


FACULTÉ DES SCIENCES

HAI912I - Développement mobile avancé, IoT et embarqué

Projet IOT - Rapport de projet

Rima ZOURANE
Toudherth MEKHNACHE

pour le 17 Janvier 2024

1 Introduction

Notre projet vise à exploiter les capacités d'un microcontrôleur ESP32, associé à des capteurs et des LED, pour créer un système polyvalent et interconnecté. L'objectif principal de cette initiative est de concevoir un environnement piloté par API fournissant des informations en temps réel à partir de capteurs de température et de lumière, tout en offrant un contrôle dynamique sur une configuration de LED RGB. Le système résultant met en avant non seulement l'intégration de composants matériels, mais exploite également les services web.

Ce rapport examine la conception et la réalisation des fonctionnalités de notre projet basé sur ESP32. Nous débuterons par une présentation détaillée du matériel utilisé, décrivant le montage de chaque composant, les résistances variables et fixes, ainsi que les LEDs. Nous aborderons les spécifications requises et détaillerons les mesures prises pour les satisfaire. Ensuite, nous explorerons les différentes fonctionnalités nécessaires pour notre API, mettant en évidence les choix et les considérations de conception. Enfin, nous partagerons les résultats des tests effectués. L'intégralité du projet peut être retrouvé ici

1.1 Logiciels utilisés

Concernant les spécifications nous avons utilisé *OpenAPI* pour les rédiger ainsi que *Swagger Editor* pour visualiser les requêtes, lien ici.

Nous avons utilisé *Arduino IDE* pour développer les fonctionnalités de notre porjet.

FIGURE 1.1 – Logo OpenAPI.

FIGURE 1.2 – Logo Arduino.

2 Mise en place de l'environnemt

Dans cette section nous allons aborder les différentes composantes matérielles qui ont été utilisées au sein de ce projet. La composante phare de notre projet étant l'*'ESP32 avec TTGO Display*, nous allons montrer comment nous avons monté les autres composantes en fonction de cette dernière. La figure 2.1 représente un ESP32 avec les détails des différents pin.

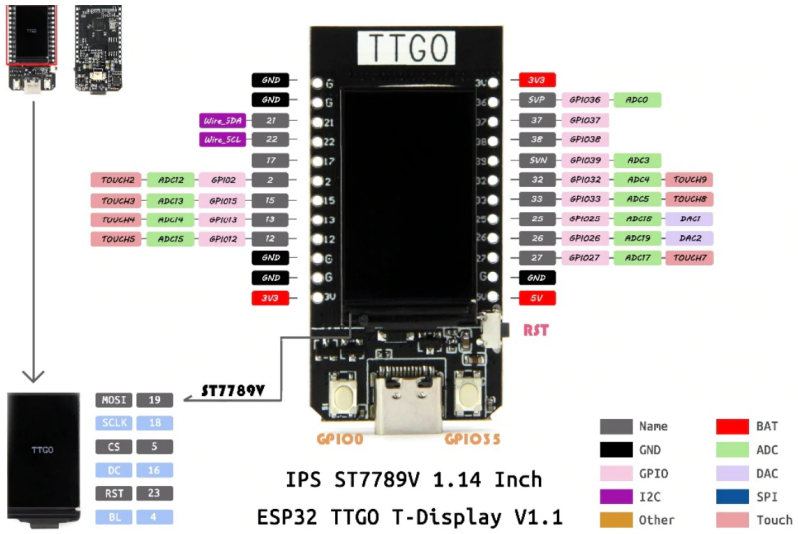


FIGURE 2.1 – Détails des pins sur l'ESP.

La configuration finale de notre matériel est comme suit :

