

Station météo IOT



Table des matières

Introduction générale.....	4
Chapitre I : Programmation de l'stm32 :.....	5
1- Introduction :.....	6
2- Matériel nécessaire.....	6
3- Explication de code.....	6
3.1- Initialisation et configuration HAL.....	6
3.2- Périphériques initialisés via CubeMX	6
3.3- Bibliothèques et handles	6
3.4- Fonctions utilitaires.....	6
3.5- Boucle principale (while(1))	7
4- Connexion hardware	7
5- Configuration CubeMX	7
6- Intégration des bibliothèques	8
7- Structure du code principal	8
8- Compilation et flash.....	9
9- Tests et validation	9
10- Conclusion	9
Chapitre II : Programmation de l'esp32 :.....	10
1- Introduction.....	11
2- Aperçu générale :	11
3- Matériel nécessaire.....	11
4- Schéma de connexion	12
5- Configuration & bibliothèques	12
6- conclusion	Erreur ! Signet non défini.
Chapitre III : Guide de Reproduction du Projet de Station Météo Firebase	15

1- Introduction	16
2- Vue d'Ensemble du Projet	16
3- Technologies Utilisées	16
4- Structure du Projet	16
5- Étapes de Reproduction	17
5.1- Configuration de Firebase	17
5.2- Configuration des Fichiers du Projet	18
5.3- Configuration d>EmailJS	19
5.4- Installation et Déploiement	21
5.5- Interaction avec la Base de Données	22
6- Fonctionnalités Clés du Code	23
7- Conclusion	23
Conclusion générale	24

Liste des tableaux

Tableau 1 : Schéma de connexion des capteur avec l'stm32..... 7

Tableau 2 : Schéma de connexion des capteurs avec l'esp32..... 12

Introduction générale

Ce document représente un guide qui permet de reproduire ce projet. Il inclut les différentes étapes de création de cette station météo qui sont principalement trois étapes (programmation de l'stm32, programmation de l'esp32 et l'interface web).

L'stm32 joue le rôle de l'mcu principal il occupe la mesure et l'analyse des données provenant des capteurs, l'affichage sur un LCD, et l'envoie des données à l'esp32 qui va les envoyer vers la base de données.

La dernière étape c'est l'interface web qui va afficher les informations reçues de puis la base de données. Le point fore de ce projet c'est qui les mesure des capteurs sont accessible via une Dashboard temps réel De n'importe où dans le monde.

Chapitre I : Programmation de l'stm32 :

1- Introduction :

Ce chapitre détaille la programmation du microcontrôleur STM32, qui assure l'acquisition des données des capteurs, l'affichage local sur écran LCD, ainsi que la communication avec l'ESP32. Il constitue le cœur embarqué du système.

2- Matériel nécessaire

- **Carte STM32** (ex. STM32F429, STM32F407, etc.)
- **Module BMP280 / BME280 (I²C)**
- **Écran LCD 16×2 I²C** (adresse 0x27)
- **Capteurs gaz et flamme** (sorties analogiques)
- **Module CO₂** avec sortie UART TTL (ex. MH-Z19)
- Fils de connexion, breadboard, alimentation 3.3 V.

[Code utiliser avec l'stm32](#)

3- Explication de code

3.1- Initialisation et configuration HAL

- HAL_Init() et SystemClock_Config() pour configurer l'horloge interne et le système .

3.2- Périphériques initialisés via CubeMX

- GPIO (LEDs, EXTI sur PA0), ADC1 (canaux 1 et 4), USART2 (UART2), I2C1 .

3.3- Bibliothèques et handles

- bmp280.h pour capteur de pression/température/humidité via I²C.
- i2c_lcd.h pour affichage sur écran LCD I²C.
- __io_putchar redirige printf vers USART2 pour debug.

3.4- Fonctions utilitaires

- adc_read_channel() lit un canal ADC sans scan.
- Callbacks :

- HAL_UART_RxCpltCallback : réception asynchrone de la mesure CO₂ (PPM) sur UART à partir de l'esp32.
- HAL_GPIO_EXTI_Callback : incrémente lcd_select à chaque appui sur PA0.

