



Rapport de Projet Capteur TTGO T-Display

Adam EL Horri, 2230500
Thomas Razafimbahoaka, 21912146
UE : HAI912I

Développement mobile, IoT et embarqué
2024-2025

Table des matières

1	Introduction	2
1.1	Contexte du Projet	2
1.2	Objectifs	2
2	Environnements	2
2.1	Environnement de Développement	2
2.2	Bibliothèques Utilisées	2
3	Capteur TTGO T-Display	2
3.1	Caractéristiques Techniques	2
3.2	Réalisation du Montage	3
3.3	Mise en Place du Système via Arduino	5
4	Fonctionnement et Variables du TTGO T-Display	6
4.1	Structure du Projet Arduino	6
4.2	Structure Générale et Variables Globales	7
4.3	Logique de Fonctionnement en Temps Réel	7
4.4	Résumé et Points Clés	8
4.5	Documentation Globale du Fonctionnement	9
4.6	Rôle des Variables Globales	10
5	API REST	10
5.1	Architecture et Conception de l'API	10
5.2	Format des Messages et Gestion JSON	10
5.3	Sécurisation et Gestion des Requêtes CORS	11
5.4	Intégration avec les Services Réseau et OTA	11
5.5	Exemples d'Utilisation et Scénarios	11
6	Affichage	11
6.1	Initialisation et Configuration de l'Écran	11
6.2	Affichage des Informations de Connexion	13
6.3	Affichage des Données des Capteurs	13
6.4	Schémas et Illustrations	13
7	Bluetooth	14
7.1	Configuration du BLE	14
7.2	Mise en Place et Lancement du Serveur BLE	14
7.3	Traitement et Transmission des Identifiants WiFi	15
8	Conclusion	15
	Bibliographie	15

1 Introduction

1.1 Contexte du Projet

Le présent projet consiste à développer une application IoT reposant sur le capteur TTGO T-Display basé sur l'ESP32. L'objectif principal est de mettre en œuvre une API RESTful permettant de contrôler des LED en fonction des données mesurées par des capteurs de température et de luminosité. Le système utilise également l'écran LCD intégré pour afficher en temps réel ces données, améliorant ainsi la visualisation locale et le suivi du dispositif.

1.2 Objectifs

Les objectifs de ce projet se déclinent ainsi :

- Mettre en place une API RESTful pour interroger et piloter les composants du système.
- Intégrer des capteurs analogiques (thermistor et photoresistor) pour mesurer la température et la luminosité.
- Offrir une gestion du contrôle des LED en mode automatique (basé sur des seuils) ou manuel (commandé via API).
- Implémenter une interface réseau flexible via WiFi, avec une solution alternative par Bluetooth Low Energy (BLE) pour la configuration.
- Assurer la persistance des paramètres importants à l'aide d'un stockage non volatile.

2 Environnements

2.1 Environnement de Développement

Le développement s'est déroulé à l'aide de l'Arduino IDE, choisi pour sa simplicité d'intégration avec l'ESP32 et ses bibliothèques dédiées. Cet environnement a facilité la compilation, le déploiement et le débogage du firmware sur le TTGO T-Display. La compatibilité avec la plateforme ainsi que la communauté d'utilisateurs ont été des atouts majeurs dans l'avancement du projet.

2.2 Bibliothèques Utilisées

Pour répondre aux différents besoins du système, plusieurs bibliothèques tierces ont été intégrées :

- **WebServer** : Gère la création du serveur HTTP embarqué sur l'ESP32, permettant d'accéder aux diverses routes de l'API.
- **WiFi** : Permet à l'ESP32 de se connecter à un réseau sans fil, une étape cruciale pour l'hébergement de l'API et la communication avec d'autres dispositifs.
- **TFT_eSPI** : Assure le contrôle de l'écran LCD et la gestion de l'affichage graphique, essentiel pour la visualisation des mesures en temps réel.
- **ArduinoJson** : Facilite le traitement des données en format JSON, utilisé pour structurer les réponses de l'API.
- **Preferences** : Gère la persistance des données telles que le SSID, le mot de passe WiFi et les seuils de mesure.
- **BLEDevice, BLEUtils, BLEServer** : Ces bibliothèques offrent la gestion du Bluetooth Low Energy, utilisé pour configurer dynamiquement la connexion WiFi.

3 Capteur TTGO T-Display

3.1 Caractéristiques Techniques

Le capteur TTGO T-Display intègre une plateforme complète reposant sur un microcontrôleur ESP32. Ses principales caractéristiques sont les suivantes :

- Une carte basée sur l'ESP32 permettant l'usage du WiFi (standards 802.11 b/g/n, débit théorique jusqu'à 150 Mb/s) et du Bluetooth (4.2 BLE).
- Un écran TFT OLED de 1,14 pouces fournissant une résolution de 240 par 135 pixels, idéal pour l'affichage en temps réel.
- Une mémoire intégrée de 520 Ko, suffisamment performante pour le traitement des opérations embarquées.
- Un port USB-C qui facilite à la fois l'alimentation et la programmation.

- Plusieurs interfaces (ports A/D, SPI, UART, I2C, et GPIO) qui permettent la connexion aisée de divers capteurs et actionneurs.
 - Trois boutons physiques sont présents : deux boutons sur les ports GPIO (affectés aux pins 0 et 35) et un bouton de réinitialisation.

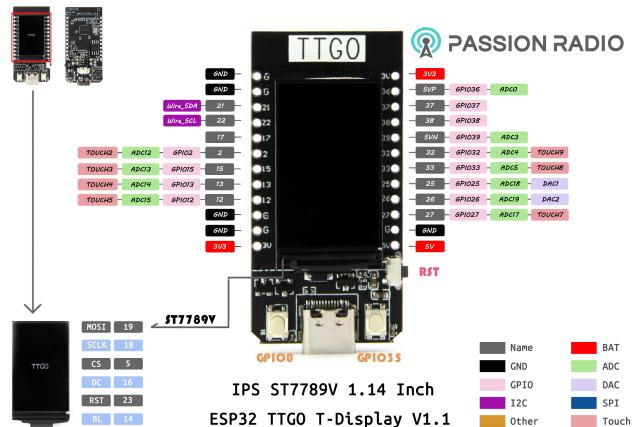


Figure 1 : Specifications des broches du TTGO.