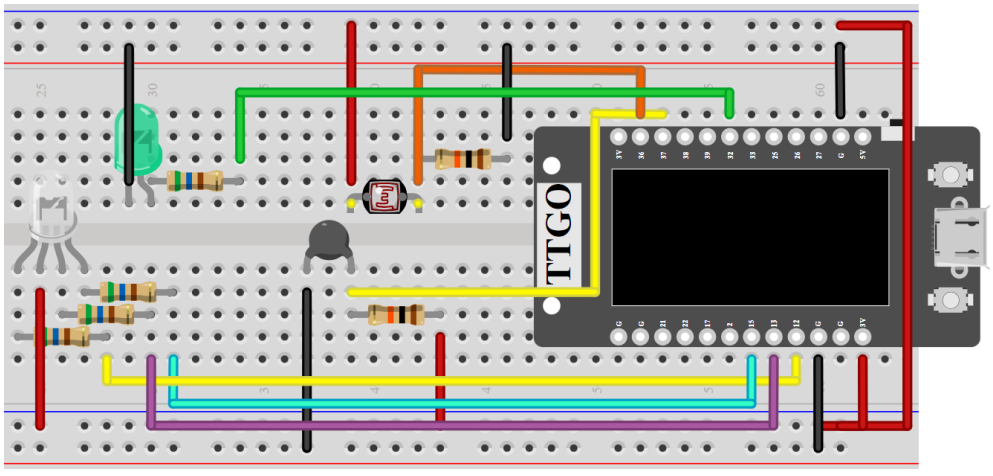


FIGURE 2.2 – Configuration finale.

2.1 Alimentation électrique

Nous avons décidé d'alimenter les différents circuits de notre installation avec une tension de $3.3v$. Étant donné que nous utilisons une platine d'essai (*breadboard*) et beaucoup de composantes, il a été nécessaire pour nous d'alimenter les deux côtés de la platine. La figure 2.3 montre les branchements que nous avons effectué pour alimenter les deux côtés de la platine via la sortie $3.3v$ de l'*ESP32*.

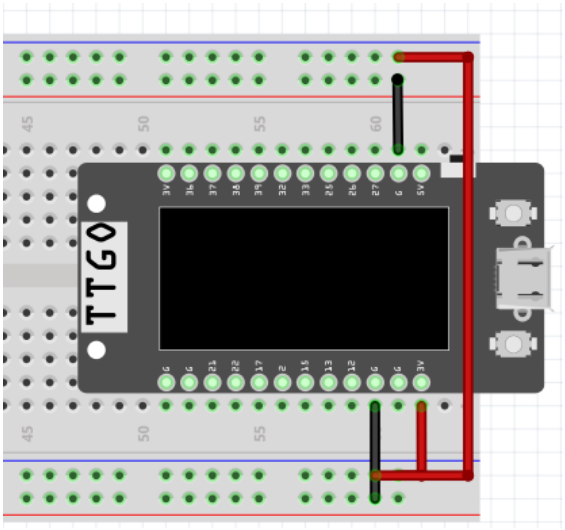


FIGURE 2.3 – Alimentation de la platine d'essai.

2.2 *Photocell*

La figure 2.4 représente l'illustration de la photo-résistance que nous utiliserons. En suivant la référence du kit qui nous a été donné nous avons pu avoir la référence exacte de la photo-résistance. Lorsque la face avec l'ondulation est exposée à plus de lumière, la résistance diminue. En pleine lumière, la résistance est d'environ $1K\Omega$, tandis qu'elle monte à environ $10K\Omega$ dans l'obscurité.

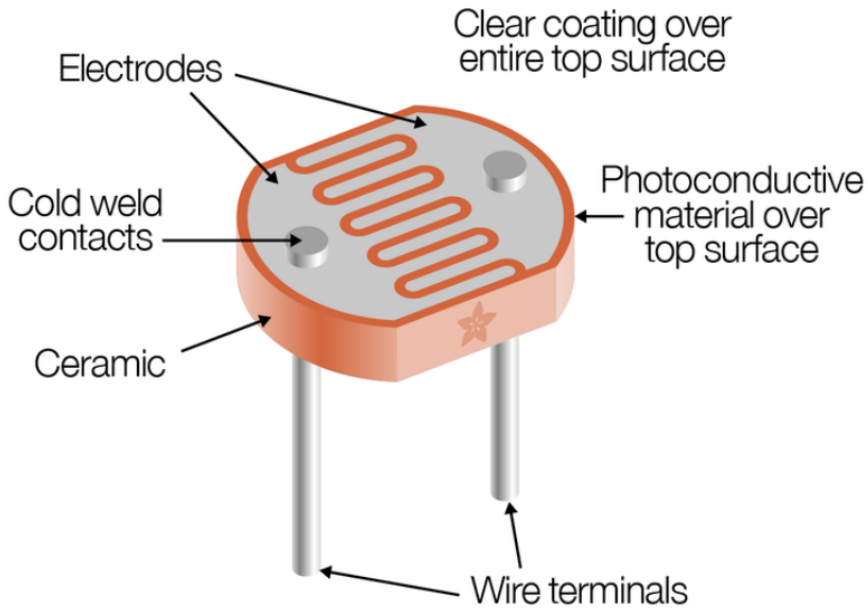


FIGURE 2.4 – Illustration de photorésistance (source).

