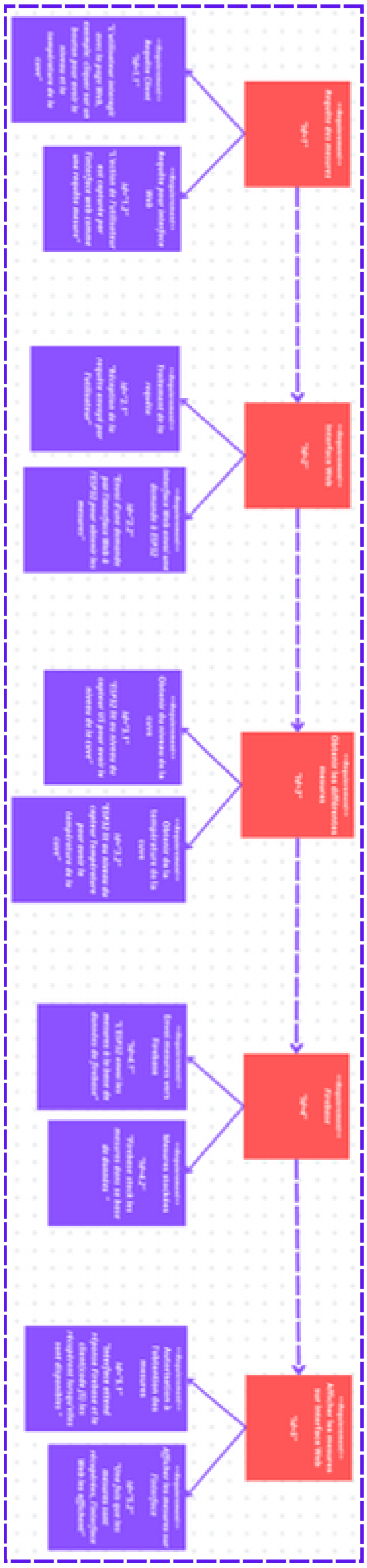
Annexes



Annexes


Réaliser boitier 3D
Ajouter le TimeStamp
Ajouter un mode adaptatif en fonction des saisons
Faire le code pour les LED et bouton poussoir
Augmenter la sécurité d'accès à l'interface Web
Développer la partie requête de l'utilisateur

Annexes



Sources

- https://www.digikey.fr/fr/products/detail/sensirion-ag/SHT41-AD1B-R2/15296591
- https://www.gotronic.fr/art-capteur-a-ultrasons-etanche-sen0311-32230.htm
- https://fr.rs-online.com/web/p/indicateurs-led-pcb/1699747
- https://fr.rs-online.com/web/p/boutons-poussoirs/2183924
- https://developers.googleblog.com/2016/05/firebase-expands-to-become-unified-app.html
- https://www.geeksforgeeks.org/firebase-introduction/
- https://firebase.google.com/docs/firestore/using-console?hl=fr
- https://bluewhaleapps.com/blog/7-reasons-to-choose-google-cloud-firestore-as-your-database-solution
- https://fr.rs-online.com/web/p/piles-rechargeables-taille-speciale/1449405?cm_mmc=FR-PLA-DS3A-_-google-_-CFS_FR_FR_RS+PRO_PO4700199950-_-Batteries+%26+Chargeurs-_-1449405&matchtype=&pla-2205821052666&gad_source=1&gclid=Cj0KCQiA7OqrBhD9ARIsAK3UXh0VOCFOIlgiv-BA5qXJG4pzgOM_XizCxdOOjZhqjrl-sWTJi_D51dkaAhlMEALw_wcB&gclsrc=aw.ds
- https://wiki.dfrobot.com/FireBeetle_Board_ESP32_E_SKU_DFR0654