3.2 Réalisation du Montage

Avant toute implémentation, la mise en place du circuit a été primordiale pour alimenter correctement les LED ainsi que les capteurs (thermistors et photoresistors). La conception du montage a été réalisée à l'aide d'un schéma Fritzing, dont la modélisation de la carte TTGO a été effectuée par CirkitDesigner.

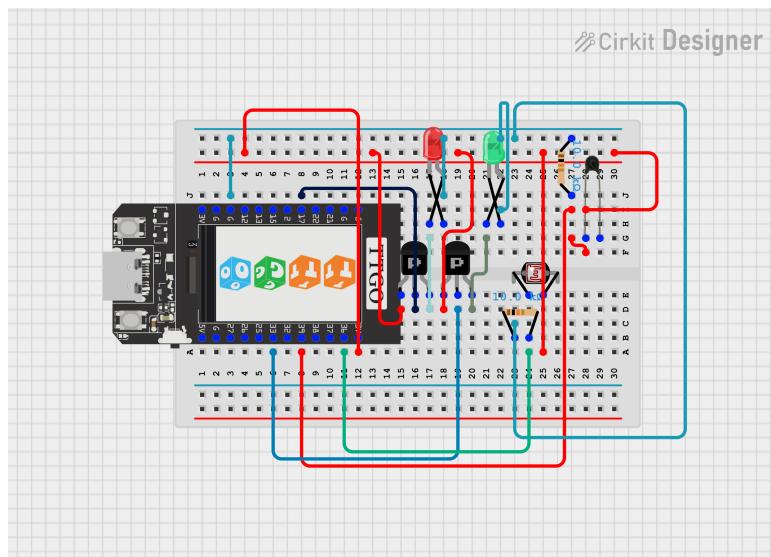


Figure 2 : Schéma Fritzing du montage.

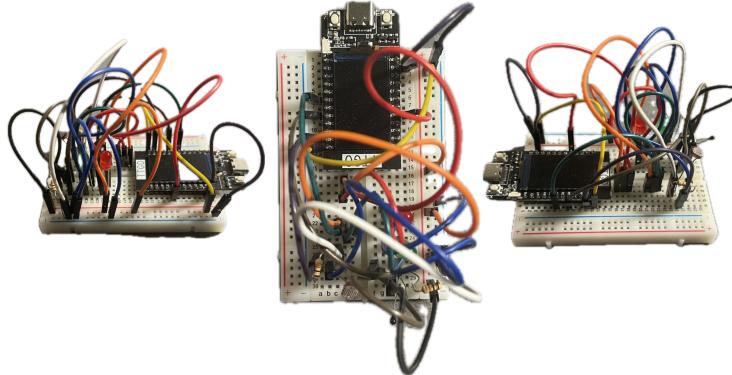


Figure 3 : Montage du TTGO.

Dans ce montage, deux LED sont connectées à l'aide d'un transistor pour chacune. Ainsi, la LED rouge est commandée par le pin 17, tandis que la LED verte l'est via le pin 33. Noter que, dans cette partie du montage, l'utilisation de résistances n'est pas indispensable, une LED nécessitant typiquement une alimentation en 3V.

Circuit du Photoresistor

Le capteur de lumière, appelé aussi photocell ou photoresistor, est branché sur le pin 36 qui se situe sur le port ADC1 (permettant l'usage des fonctions `analogRead()` tout en facilitant la communication WiFi). Son principe de fonctionnement est basé sur l'inversement de la résistance : plus l'intensité lumineuse augmente, plus la résistance diminue, et inversement.

Pour mesurer ce signal, un diviseur de tension est utilisé. Le montage s'organise de la manière suivante :

- La résistance variable (R_1) correspond au photoresistor.
- Une résistance fixe de 10 k Ω (R_2) est utilisée en parallèle.
- La tension d'alimentation fournie est de 3,3 V, appliquée sur le montage.

Le courant électrique se déplace de la source V_{in} (3,3 V) à travers R_1 (photoresistor) puis R_2 (10 k Ω) avant de rejoindre la masse (GND). Pour déterminer la tension de sortie V_{out} (connectée au pin 36), on applique la loi des mailles (de Kirchhoff) conjointement avec la loi d'Ohm.

On définit les tensions aux bornes de R_1 et R_2 :

$$V_1 = R_1 \cdot I \quad \text{et} \quad V_2 = R_2 \cdot I$$

où I est le courant circulant dans le circuit. La loi de Kirchhoff indique que :

$$V_{in} = V_1 + V_2 = R_1 \cdot I + R_2 \cdot I = I \cdot (R_1 + R_2)$$

D'où l'intensité du courant s'exprime par :

$$I = \frac{V_{in}}{R_1 + R_2}$$

La tension mesurée par le photoresistor (résistif) sera donc :

$$V_{out} = I \cdot R_2 = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Pour des valeurs typiques, on peut effectuer le calcul suivant :

- Si $R_1 = 0$ (lorsque le capteur est fortement éclairé) alors :

$$V_{out} = 3.3 \times \frac{10\,000}{0 + 10\,000} = 3.3 \text{ V}$$

- Si $R_1 = 10\,000\,\Omega$ alors :

$$V_{out} = 3.3 \times \frac{10\,000}{10\,000 + 10\,000} = 1.65 \text{ V}$$

Cette tension, comprise entre 1.65 V et 3.3 V, est convertie par l'ADC du TTGO et renvoyée sous forme d'une valeur numérique comprise entre 0 et 4095.

Montage du Thermistor

Le thermistor, utilisé pour mesurer la température, est connecté au pin 39, également en ADC1. Son montage suit un principe similaire à celui du photodiode : un pont diviseur de tension. Dans ce cas précis, la résistance variable (R_1) du thermistor varie sur une plage comprise entre 0 et environ $2,2\,\text{k}\Omega$, et la résistance fixe demeure $R_2 = 10\,000\,\Omega$.

- On applique la même formule générale pour calculer V_{out} :

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

Ainsi :

- Pour $R_1 = 0\,\Omega$ (cas extrême) :

$$V_{out} = 3.3 \times \frac{10\,000}{0 + 10\,000} = 3.3 \text{ V}$$

- Pour $R_1 = 2200\,\Omega$:

$$V_{out} = 3.3 \times \frac{10\,000}{2200 + 10\,000} \approx 2.71 \text{ V}$$

La tension obtenue, variant entre environ 2.71 V et 3.3 V, sera traitée pour en déduire la température correspondante. (La conversion en degrés Celsius s'effectue à partir de la résistance mesurée via une formule dérivée de Steinhart-Hart, mais le calcul des coefficients ainsi que les ajustements ne sont pas détaillés ici, étant remplacés par une calibration empirique dans notre projet.)