Pour son utilisation nous l'avons combiné avec une résistance de $10k\Omega$ en formant un pont diviseur de tension, la figure 2.5 nous montre comment nous avons fait le montage de ce circuit. Ainsi, au niveau du pin 36 nou pourrons récupérer une valeur qui nous indiquera le niveau de lumière, plus cette valeur sera grande plus cela voudra dire qu'il y a de la lumière.

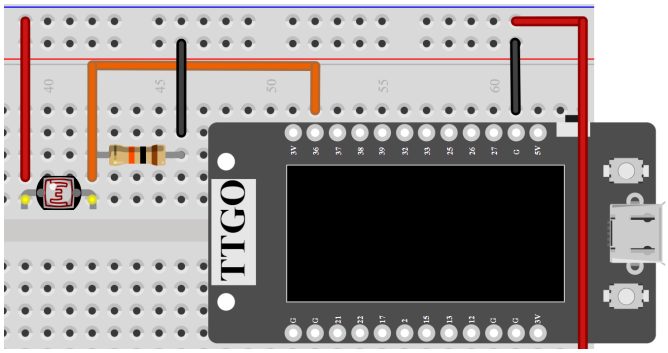


FIGURE 2.5 – Montage de la photo-résistance.

2.3 *Thermistor*

Les *thermistors* sont des résistances thermiques qui modifient leur résistance en fonction de la température. Il existe deux types de thermistors, NTC (coefficient de température négatif) et PTC (coefficient de température positif). Les NTC sont généralement utilisés pour la mesure de la température. Le modèle que nous utilisons est le thermistor 3950 NTC, nous pouvons le voir au niveau de la figure 2.6.

FIGURE 2.6 – 3950 NTC thermo-résistance (source)

De la même manière qu'avec la photo-résistance, nous avons monté la thermo-résistance avec une résistance de $10k\Omega$ pour créer un pont diviseur de tension. Comme nous pouvons le voir au niveau de la figure 2.7, le circuit est attaché au pin 37. Du pin 37, nous pourrons lire une valeur numérique qui, après quelques transformations, indiquera la température.

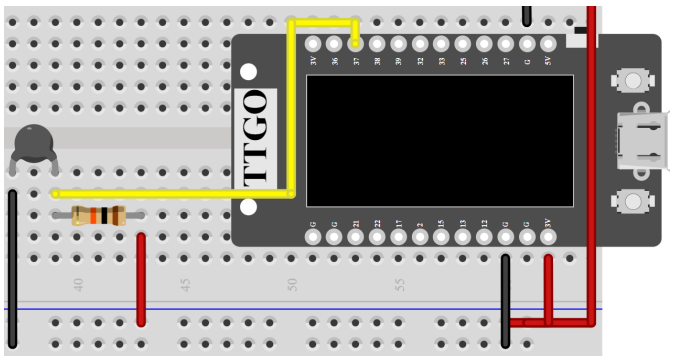


FIGURE 2.7 – Montage de la thérmo-résistance.

2.4 LEDS

Nous avons utilisé deux LEDs : une LED verte simple ainsi que la LED RGB fournies dans le kit. Afin d'éviter le risque de surchauffe nous avons mis des résistances de 560Ω en série avec chacune des LED. La LED RGB étant trois LEDs combinés avec l'anode commune aura donc besoin de trois résistances. Le schéma détaillé est fourni au niveau de la figure 2.8.

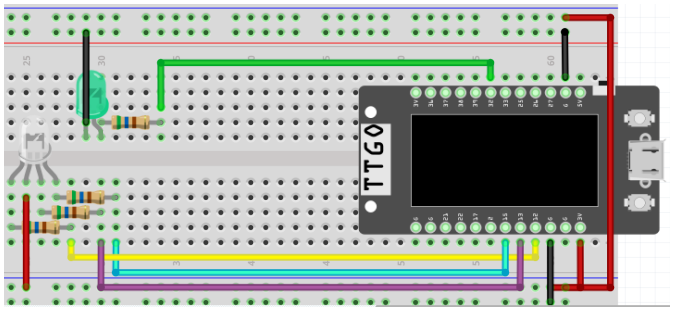


FIGURE 2.8 – Montage des LEDS en série avec des résistances.