3.5- Boucle principale (while(1))

- Lecture des capteurs gaz (ADC4) et flamme (ADC1).
- Lecture BMP280 (température, pression, humidité).
- Allumage/arrêt des LEDs D13/D15 selon seuils.
- Affichage cyclique sur LCD (press./temp., hum./CO₂, gaz/flamme).
- Envoi JSON des mesures sur UART via printf vers l'esp32

4- Connexion hardware

Le tableau suivant représente les connexions des différentes composants avec l'stm32

Périphérique	STM32 Pin	Remarque
BMP280 SDA	I2C1_SDA (PB7)	
BMP280 SCL	I2C1_SCL (PB6)	
LCD I²C SDA	PB7	même bus I ² C1
LCD I²C SCL	PB6	
Capteur gaz (ADC)	PA4 (ADC1_IN4)	ADC_CHANNEL_4
Capteur flamme (ADC)	PA1 (ADC1_IN1)	ADC_CHANNEL_1
Bouton poussoir	PA0	EXTI0 (interruption)
LED Gaz (verte)	PD15	active si gaz
LED Flamme (bleue)	PD13	active si flamme
LED Erreur (rouge)	PD14	allumée en Error_Handler()
UART_RX (CO₂)	PA3 (USART2_RX)	réception asynchrone
UART_TX	PA2 (USART2_TX)	debug printf

Tableau 1 : Schéma de connexion des capteurs avec l'stm32

5- Configuration CubeMX

1. **Sélection du microcontrôleur** (ex. STM32F429ZI).
2. **Clock Configuration**
 - Choisir HSE puis résolve automatique
3. **GPIO**
 - PD13, PD14, PD15 en GPIO_Output (LEDs).
 - PA0 en GPIO_EXTI_Falling (bouton).
4. **ADC1**
 - Mode normal, pas de scan.
 - Canaux 1 et 4 configurés en “Rank 1” manuellement (on reconfigure à la volée).
5. **I²C1**
 - Mode I²C, Fast mode, 100 kHz.
6. **USART2**
 - 115200 baud, 8N1, RX en IT (UART_IT_RXNE).
7. **NVIC**
 - Activation de l'interruption EXTI0_IRQHandler (priorité 3).
 - Activation de l'interruption USART2 (priorité 5).

Après configuration, générer le code pour STM32CubeIDE.

6- Intégration des bibliothèques

1. **Copier** les fichiers bmp280.c/h et i2c_lcd.c/h dans le dossier Core/Src et Core/Inc.
2. **Inclure** leurs déclarations dans main.c : #include "bmp280.h" et #include "i2c_lcd.h"

7- Structure du code principal

- **init_lcds()** : initialise le LCD I²C avec l'adresse 0x27.
- **adc_read_channel(channel)** : sélectionne et lit un canal ADC.
- **Callbacks** pour UART et EXTI.
- **Boucle infinie** :
 1. Lire ADC gaz et flamme.
 2. Lire capteur BMP280 (temp., press., humid.).

3. Mettre à jour LEDs.
4. Afficher sur LCD selon lcd_select.
5. Envoyer JSON sur UART.
6. HAL_Delay(1000) et lcd_clear().

8- Compilation et flash

1. **Ouvrir** le projet généré dans STM32CubeIDE.
2. **Vérifier** que les fichiers bmp280.c et i2c_lcd.c sont inclus.
3. **Compiler** (Build → Build Project).
4. **Connecter** le ST-Link / JTAG.
5. **Flasher** (Run → Debug ou Run → STM32CubeProgrammer).

9- Tests et validation

1. **Bouton poussoir** : chaque pression fait défiler l'affichage LCD.
2. **Capteurs gaz/flamme** : vérifier changement d'état des LEDs PD15/PD13.
3. **BMP280** : afficher valeurs plausibles (temp. \approx 20–30 °C, pression \approx 1000 hPa).
4. **CO₂** : envoyer via esp32 des trames ASCII (ex. “450\n”) pour voir le ppm sur LCD et JSON.
5. **Anomalies** :
 - Si échec BMP280 → LED rouge PD14 allumée (dans Error_Handler()).
 - Si UART reçoit mal → vérifier buffer et délimiteur \n.