3.3 Mise en Place du Système via Arduino

Pour programmer le TTGO T-Display avec Arduino, quelques étapes de configuration sont indispensables :

- Installation du gestionnaire de cartes pour l'ESP32 afin de permettre la compilation et le téléchargement sur la carte.
- Modification du fichier de configuration de la bibliothèque TFT_eSPI (notamment dans le fichier `User_Setup.h`). Il s'agit de désactiver le setup par défaut et d'activer l'option correspondant aux paramètres du TTGO T-Display.
- Vérification des droits d'accès aux ports (sous Linux, par exemple, il est nécessaire de changer les permissions du port série avec la commande :

```
sudo chmod a+r /dev/ttyACM0
```

afin d'éviter les erreurs lors du téléchargement).

Essais et Paramétrages

Avant l'assemblage complet du système, chaque bibliothèque a été testée individuellement. Des exemples de code ont été fournis pour :

- Vérifier la communication avec l'écran TFT et ajuster l'interface graphique.
- Valider la connexion WiFi avec le TTGO.
- Confirmer le traitement des capteurs via les fonctions `analogRead()`.

Les paramètres définis dans Arduino (p.ex. choix du port série, débit de transmission, etc.) ont été vérifiés pour assurer un téléchargement sans erreur et une communication fiable entre le TTGO et l'IDE.

Paramètres choisis :

- Board : TTGO T1
- Port : COM3
- CPU Frequency : 240MHz (WiFi/BT)
- Core Debug Level : None
- Erase All Flash Before Sketch Upload : Enabled
- Flash Frequency : 80MHz
- Flash Mode : DIO
- Flash Size : 16MB (128Mb)
- Partition Scheme : Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
- Upload Speed : 921600

4 Fonctionnement et Variables du TTGO T-Display

4.1 Structure du Projet Arduino

Le projet Arduino pour l'ESP32 est organisé de manière modulaire, facilitant ainsi la maintenance, l'extensibilité et la compréhension du code. Chaque module est constitué de fichiers source (`.cpp`) et d'en-tête (`.h`) dédiés, définissant des fonctionnalités spécifiques du système. Voici un aperçu détaillé de la structure du projet :

Fichiers Principaux

- **api_rest_corrected_final.ino** : Point d'entrée principal du programme contenant les fonctions `setup()` et `loop()` qui orchestrent l'exécution globale du système.

Gestion Globale

- **globals.h** : Déclare toutes les variables globales, constantes (définies avec `#define`) et inclut les bibliothèques nécessaires pour l'ensemble du projet.
- **globals.cpp** : Définit et initialise les variables globales déclarées dans `globals.h`, assurant ainsi leur disponibilité à travers les différents modules.

Modules de Communication

- **ble_module.h** : Déclare les fonctions nécessaires pour l'initialisation et la gestion de la communication Bluetooth Low Energy (BLE).
- **ble_module.cpp** : Implémente les fonctions d'initialisation du BLE ainsi que les callbacks utilisés pour la communication sans fil.
- **wifi_module.h** : Déclare les fonctions dédiées à la connexion et à la gestion du réseau WiFi.
- **wifi_module.cpp** : Implémente les fonctions permettant de se connecter au réseau WiFi et de configurer les paramètres IP.

Gestion des Requêtes HTTP et CORS

- **cors_module.h** : Déclare les fonctions pour gérer les en-têtes CORS et les requêtes HTTP de type OPTIONS.
- **cors_module.cpp** : Implémente les fonctions permettant d'ajouter les en-têtes CORS nécessaires et de traiter les requêtes OPTIONS pour assurer la compatibilité avec les navigateurs.
- **server_routes.h** : Déclare les fonctions responsables de la définition et de la gestion des différentes routes du serveur HTTP.
- **server_routes.cpp** : Implémente les routes HTTP (GET, POST) ainsi que leurs gestionnaires (*handlers*) permettant d'interagir avec le système via l'API REST.

Mises à Jour Over-the-Air (OTA)

- **ota_module.h** : Déclare les fonctions nécessaires pour configurer et gérer les mises à jour du firmware via OTA.
- **ota_module.cpp** : Implémente la gestion des mises à jour OTA, incluant le suivi de la progression et le traitement des éventuelles erreurs lors du processus de mise à jour.

Affichage et Interface Utilisateur

- **display_module.h** : Déclare les fonctions destinées à gérer l'affichage des données sur l'écran TFT.
- **display_module.cpp** : Implémente les fonctions responsables du dessin et de l'affichage des valeurs des capteurs sur l'écran, offrant ainsi une interface visuelle intuitive.

Gestion des Capteurs

- **sensors_module.h** : Déclare les fonctions nécessaires pour la lecture des données des capteurs et la vérification des seuils définis.
- **sensors_module.cpp** : Implémente la logique permettant de lire les valeurs des capteurs (theristor et photoresistor) et de déterminer si les seuils configurés sont dépassés.

Contrôle des LEDs

- **led_module.h** : Déclare les fonctions dédiées à la gestion des états des LEDs, que ce soit en mode manuel ou automatique.
- **led_module.cpp** : Implémente les fonctions de contrôle des LEDs en fonction du mode sélectionné et des données provenant des capteurs, permettant ainsi une indication visuelle des états du système.

Résumé de la Structure Modulaire

Cette organisation modulaire permet une séparation claire des responsabilités, facilitant ainsi le développement, le débogage et l'extension des fonctionnalités du système. Chaque module peut être développé et testé indépendamment, assurant une meilleure maintenabilité et évolutivité du projet.

4.2 Structure Générale et Variables Globales

Le programme s'appuie sur des variables et constantes définies en début de fichier, assurant la gestion de l'état du système et la synchronisation des tâches périodiques.