2.5 Problèmes rencontrés

À cause de l'utilisation du pin GPIO2 (première version de notre montage), nous avons eu du mal à charger le programme sur le micro-controlleur. La figure 2.9 montre le message d'erreur qui s'affiche lorsque nous essayons de charger un programme sur l'ESP32. D'abord nous avons trouvé une solution qui proposait de mettre l'ESP32 en mode "upload" de manière manuelle et ce en appuyant sur le bouton "reset" en même temps que le chargement du code sur la carte. Cette solution ne marchait pas à toutes les reprises et n'était pas optimal du tout. Puis nous avons trouvé un forum qui évoquait ce problème de mode de chargement lorsque le pin GPIO2 est utilisé nous avons donc décidé de ne plus l'utiliser et celà nous a permis de régler le problème.

```
A fatal error occurred: Failed to connect to ESP32: Wrong boot mode detected (0xb)! The chip needs to be in download mode.
For troubleshooting steps visit: https://docs.espressif.com/projects/esptool/en/latest/troubleshooting.html
Failed uploading: uploading error: exit status 2
```

FIGURE 2.9 – Message d'erreur lors du chargement d'un programme

2.6 Fonctionnalités logicielles

2.6.1 Band passante Wifi

Ayant besoin d'une connexion Wifi pour établir quelconque communication entre le micro-contrôleur et une entité tierce, nous avons testé avec les exemples de code fournis au niveau de l'environnement de développement Arduino IDE. Nous avons activé le point d'accès sans fil mobile de l'ordinateur pour pouvoir connecter l'ESP à internet. Nous avons remarqué une chose très importante en utilisant l'exemple WifiScan pour vérifier si le micro-contrôleur détecte notre point d'accès. Lorsque le paramètre bande passante réseau du point d'accès est défini à 5GHz ou disponible (voir figure 2.10) l'ESP32 ne détectait pas le réseau contrairement à lorsque ce paramètre est défini à 2,4 GHz. La figure

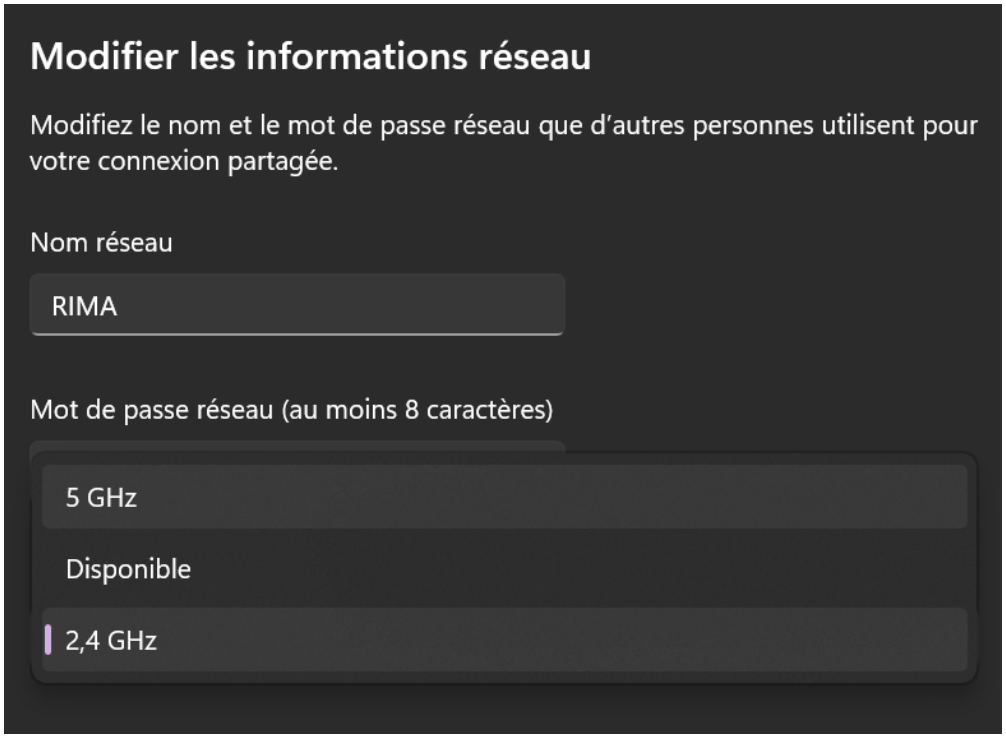


FIGURE 2.10 – Paramètre bande passante réseau du point d'accès mobile.

```
Scan start
Scan done
21 networks found
Nr | SSID                             | RSSI | CH | Encryption
 1 | RIMA                             |  -31 |  6 | WPA2
```

FIGURE 2.11 – Détection de notre Wifi par l'ESP32

3 Fonctionnalités de l'API

Comme cela a été demandé notre application vise à accomplir les tâches suivantes :
— Extraire les informations associées à un ou plusieurs capteurs.
— Gérer l'allumage, l'extinction et le changement d'état d'une LED connectée.
— Établir un seuil pour réguler l'allumage/extinction de la LED en fonction des valeurs captées.
— Répertorier la configuration (capteurs, LEDs, seuils).
— Utiliser l'écran pour afficher des informations.
Nous avons respecté les directives dans l'ensemble, avec quelques ajustements mineurs.

