Documentation de mon mini-projet

Élaboré par MANAI RANIM

Résumé du concept du projet :

Ce projet vise à créer un système IoT qui utilise un microcontrôleur ESP32, un capteur de température et d'humidité DHT11, un serveur GraphQL et une LED. Le microcontrôleur ESP32 récupère les données du capteur et les envoie à un serveur GraphQL.Ce serveur assure l'allumage du led lorsque la condition est vraie. De là, un client web peut accéder aux données du capteur et contrôler l'état de la LED à distance via des requêtes GraphQL.

Connection :

1. Composant matériels :

*/ESP32

*/Capteur DH11(sesor pin connecté à une broche numérique de ESP32)

*/LED(anode connectée à une broche numérique de ESP32 et cathode à la masse GND)

2. **Cofiguration:**

- Configuration du serveur graphQl:

Le serveur GraphQL est distant .Il reçoit les données du capteur et les requêtes de contrôle de la LED.

Technologies utilisées:

*Node.js

*Express.js

*GRAPHQL

*graphql-tools

Cors(cross-Origin Resource sharing

- Cofiguration de ESP32:

L'ESP32 est responsable de la collecte des données du capteur et de l'envoi au serveur GraphQL. Il contrôle également l'état de la LED en fonction des requêtes reçues du serveur GraphQL.

Langage de programmation :

Arduino(c++)

Bibliothèques utilisées :

*WIFI.h:pour la connection Wi-Fi

*HTTPClient.h: pour les requêtes de http

*DHT.h:pour l'intéraction avec le capteur DH11

- Configuration Interface utilisateur Web:

Une interface web simple permet à l'utilisateur de visualiser les données du capteur (température et humidité) et de contrôler l'état de la LED à distance.

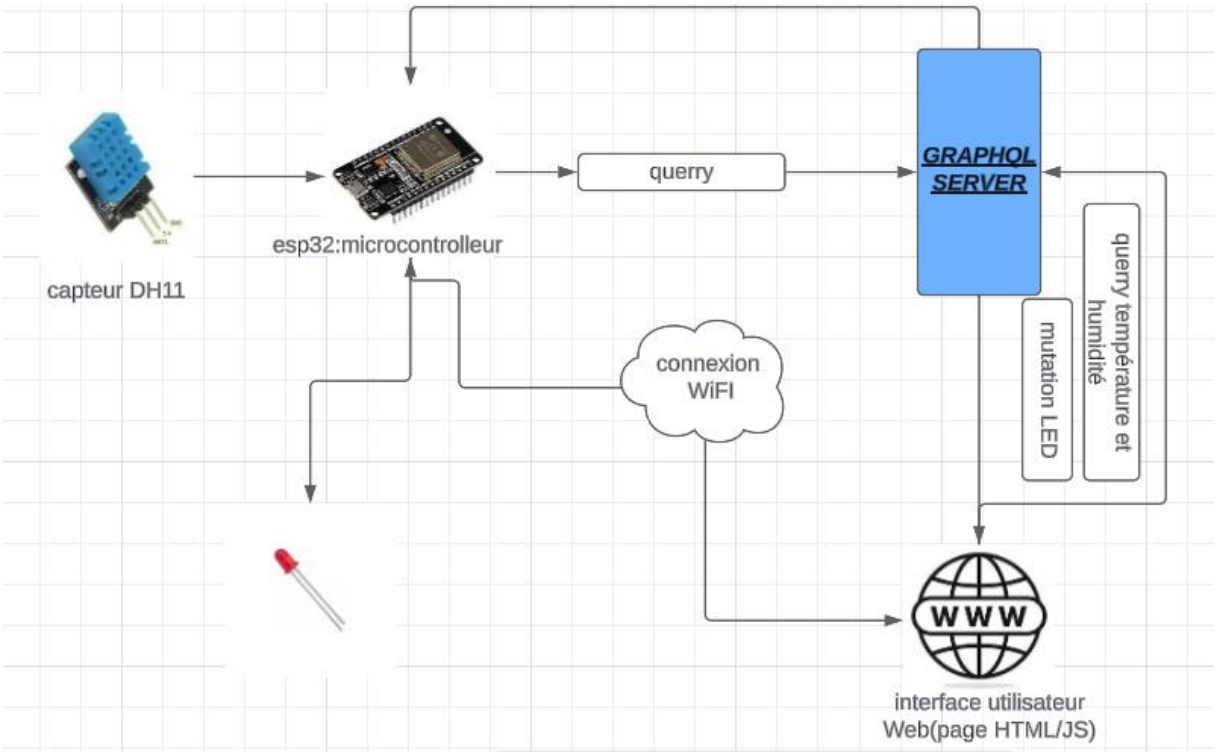
Technologies utilisées :