Variables de Configuration et d'État

- **Paramètres Réseau et Préférences :**
 - `wifi_ssid` et `wifi_password` définissent respectivement le réseau WiFi et le mot de passe.
 - `globalUserControl` (booléen) détermine le mode de gestion des LED : `false` pour le mode automatique (piloté par les capteurs) et `true` pour le mode manuel (commandé via l'API REST).
- **Configuration Matérielle** : Les constantes `LIGHT_SENSOR_PIN` (pin 36), `THERMISTOR_PIN` (pin 39), `LED_PIN_RED` (pin 17) et `LED_PIN_GREEN` (pin 33) désignent les connexions physiques des capteurs et des LED.
- **Paramètres de Temporalisation** : Les variables telles que `ledIntervalA`, `ledIntervalB` et `displayInterval` fixent la fréquence d'actualisation de l'affichage et le rythme de clignotement des LED. La durée d'activation, via `blinkDuration`, est également configurable et stockée de manière persistante à l'aide de l'objet `Preferences`.

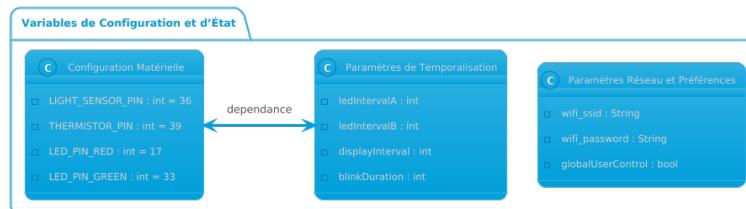


Figure 8 : Configuration des variables globales.

Acquisition et Conversion des Données Capteurs

Deux capteurs sont intégrés au système, chacun connecté à une entrée analogique distincte :

- **Photodiode :**
 - Connecté sur le pin 36, il fait partie d'un pont diviseur de tension associé à une résistance fixe de 10 kΩ.
 - L'ADC de l'ESP32 convertit la tension mesurée en une valeur numérique comprise entre 0 et 4095.
- **Thermistor :**
 - Placé sur le pin 39, il utilise également un pont diviseur de tension.
 - La valeur lue est convertie en température via l'équation de Steinhart-Hart (avec les coefficients A, B, et C). La température en Kelvin est ensuite ajustée (soustraction de 273.15 et correction expérimentale de 53.0°) pour obtenir la température en degrés Celsius.

4.3 Logique de Fonctionnement en Temps Réel

Le cœur du programme repose sur une boucle principale (`loop()`) qui orchestre l'ensemble des tâches en temps réel.

Acquisition et Traitement des Données Capteurs

À intervalles définis par `displayInterval`, la fonction `displaySensorData()` exécute les opérations suivantes :

1. Lecture de la valeur brute du thermistor via `analogRead(THERMISTOR_PIN)` et conversion en température.
2. Lecture de la valeur du photoresistor via `analogRead(LIGHT_SENSOR_PIN)`.
3. Actualisation de l'affichage graphique par la fonction `drawSensorDataCircle()`, qui représente les mesures sous forme de cercles et de valeurs numériques.

Contrôle des LED

Les LED sont pilotées via deux fonctions dédiées (`updateLedStateA()` pour la LED rouge et `updateLedStateB()` pour la LED verte). La commande dépend du mode de contrôle :

- **Mode Manuel** (`globalUserControl` activé) :
 - Les variables `ledOnOffA` et `ledOnOffB` définissent l'état des LED.
 - La commande `digitalWrite()` modifie l'état, puis celui-ci est sauvegardé en mémoire pour persistance.
- **Mode Automatique** (`globalUserControl` désactivé) :
 - La fonction `verifyThresholdActivation()` compare les mesures aux seuils définis (par exemple, `tempThreshold` pour la température).
 - En fonction du résultat, les LED sont allumées ou éteintes pour signaler des anomalies.

Pour assurer un clignotement cohérent, le temps écoulé depuis le dernier changement d'état est stocké dans des variables `unsigned long` (`previousLoopTimeA` et `previousLoopTimeB`). La gestion des LED est illustrée dans l'image ci-dessous.

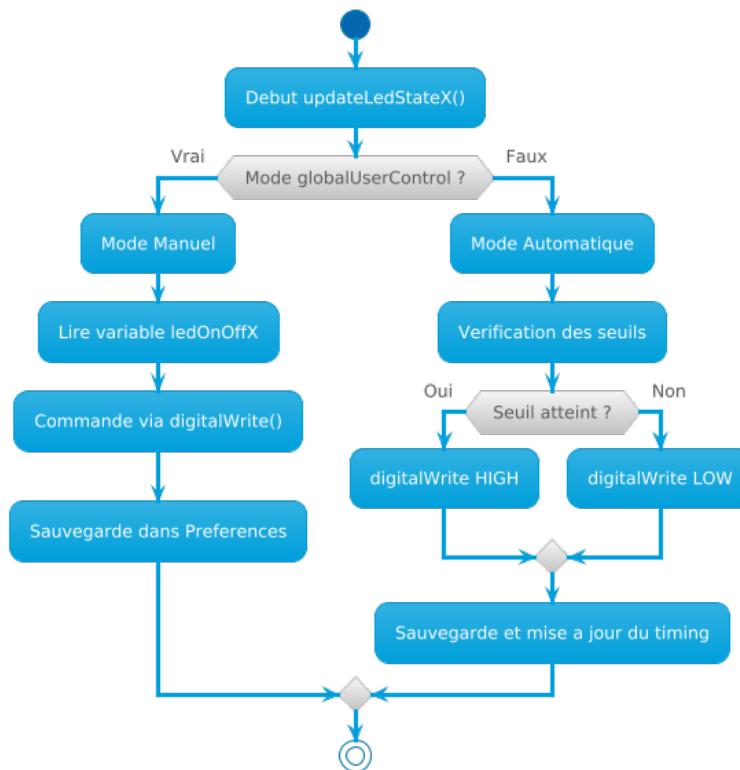


Figure 9 : Gestion des LED en mode manuel et automatique.

Synchronisation des Tâches Réseau et OTA

Parallèlement, la boucle principale gère :

- La vérification de l'état de la connexion WiFi via `WiFi.status()`.
- Le traitement des requêtes HTTP entrantes avec `server.handleClient()` pour permettre les interactions via l'API REST (consultation des mesures, modification des paramètres, etc.).
- La gestion des mises à jour Over The Air (OTA) via `ArduinoOTA.handle()`, autorisant des mises à jour du firmware sans interrompre le fonctionnement du système.