3.1 Conception et Implémentation

Nous sommes passés par différentes version de l'API avant de choisir cette dernière. Toutes les spécifications sont au niveau du fichier `iotapi.yaml` dans le dossier `specification`. La figure 3.1 nous montre un résumé des fonctionnalités de notre api.

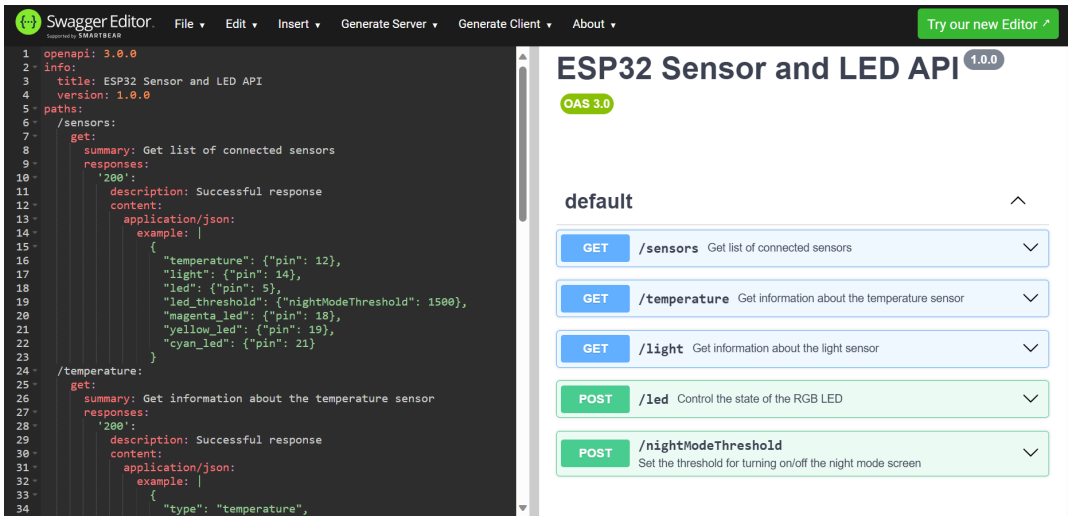


FIGURE 3.1 – *Swagger editor* pour les spécifications de notre api.

L'API web comprend plusieurs routes :
— `/sensors` : de type GET renvoie une liste des capteurs connectés ainsi que des informations les concernant.
— `/temperature` : de type GET renvoie la valeur actuelle du thermistor.
— `/light` : de type GET renvoie la valeur actuelle du capteur de lumière.
— `/led` : de type POST contrôle l'état de la LED RGB.
— `/nightModeThreshold` : de type POST définit le seuil pour activer/désactiver le mode nuit de l'écran.

3.2 Fonctionnalités et tests

Nous avons essayé d'organiser le code comme suit :
— Bibliothèques et variables.
— Configuration : fonctions `setup` et `loop`.
— Gestion de l'API.
— Gestion de la LED RGB.
— Gestion de l'écran TTF.
— Gestion des capteurs.
Nous allons commencer par les fonctions élémentaires c'est-à-dire celles qui n'utilisent pas d'autres fonctions.