10- Conclusion

La programmation du STM32 est essentielle pour la fiabilité des mesures et leur affichage local. Une configuration correcte des périphériques et une gestion rigoureuse des interruptions garantissent la robustesse du système.

Chapitre II : Programmation de l'esp32 :

1- Introduction

Dans ce chapitre, nous abordons la configuration de l'ESP32, qui joue le rôle de passerelle entre le STM32 et Firebase. Il permet également de lire des capteurs supplémentaires et d'agir sur des éléments de commande à distance.

2- Aperçu générale :

Lien vers le code de l'esp32

Dans ce projet l'esp32 occupe généralement les taches suivantes

- **Connexion Wi-Fi** et authentification sur Firebase RTDB.
- **Capteur CCS811** (I²C) pour CO₂ et TVOC.
- **UART2** (GPIO16 RX2, GPIO17 TX2) pour dialogue STM32 ↔ ESP32.
- **Parsing JSON** avec Arduino_JSON des trames reçues (pressure, temperature, humidity, gaz, flame).
- **Envoi périodique** (toutes les 5 s) vers Firebase :
 - UsersData/<UID>/temperature
 - .../humidity
 - .../pressure
 - .../gaz
 - .../flame
 - .../co2
- **Lecture** des nœuds Firebase btn1, btn2, btn3 pour piloter LEDs sur GPIO5, GPIO4, GPIO0.

3- Matériel nécessaire

- ESP32-WROOM-32
- Capteur CO₂ CCS811 (I²C SDA=21, SCL=22)
- Fils pour UART 2 vers STM32 (GPIO16, GPIO17)
- LEDs de statut (GPIO5, GPIO4, GPIO0, GPIO2)
- Alimentation 3.3 V

4- Schéma de connexion

Le tableau suivant représente les différentes connexion hardware de l'esp32

Fonction	ESP32 Pin	Périphérique connecté
UART RX (réception)	GPIO16 (RX2)	STM32 TX (PA2)
UART TX (transmission)	GPIO17 (TX2)	STM32 RX (PA3)
I²C SDA	GPIO21	CCS811 SDA
I²C SCL	GPIO22	CCS811 SCL
Buzzer (contrôlée par btn1)	GPIO5	Buzzer
LED2 (contrôlée par btn2)	GPIO4	LED2
LED3 (contrôlée par btn3)	GPIO0	LED3
LED statut Wi-Fi	GPIO2	Clignote pendant la connexion Wi-Fi

Tableau 2 : Schéma de connexion des capteurs avec l'esp32

5- Configuration & bibliothèques

1. Installer :

- **Arduino_JSON**
- **Adafruit_CCS811**
- **FirebaseClient + WiFiClientSecure**

2. Définir dans le code :

```
const char* ssid    = "VOTRE_SSID";
const char* password = "VOTRE_MDP";
#define FIREBASE_HOST "yourproject.firebaseio.com"
#define FIREBASE_AUTH "votre_clé_secrète"
```

3. Initialiser Serial2 :

```
Serial2.begin(115200, SERIAL_8N1, RXD2, TxD2);
```

5. Structure du code

- **setup()** :
 - Serial.begin(115200) pour debug.
 - Serial2.begin() pour UART2.
 - Connexion Wi-Fi + LED de statut.
 - Init Firebase et CCS811.

- **loop()** :

1. **Lire** trame JSON du STM32 :

```
String line = Serial2.readStringUntil('\n');
```

```
JSONVar data = JSON.parse(line);
```

2. **Extraire** pressure, temperature, humidity, gaz, flame.
3. **Lire** CO₂/TVOC du CCS811 et envoyer la valeur sur Serial2.
4. **Afficher** toutes les valeurs sur le moniteur série.
5. **Envoyer** les 6 mesures à Firebase (toutes les 5 s).
6. **Lire** btn1(btn2)/btn3 depuis Firebase, piloter LEDs GPIO5/4/0.