4.4 Résumé et Points Clés

L'architecture du TTGO T-Display repose sur quatre modules principaux :

1. **Acquisition des Données** : Lecture périodique des capteurs pour surveiller la température et la luminosité.

2. **Interface Visuelle** : Mise à jour régulière de l'écran TFT afin de présenter graphiquement et numériquement les mesures.
 3. **Contrôle des LED** : Gestion dynamique des LED en mode manuel (commandé via l'API REST) ou automatique (piloté par les seuils des capteurs).
 4. **Communication et Maintenance** : Traitement des requêtes HTTP pour les interactions externes et mises à jour OTA pour assurer la maintenance du système.

L'organisation modulaire et la persistance des réglages (via `Preferences`) garantissent une application robuste et évolutive, apte à intégrer de futures améliorations (nouvelles interfaces, calibrations avancées, etc.) sans modifier l'ossature principale du code.

4.5 Documentation Globale du Fonctionnement

Le cycle de vie du système se décompose en deux phases complémentaires :

— Phase d'Initialisation :

- Configuration de l'écran TFT et des broches des LED.
 - Lecture des préférences stockées (réglages WiFi, seuils, etc.).
 - Tentative de connexion au réseau WiFi, avec basculement sur le mode Bluetooth en cas d'échec afin de permettre la saisie des paramètres réseau.

— Phase d'Exécution Continue (boucle principale) :

1. Mise à jour du firmware via OTA.
 2. Acquisition périodique des valeurs issues des capteurs.
 3. Actualisation de l'affichage sur l'écran TFT.
 4. Contrôle des LED en fonction du mode sélectionné et des seuils configurés.
 5. Traitement des requêtes HTTP via l'API RESTful.

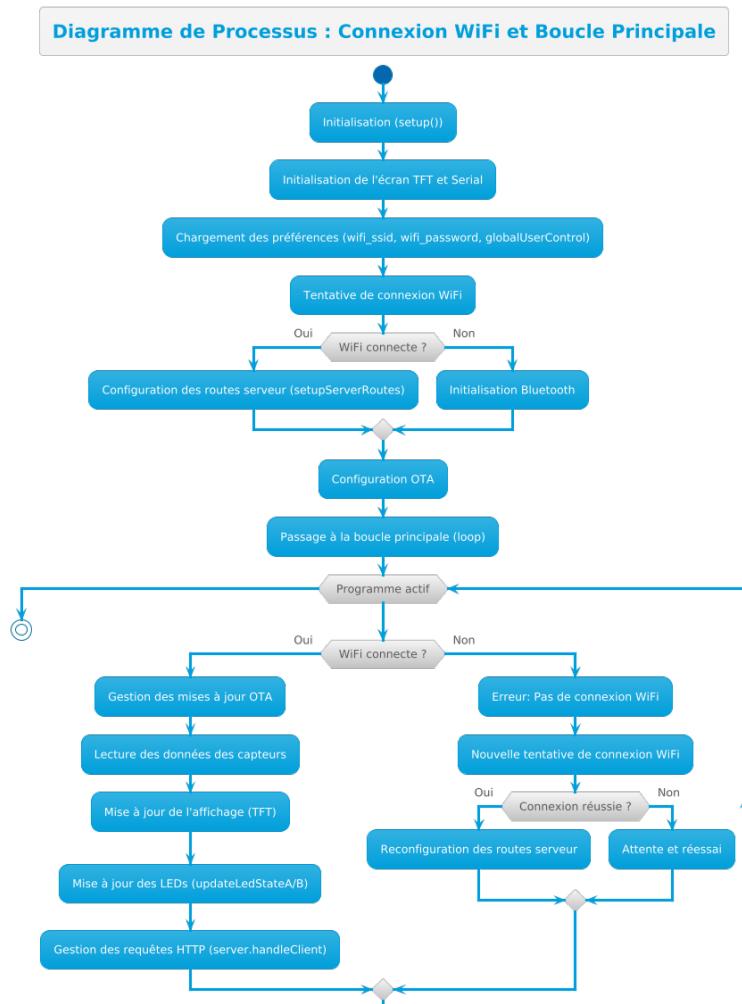


Figure 10 : Cycle d'initialisation et d'exécution continue.

4.6 Rôle des Variables Globales

Les variables globales interviennent à plusieurs niveaux pour :

- **Assurer la persistance des Préférences** : Conserver les réglages essentiels (SSID, mot de passe, seuils, etc.) même après redémarrage.
- **Configurer le Matériel** : Définir les pins utilisées pour les capteurs et les LED.
- **Gérer la Temporalisation** : Réguler la fréquence d'actualisation de l'affichage et le clignotement des LED.
- **Déterminer le Mode d'Opération** : La variable `globalUserControl` sélectionne le mode de gestion des LED (manuel ou automatique).

5 API REST

L'API REST constitue l'interface principale permettant l'interaction avec le système embarqué sur l'ESP32. Elle offre un ensemble de routes HTTP pour interroger les valeurs des capteurs, contrôler l'état des LED et modifier certains paramètres de fonctionnement (tels que les intervalles de clignotement et les seuils de déclenchement). Ce section détaille l'architecture, le fonctionnement et la sécurisation de l'API.

5.1 Architecture et Conception de l'API

L'API REST est implémentée grâce à la bibliothèque `WebServer` de l'ESP32 et fonctionne sur le port 80. Elle repose sur une organisation modulaire des routes, où chaque route est associée à un handler spécifique permettant de :

- **Interroger les données des capteurs** (température mesurée via le thermistor et luminosité mesurée via le photoresistor).
- **Lire et modifier l'état des LED** en mode manuel et automatique.
- **Mettre à jour la configuration** (seuils de déclenchement, intervalles de clignotement, mode global de fonctionnement, etc.).

Définition des Routes et des Handlers

Chaque route correspond à une URL spécifique et est associée à une méthode HTTP (GET ou POST). La configuration s'effectue dans la fonction `setupServerRoutes()` qui définit et associe les routes suivantes :

Routes GET Ces routes permettent aux clients (interface web, application mobile, etc.) d'obtenir l'état courant du système :

- `/temperature` : Renvoie la température mesurée par le thermistor, après conversion et ajustement.
- `/light` : Fournit la valeur mesurée par le photoresistor.
- `/ledA` et `/ledB` : Retourne l'état (allumé ou éteint) et la vitesse de clignotement des LED rouge et verte, respectivement.
- `/led` : Renvoie dans une seule réponse JSON les informations relatives aux deux LED.
- `/ledGlobalMode` : Affiche le mode global actuel (manuel ou automatique).