3.2.1 Configuration

La fonction `setup()` est une composante essentielle des programmes Arduino, s'exécutant une seule fois au démarrage du microcontrôleur. Dans le code source, elle accomplit les tâches suivantes :
— **Initialisation de la communication série :** `Serial.begin(115200);` initialise la communication série à un débit de 115200 bits par seconde, principalement utilisé à des fins de débogage.
— **Initialisation de l'écran TFT :** `tft.init();` initialise l'écran TFT. `tft.setRotation(1);` définit la rotation de l'écran, et `tft.fillScreen(TFT_BLACK);` remplit l'écran de noir.
— **Configuration des broches LED :** Les appels à `pinMode()` configurent les broches dédiées au contrôle des LEDs en tant que sorties.
— **Création d'une tâche pour la transition de la lumière RGB :** `xTaskCreate()` est une fonction spécifique à l'ESP32 qui crée une tâche parallèle exécutant la fonction `setupRGB()`. Cette fonction sera plus détaillée en bas.
— **Connexion au WiFi :** `connectToWiFi();` est probablement une fonction assure la connexion à un réseau WiFi. Elle sera vu en détails juste après.
— **Configuration des routes de l'API :** `setup_routing();` est une fonction qui configure les routes pour le serveur web.
— **Démarrage du serveur web :** `server.begin();` lance le serveur web.

La fonction `loop()` est une autre partie cruciale des programmes Arduino, s'exécutant en boucle après l'achèvement de `setup()`. Dans le code source, elle réalise les actions suivantes :
— **Effacement de l'écran :** `tft.fillScreen(TFT_BLACK);` remplit l'écran de noir.
— **Lecture de la lumière et de la température :** `readLight();` et `readTemperature();` lisent les valeurs du capteur de lumière et du thermistor, respectivement.
— **Impression de la valeur du photocell :** `Serial.println(analogRead(photoCellPin));` lit la valeur analogique de la broche du photocell et l'imprime sur la console série.
— **Délai :** `delay(500);` suspend l'exécution du programme pendant 500 millisecondes.

3.2.2 Gestion de la LED RGB et la LED verte

La gestion de la LED RGB dans le code est assurée par deux fonctions principales : `setupRGB` et *setRGBColor* et les fonctions auxiliaires associées. La vidéo montrant cette fonctionnalité peut être retrouvé dans le dossier video

Initialisation et Transition des Couleurs

La fonction `setupRGB`, créée en tant que tâche à l'aide de `xTaskCreate()`, gère la transition des couleurs de la LED RGB. Elle initialise les valeurs des composantes rouge (`r`), bleue (`b`), et verte (`g`) et effectue une transition entre ces couleurs sur une période de 3 secondes. Cette transition est réalisée en ajustant les valeurs de chaque composante de manière à créer une variation harmonieuse des couleurs. xTaskCreate nous permet de lancer des tâches avec délais sans pour autant qu'elles soient bloquantes pour le reste des tâches à la place de `delay()` on met `vTaskDelay();`.

Réglage de la Couleur de la LED RGB

La fonction `setRGBColor` est responsable du réglage précis de la couleur de la LED RGB. Elle prend en entrée les valeurs des composantes magenta, cyan et jaune, et utilise `analogWrite()` pour ajuster les intensités lumineuses de chaque composante. Cette fonction garantit que les valeurs sont dans la plage valide.

Ajustement de la Luminosité de la LED verte

La fonction `adjustLEDBrightness` ajuste la luminosité de la LED en fonction de la valeur du capteur de lumière. Elle utilise la fonction `map()` pour faire correspondre la valeur du capteur à la plage de luminosité de la LED (de 0 à 255). Ainsi, la LED brille moins lorsque l'environnement est plus lumineux et vice versa.

Ces fonctions permettent un contrôle précis de la couleur et de la luminosité de la LED RGB en fonction des conditions ambiantes et contribuent à la polyvalence du système. La vidéo "ledBrightness" qui se trouve ici nous montre cette fonctionnalité.

3.2.3 Gestion de l'écran TTF

La gestion de l'écran TFT dans le code est effectuée à travers plusieurs fonctions dédiées à l'affichage, à la gestion des messages de succès ou d'erreur, ainsi qu'au contrôle du mode nuit. Ces fonctions sont décrites ci-dessous :

Affichage des valeurs des capteurs sur l'écran

Deux fonctions, `displaySensorValueOnScreen` et `displayTemperatureOnScreen`, sont responsables de l'affichage des valeurs du photocell et de la température, respectivement. Elles ajustent la taille du texte, la couleur et la position du curseur sur l'écran TFT pour présenter ces informations de manière lisible.

Messages de succès et d'erreur

- `showSuccessMessage()` : Cette fonction affiche un message de succès sur l'écran, indiquant que la requête a été acceptée. Elle utilise la couleur verte pendant 2 secondes (modifiable) et efface ensuite l'écran.
- `showErrorMessage(const char* errorMessage)` : En cas d'erreur, cette fonction affiche un message d'erreur sur un fond rouge. Elle affiche le message d'erreur fourni en argument pendant 2 secondes (modifiable) et efface ensuite l'écran.