*/HTML

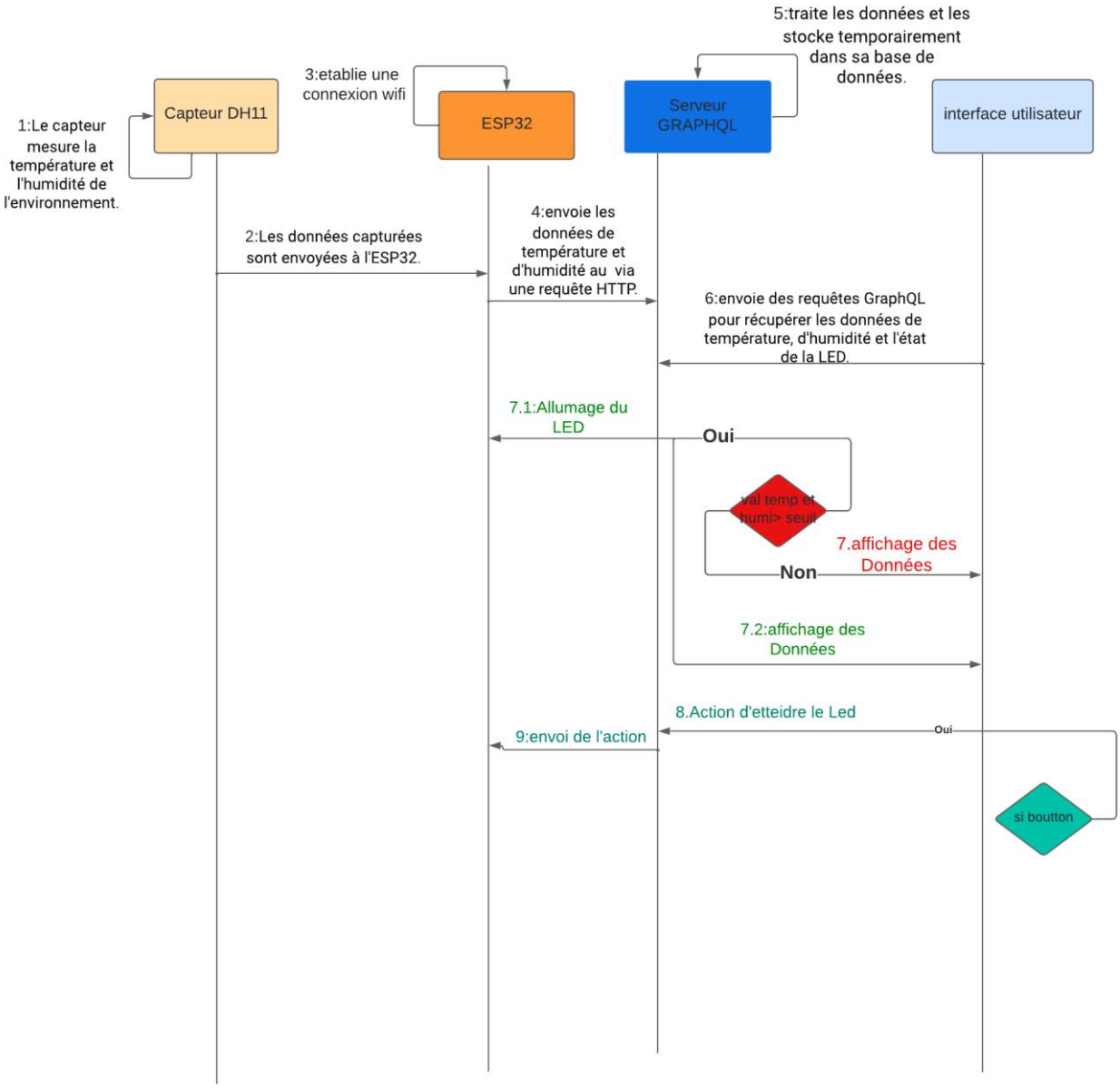
*/CSS

*/JavaScript

Schéma synoptique du Projet :



3. Fonctionnement Du Projet :



Résumé de fonctionnement:

*/Initialisation du WiFi et du Capteur :

*/L'ESP32 démarre et initialise la connexion WiFi.

*/Le capteur DHT11 est configuré pour la lecture des données de température et d'humidité.

*/Lecture des Données du Capteur :

*/L'ESP32 lit les valeurs de température et d'humidité du capteur.

*/Envoi des Données au Serveur GraphQL :

L'ESP32 envoie les données de température et d'humidité au serveur GraphQL via une requête HTTP POST.

(Requête HTTP POST vers l'URL du serveur GraphQL.)

Utilisation de l'outil HTTPClient pour envoyer la requête.

*/Réception des Données dans le Serveur GraphQL :

Le serveur GraphQL reçoit les données de température et d'humidité.

Les données sont traitées par les mutations setTemperature et setHumidity du schéma GraphQL.

*/Contrôle de l'État de la LED :

Le serveur GraphQL contrôle l'état de la LED en fonction des valeurs de température et d'humidité reçues.