Routes POST Ces routes sont utilisées pour modifier l'état du système ou mettre à jour ses paramètres :

- `/ledChangeA` et `/ledChangeB` : Changent la valeur et la vitesse de clignotement des LED individuellement en mode manuel.
- `/ledChange` : Permet d'envoyer une commande simultanée pour les deux LED.
- `/autoManTemp` et `/autoManLight` : Basculent respectivement le mode automatique pour la température et la luminosité.
- `/autoManTempParams` et `/autoManLightParams` : Met à jour les paramètres (état souhaité et vitesse) pour le mode automatique.
- `/thresholdTemp` et `/thresholdLight` : Configurent les seuils de déclenchement pour la température et la lumière.
- `/ledGlobalMode` (POST) : Change le mode global entre `manual` et `auto`.

5.2 Format des Messages et Gestion JSON

Les réponses de l'API sont formatées en JSON grâce à la bibliothèque `ArduinoJson`. Les objets JSON retournés comportent notamment :

- Un champ `"type"` indiquant la nature de la donnée (par exemple, `"temperature"` ou `"lumiere"`).
- Un champ `"value"` contenant la mesure ou l'état courant.
- Pour les LED, un champ `"speed"` précisant l'intervalle de clignotement.

La fonction `serializeJson()` convertit l'objet JSON en chaîne de caractères pour transmission au client.

5.3 Sécurisation et Gestion des Requêtes CORS

Afin de faciliter les interactions entre différents domaines et de prévenir les problèmes de sécurité liés aux requêtes cross-origin, les en-têtes CORS (Cross-Origin Resource Sharing) sont ajoutés à chaque réponse HTTP. La fonction `addCorsHeaders()` insère notamment :

- `Access-Control-Allow-Origin`: * pour autoriser toutes les origines.
- `Access-Control-Allow-Methods`: GET, POST, OPTIONS pour indiquer les méthodes acceptées.
- `Access-Control-Allow-Headers`: Origin, Content-Type, Accept pour définir les en-têtes autorisés.

Les requêtes préliminaires de type OPTIONS sont gérées par la fonction `handleCors()`, qui renvoie les en-têtes adéquats et un code HTTP 200, assurant ainsi la compatibilité avec les navigateurs et clients tiers.

5.4 Intégration avec les Services Réseau et OTA

L'API REST fonctionne en étroite collaboration avec le service WiFi de l'ESP32. Une fois la connexion établie, le serveur HTTP démarre et écoute les requêtes des clients. En cas d'échec de la connexion WiFi, le système bascule vers une configuration alternative via BLE, assurant ainsi une résilience accrue.

Par ailleurs, le traitement des mises à jour OTA (Over The Air) s'effectue parallèlement dans la boucle principale, évitant tout conflit avec la gestion des requêtes API et permettant une mise à jour du firmware en toute sécurité.

5.5 Exemples d'Utilisation et Scénarios

Pour illustrer la mise en œuvre de l'API, deux scénarios courants sont présentés :

Lecture d'une Mesure de Température

1. Le client envoie une requête GET à l'adresse `/temperature`.
2. Le handler `handleGetTemperature()` réalise les opérations suivantes :
 - Effectue une lecture analogique sur le thermistor.
 - Convertit la mesure en degrés Celsius (selon une formule basée sur Steinhart-Hart avec correction).
 - Construit un objet JSON contenant le type de donnée et la valeur calculée.
 - Renvoie le JSON formaté au client.

Modification du Mode de Contrôle Global

1. Pour passer en mode manuel, le client envoie une requête POST à l'adresse `/ledGlobalMode` avec le paramètre `mode=manual`.
2. Le handler correspondant met à jour la variable globale `globalUserControl` et enregistre cette configuration en mémoire persistante via `Preferences`.
3. Le serveur renvoie une réponse confirmant le changement de mode.

6 Affichage

L'ESP32 intégré au TTGO T-Display est équipé d'un écran TFT qui sert à afficher en temps réel les relevés des capteurs ainsi que l'état des connexions (WiFi ou Bluetooth). Cette fonctionnalité d'affichage permet à l'utilisateur d'avoir un retour visuel immédiat sur le fonctionnement du système, notamment durant l'étape de connexion initiale.

6.1 Initialisation et Configuration de l'Écran

Avant de pouvoir exploiter l'écran, il est nécessaire d'ajouter la bibliothèque `TFT_eSPI` qui fournit l'ensemble des fonctions de contrôle graphique. L'objet d'affichage est initialisé dès le démarrage du programme via la commande `init()`. La configuration de l'écran s'effectue immédiatement après, comme illustré ci-dessous :

- `setRotation(1)`
Cette commande permet de passer l'écran en mode paysage, assurant ainsi une meilleure disposition de l'information.
- `fillScreen(TFT_BLACK)`
Cette fonction remplit l'écran d'un fond noir, ce qui permet d'effacer le contenu précédent et de repartir sur une base propre.