Contrôle du mode nuit

Deux fonctions, `enableNightMode()` et `disableNightMode()`, gèrent le passage en mode nuit et le retour au mode jour, respectivement. Elles inversent les couleurs sur l'écran TFT pour s'adapter aux conditions lumineuses.

Ces fonctions contribuent à rendre l'interface utilisateur plus interactive et informatif en utilisant l'écran TFT pour afficher des informations importantes et des retours visuels.

3.2.4 Gestion des capteurs

Le code comprend des fonctions dédiées à la lecture des capteurs de température et de lumière, ainsi qu'au réglage de la luminosité de la LED en fonction de la lecture du capteur de lumière.

Fonction `readRawTempData()`

La fonction `readRawTempData()` lit la valeur brute du capteur de température (thermistor). Elle utilise la fonction `analogRead()` pour obtenir la valeur analogique du capteur, puis consulte une table de conversion `ADC_LUT` pour obtenir la valeur corrigée du capteur.

Fonction `readTemperature()`

La fonction `readTemperature()` utilise les valeurs du capteur de température obtenues précédemment pour calculer la température en degrés Celsius (`Tc`) et Fahrenheit (`Tf`). Elle ajuste ensuite la luminosité de l'écran TTGO en fonction de la température mesurée et affiche cette valeur sur l'écran, si la température est supérieure à zéro degrés Celsius.

Fonction `readLight()`

La fonction `readLight()` lit la valeur du capteur de lumière à l'aide de `analogRead()`. Elle ajuste la luminosité de la LED verte en fonction de la lecture du capteur de lumière à l'aide de la fonction `adjustLEDBrightness()`. De plus, elle affiche la valeur lue du capteur sur l'écran et active le mode nuit si l'intensité lumineuse est inférieure au seuil spécifié (`nightModeThreshold`).

Ces fonctions permettent la surveillance en temps réel de l'environnement, fournissant des informations sur la température et la luminosité, tout en ajustant dynamiquement la LED et l'écran TTGO en conséquence.

3.2.5 Gestion de l'API

Le programme Arduino intègre un serveur web qui permet la communication avec l'ESP32. Les routes de l'API et les fonctions associées sont définies dans la fonction `setup_routing()`.

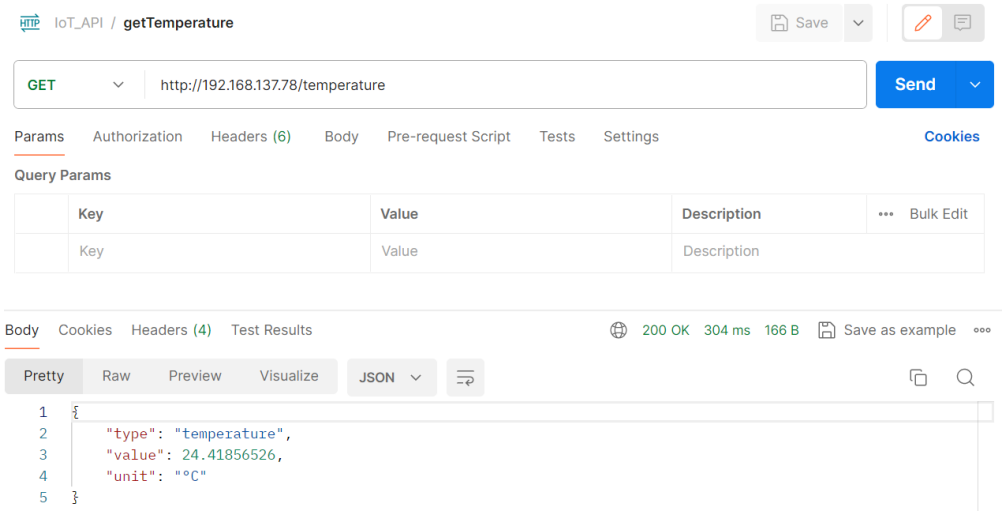
Fonction `setup_routing()`

La fonction `setup_routing()` configure les différentes routes de l'API web. Chaque route est associée à une fonction spécifique qui sera appelée lorsqu'une requête est effectuée sur cette route. Les principales routes et leurs fonctions associées sont les suivantes :