Si la température et l'humidité atteignent les valeurs d'alerte, la LED est allumée en utilisant la mutation setLEDState.

La condition d'allumage de la LED est vérifiée dans la mutation setLEDStateAUTO.

*/Récupération des Données sur l'Interface Utilisateur (Page Web) :

L'interface utilisateur (page web) envoie des requêtes GraphQL au serveur pour récupérer les données de température, d'humidité et l'état de la LED.

(Utilisation de requêtes GraphQL de type query.)

*/Affichage des Données sur l'Interface Utilisateur (Page Web) :

Les données de température et d'humidité sont affichées sur l'interface utilisateur.

*/L'état de la LED est également affiché et contrôlé par un bouton.

(Utilisation de JavaScript pour effectuer les requêtes GraphQL depuis la page web.)

(Utilisation de l'outil fetch pour envoyer les requêtes GraphQL.)

*/Contrôle de l'État de la LED depuis l'Interface Utilisateur :

L'utilisateur peut contrôler l'état de la LED en appuyant sur un bouton sur l'interface utilisateur.

Lorsque le bouton est actionné, une requête GraphQL de type mutation est envoyée au serveur pour changer l'état de la LED. (Utilisation de requêtes GraphQL de type mutation.)

Ces étapes décrivent le processus complet du projet, de la capture des données par le capteur jusqu'à leur affichage et leur contrôle sur l'interface utilisateur.

4. Les étapes des codes :

- Code arduino ide:

Mon code Arduino permet de lire les données d'un capteur DHT11 qui mesure l'humidité et la température, puis envoyer ces données à un serveur GraphQL via une connexion Wi-Fi.
Explication détaillée:

1/ inclusion des bibliothèques nécessaires :

Les bibliothèques sont nécessaires pour la gestion du Wi-Fi, des requêtes HTTP et du capteur DHT.

-->Ces données sont envoyées sous forme de requêtes HTTP POST contenant des données au format JSON, avec les valeurs lues par le capteur

2/ définition des paramètres de connexion Wi-Fi et de l'URL du serveur GraphQL :

3/ initialisation du capteur DHT11:

4/ configuration de la connexion Wi-Fi dans la fonction “setup()”

5/ Lecture périodique des données du capteur dans la boucle “loop()”

6/ Envoi des données au serveur GraphQL :

Les données d'humidité et de température sont envoyées au serveur via des requêtes POST GraphQL.

7/gestion des réponse du serveur:

Le code vérifie si les requêtes ont été envoyées avec succès et affiche un message approprié.

- Code GraphQL:

1/importation des modules nécessaire:

ces modules sont nécessaires pour créer un serveur HTTP avec Express, gérer les requêtes GraphQL et mettre en place le schéma GraphQL.

2/définition du schéma GraphQL:

schéma définit les types de requêtes (Query) et de mutations (Mutation) disponibles dans l'API GraphQL.

3/implémentation des résolveurs:

Les résolveurs sont des fonctions qui déterminent comment répondre à chaque requête ou mutation GraphQL. Ils sont associés aux types définis dans le schéma.

4/création du schéma executable:

schéma est créé en utilisant les types définis et les résolveurs associés.

5/configuration du serveur express:

express est initialisé et le middleware CORS est utilisé pour autoriser les requêtes de différentes origines.

6/définition du point d'accés GraphQL:

Cette configuration permet à Express de gérer les requêtes GraphQL à l'URL/graphql, en utilisant le schéma défini précédemment. L'interface GraphiQL est activée pour faciliter le test des requêtes.

7/configuration des point d'accés pour la page web:

lorsqu'un utilisateur accède à la racine de l'URL, le serveur renvoie la page web spécifiée

8/lancement du serveur express:

Pour servir la page web.

- Code page web:

le code HTML et JavaScript crée une page web qui interagit avec le serveur GraphQL pour afficher les données de température et d'humidité provenant d'un capteur connecté à un ESP32 en temps réel, ainsi que pour contrôler une LED à distance..

1/structure html

2/contenu du page :

Cette partie crée les éléments HTML pour afficher les données de température et d'humidité ainsi qu'un bouton pour éteindre une LED. Les données initiales sont définies comme "Loading..." jusqu'à ce qu'elles soient récupérées du serveur.

3/script JavaScript:

Le script contient la logique JavaScript pour interagir avec le serveur GraphQL et mettre à jour les données affichées sur la page.

4/fonction JavaScript:

- ‘fetchTemperature()’:fonction qui envoie une requête graphql pour obtenir la température du capteur
- ‘fetchHumidity()’:même chose mais pour humidity
- ‘turnOffLED()’:envoie une requête graphQl pour éteindre le led

5/évènement et intervalle de rafraîchissement :

Les données de température et d'humidité sont récupérées initialement lors du chargement de la page, puis mises à jour toutes les 5 secondes à l'aide de ‘setInterval()’L'événement de clic sur le bouton "Turn Off LED" est géré pour éteindre la LED lorsque le bouton est cliqué.