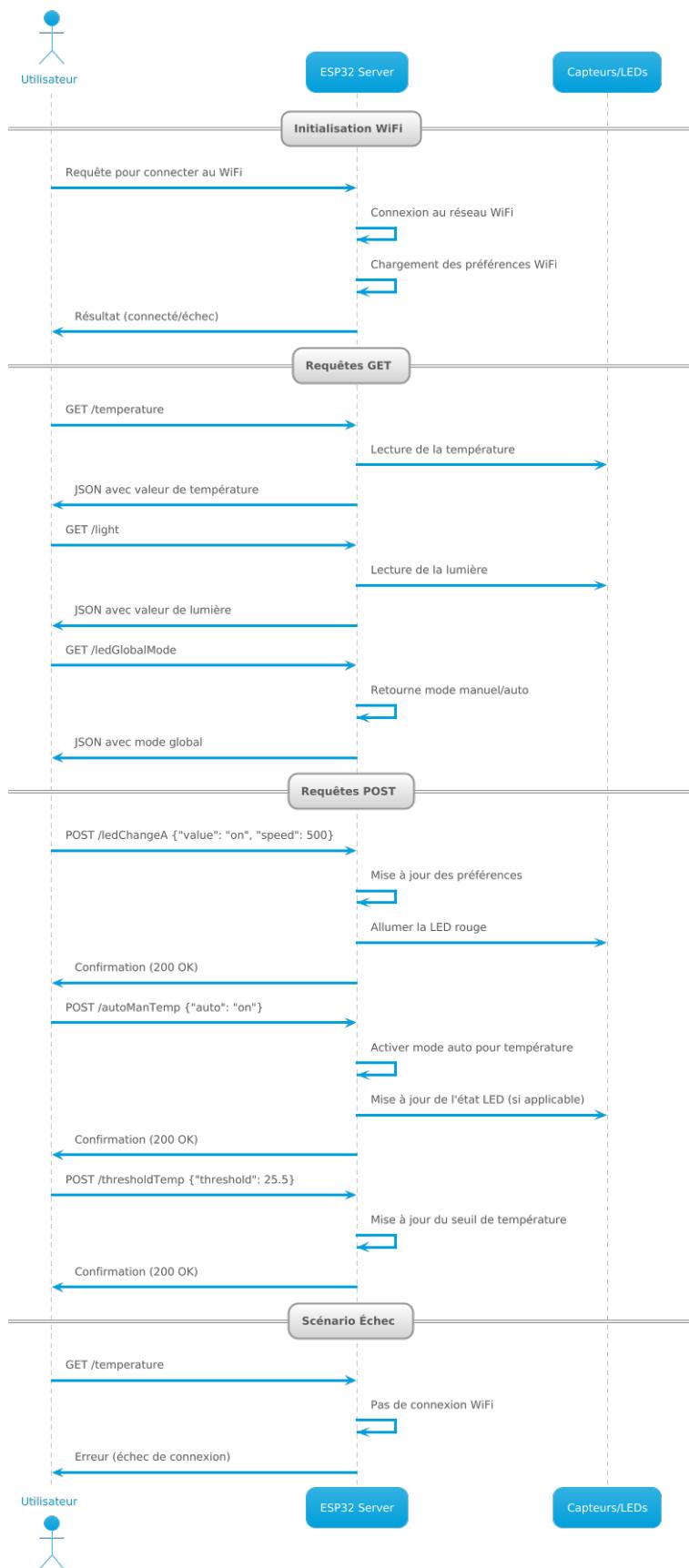


FIGURE 1 – Schéma de traitement des requêtes via l'API REST

— `setTextColor(TFT_WHITE)`

Pour rendre le texte lisible sur le fond noir, la couleur du texte est paramétrée en blanc.

— `setTextSize(1)`

La taille du texte est ajustée à l'aide de cette commande pour que les informations rentrent correctement sur l'écran sans surcharge visuelle.

- **setCursor(x, y)**

La position du curseur détermine l'endroit où le texte débutera. Cette commande est utilisée pour positionner les informations dans des zones précises de l'écran.

Une fois ces commandes exécutées dans la routine d'initialisation, l'écran est prêt à recevoir les informations de mesure et d'état.

6.2 Affichage des Informations de Connexion

Dès le démarrage, le système tente d'établir une connexion via WiFi ou, en cas d'échec, via Bluetooth. Pour informer l'utilisateur de cette phase d'attente, l'écran affiche un message indiquant que le TTGO est en attente de connexion. Par exemple, un message du type « En attente de connexion WiFi... » ou « Démarrage Bluetooth... » est présenté, permettant ainsi à l'utilisateur de connaître l'état du système.

6.3 Affichage des Données des Capteurs

Une fois la connexion établie et les capteurs activés, l'écran affiche les relevés en temps réel. L'implémentation s'inspire de la documentation de la bibliothèque TFT_eSPI et utilise deux fonctions principales pour l'affichage :

- **testdrawcircles(...)** : Cette fonction dessine des cercles sur l'écran. Chaque cercle représente une zone dédiée à l'affichage d'une information issue des capteurs (par exemple, la température ou la luminosité). Le rayon et la couleur des cercles peuvent être paramétrés pour différencier visuellement les types de données.
- **testdrawtext(...)** : Une fois les cercles dessinés, cette fonction est appelée pour placer les valeurs des capteurs à l'intérieur ou à proximité des cercles. Ainsi, par exemple, la température calculée (en °C) est positionnée au centre du cercle correspondant.

L'affichage se fait de manière périodique grâce à une fonction dédiée, `displaySensorData()`, qui vérifie si l'intervalle entre deux mises à jour a été respecté (variable `displayInterval`). À chaque cycle, les valeurs mesurées par les capteurs sont relues, converties, puis affichées sur l'écran via les fonctions de dessin. Cette mise à jour régulière garantit une représentation fidèle et actuelle de l'état ambiant.

6.4 Schémas et Illustrations

Afin de faciliter la compréhension du fonctionnement de l'affichage, plusieurs schémas illustrent le processus :

- **Figure de Connexion** : Cette illustration montre l'écran affichant l'état de la connexion, signalant à l'utilisateur s'il doit attendre ou s'il l'appareil est déjà connecté en WiFi ou via Bluetooth.
- **Schéma d'Affichage des Capteurs** : Un second schéma présente la disposition des informations sur l'écran. On y voit par exemple un cercle affichant la température avec la valeur numérique au centre et un second cercle pour la luminosité.