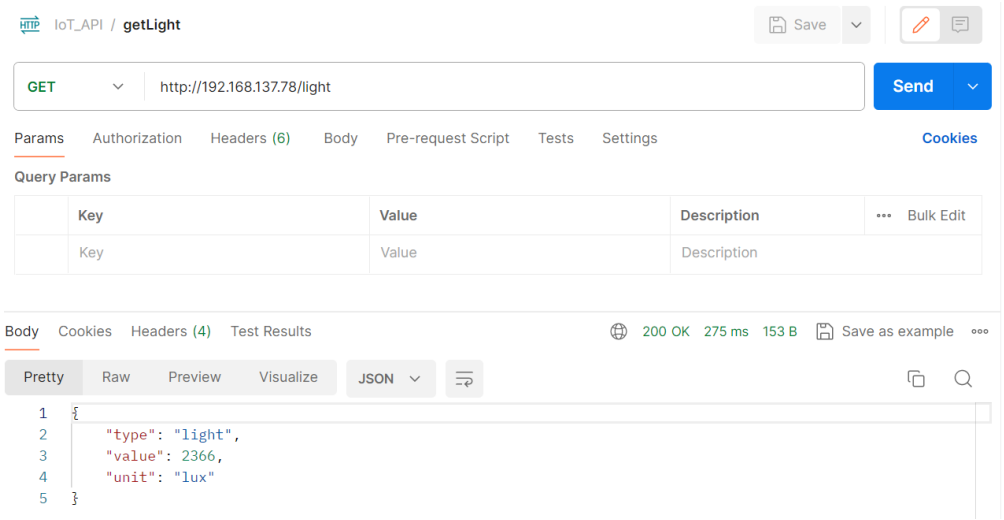
— `/sensors` (GET) : Renvoie une liste des capteurs connectés avec leurs informations respectives. La figure 3.2 nous montre le résultat de cette requête

FIGURE 3.2 – Réponse de la requête `/sensors`

— `/temperature` (GET) : Renvoie la valeur actuelle du capteur de température en degrés Celsius. Nous avons testé cette requêtes sur *Postman* et la figure suivante nous le montre.



— `/light` (GET) : Renvoie la valeur actuelle du capteur de lumière.



— `/led` (POST) : Contrôle l'état de la LED RGB en fonction des paramètres reçus dans le corps de la requête.
— `/nightModeThreshold` (POST) : Définit le seuil pour activer/désactiver le mode nuit de l'écran en fonction de la valeur spécifiée dans le corps de la requête.

Fonctions de Gestion des Routes

Les fonctions associées aux routes effectuent les opérations suivantes :
— `getSensors()` : Récupère les informations des capteurs de température, de lumière et de la LED, puis les renvoie sous forme de réponse JSON.
— `getTemperature()` : Renvoie la valeur actuelle du capteur de température en format JSON.
— `getLight()` : Renvoie la valeur actuelle du capteur de lumière en format JSON.
— `postNightModeThreshold()` : Définit le seuil pour le mode nuit en fonction de la valeur spécifiée dans le corps de la requête. La figure 3.3 nous montre que la requête a bien été prise en compte, nous pouvons vérifier celà en appelant la requête `/sensors`,

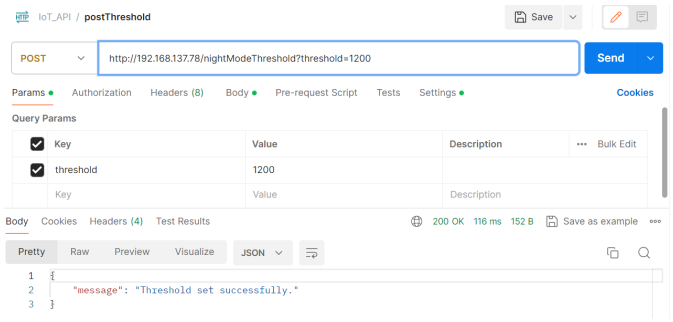


FIGURE 3.3 – Résultat de la requête de mise à jour du seuil `nightModeThreshold`.

— `controlLED()` : Contrôle l'état de la LED RGB en fonction des paramètres spécifiés dans le corps de la requête.

Connexion au WiFi

La fonction `connectToWiFi()` permet à l'ESP32 de se connecter à un réseau WiFi spécifié (`ssid` et `password`). Elle utilise la bibliothèque `WiFi` pour établir la connexion et affiche l'adresse IP attribuée à l'ESP32. Si l'ESP ne se connecte à partir d'un certain moment il redémarre.

Ces fonctionnalités permettent la mise en place d'une interface web permettant de surveiller et contrôler les capteurs et la LED RGB à distance.