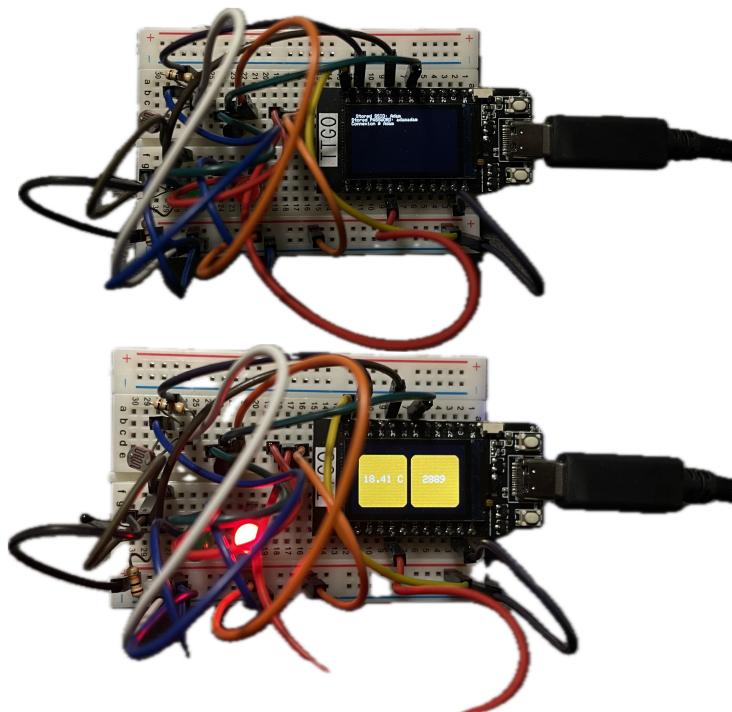


Figure 11 : Affichage de connection et des valeurs des capteurs.

7 Bluetooth

L'utilisation directe du SSID et du mot de passe en clair pour la connexion WiFi présente des failles en termes de sécurité et de flexibilité. Pour remédier à cela, nous avons mis en place une configuration via Bluetooth Low Energy (BLE), permettant ainsi de recevoir dynamiquement les identifiants du réseau WiFi depuis un appareil externe.

7.1 Configuration du BLE

Afin d'implémenter le BLE, plusieurs bibliothèques spécifiques ont été ajoutées au projet. Comme illustré dans la Figure 7.1, celles-ci facilitent la création de services et de caractéristiques nécessaires pour la communication BLE.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

Figure 12 : Inclusion des bibliothèques BLE.

Le BLE repose sur un ensemble de services et de caractéristiques internes. Nous commençons par définir un identifiant unique pour le service (SERVICE UUID) ainsi qu'un identifiant pour la caractéristique (CHARACTERISTIC UUID).

7.2 Mise en Place et Lancement du Serveur BLE

La fonction dédiée, `connectionBluetooth()`, est responsable de la configuration initiale du module BLE. Elle effectue les opérations suivantes :

- Attribution du nom de l'appareil, par exemple, « ESP32 ».
- Création du service BLE en associant l'identifiant unique défini précédemment.
- Création d'une caractéristique, configurée pour autoriser l'écriture parmi d'autres propriétés, et liée à son UUID.

Une fois ces éléments mis en place, le serveur BLE est lancé grâce aux fonctions `start()` et `getAdvertising()`. Ces fonctions placent le module en mode écoute, prêt à recevoir les données émises par un appareil externe.

7.3 Traitement et Transmission des Identifiants WiFi

Une fois connecté via BLE, la caractéristique configurée reçoit les données envoyées par l'utilisateur. La propriété d'écriture de la caractéristique déclenche l'exécution d'un callback spécifique :

- La valeur écrite est récupérée à l'aide de la fonction `getValue()`, puis convertie en chaîne de caractères.
- Il est présumé que les données reçues contiennent le SSID et le mot de passe séparés par une virgule. La fonction `indexOf(‘,’)` permet ainsi de diviser la chaîne en deux parties distinctes.
- Ces deux valeurs sont ensuite passées à la fonction de connexion WiFi `connectToWifi()`. Si la connexion est établie, un booléen `true` est retourné, sinon `false` est renvoyé.

Le serveur BLE reste en attente constante de nouvelles données, permettant ainsi à l'utilisateur de modifier les identifiants du réseau à tout moment si nécessaire.

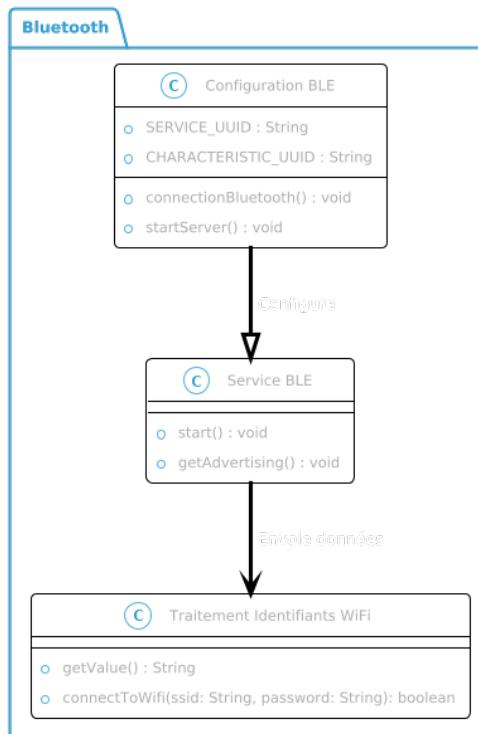


Figure 13 : Schémat de fonctionnement du BLE.

8 Conclusion

Ce projet a permis de démontrer la faisabilité et la robustesse d'une application IoT intégrée dans un TTGO T-Display. Les points forts du système comprennent :

- La mise en œuvre d'une API RESTful permettant le contrôle des LED et la consultation des mesures de capteurs.
- L'intégration d'un affichage en temps réel sur un écran TFT pour une meilleure visualisation locale.
- La flexibilité du réseau, avec une configuration initiale via WiFi et une solution de secours par BLE.
- L'implémentation des mises à jour OTA qui facilitent la maintenance et les évolutions futures.
- La persistance des paramètres grâce à la bibliothèque Preferences garantissant une stabilité sur le long terme.

Les perspectives d'amélioration incluent l'accroissement de la sécurité des échanges (via l'implémentation d'une authentification forte et de HTTPS), l'extension de la gamme de capteurs connectés et la création d'une interface utilisateur plus avancée. Ce projet s'inscrit donc comme une base solide pour des applications IoT évolutives, telles que la domotique ou la surveillance environnementale.

Fin du rapport.

Bibliographie

- ESP32 Technical Reference Manual.
- Documentation Arduino (<https://www.arduino.cc>).

- TFT_eSPI Library Documentation (https://github.com/Bodmer/TFT_eSPI).*ArduinoJsonDocumentation*(<https://arduinojson.org>).
- Arduino OTA Library Documentation (<https://github.com/esp8266/Arduino/tree/master/libraries/ArduinoOTA>).
- Documentation Bluetooth Low Energy (<https://www.bluetooth.com>).
- Références sur la formule de Steinhart-Hart.