6. Compilation et flash

- Utiliser **Arduino IDE** ou **PlatformIO**.
- Vérifier les dépendances via le Library Manager.
- Compiler et flasher l'ESP32.
- Ouvrir le moniteur série à 115 200 baud.

7. Tests et validation

- **Wi-Fi** : LED de statut clignote jusqu'à connexion.
- **UART** : envoyer des JSONs de test depuis STM32, vérifier réception.
- **Firebase** : consulter la RTDB pour voir les valeurs mises à jour.
- **Boutons virtuels** : changer btn1/2/3 dans Firebase, observer les LEDs correspondantes.

Pour plus d'information consternant la communication entre le fire base et l'esp32 vous pouvez suivre ce [tutorial](#).

6- Conclusion

En conclusion, l'ESP32 remplit un rôle essentiel en connectant la partie embarquée du système à Internet. Il permet la visualisation distante des mesures environnementales, la gestion de capteurs supplémentaires, ainsi que l'interaction avec l'utilisateur via Firebase. Grâce à ses capacités de communication et de traitement, l'ESP32 offre une extension intelligente et connectée au système STM32.

Chapitre III : Guide de Reproduction du Projet de Station Météo Firebase

1- Introduction

Ce chapitre propose un guide complet pour reproduire l'application web du projet de station météo. Il couvre l'intégration avec Firebase, la configuration d'EmailJS pour les alertes, et le déploiement sur Firebase Hosting.

2- Vue d'Ensemble du Projet

Le projet consiste en une application web front-end qui interagit avec Firebase pour :

- Authentifier les utilisateurs.
- Afficher les données de capteurs (température, humidité, pression, gaz, flamme, CO2) stockées dans la base de données Firebase en temps réel.
- Envoyer des alertes par e-mail en cas de détection de gaz ou de flamme, en utilisant EmailJS.
- Permettre aux utilisateurs de définir une adresse e-mail d'alerte personnalisée.
- Fournir des boutons pour contrôler des états (représentés par 0 ou 1) dans la base de données.

[Lien vers le projet Fire base](#)

3- Technologies Utilisées

- **Firebase:**
 - **Firebase Authentication:** Pour la gestion des utilisateurs (connexion/déconnexion par e-mail et mot de passe).
 - **Firebase Realtime Database:** Pour stocker et synchroniser les données des capteurs en temps réel.
 - **Firebase Hosting:** Pour déployer l'application web.
- **EmailJS:** Pour envoyer des e-mails directement depuis le navigateur.
- **HTML, CSS, JavaScript :** Pour la structure, le style et la logique front-end de l'application.

4- Structure du Projet

Le projet est organisé avec les fichiers suivants :

- **public/:** Le répertoire qui contient tous les fichiers de l'application web à héberger.
 - **index.html:** La page principale du tableau de bord.
 - **style.css:** Le fichier de styles CSS pour l'application.
 - **scripts/:**

- `index.js`: Le script JavaScript principal qui gère l'initialisation de Firebase, la lecture des données, les mises à jour de l'interface utilisateur et l'envoi d'e-mails.
- `auth.js`: Le script JavaScript qui gère l'authentification des utilisateurs.
- `firebase.json`: Le fichier de configuration pour Firebase Hosting et Database.
- `database.rules.json`: Les règles de sécurité pour la Firebase Realtime Database.
- `404.html`: Une page d'erreur 404 personnalisée.

5- Étapes de Reproduction

5.1- Configuration de Firebase

1- Créer un nouveau projet Firebase:

- Allez sur la [console Firebase](#).
- Cliquez sur "Ajouter un projet" et suivez les instructions. Donnez un nom à votre projet (ex: "WeatherStationApp").

2- Mettre à niveau vers le forfait Blaze (nécessaire pour EmailJS):

- EmailJS est un service tiers et pour l'utiliser avec Firebase Functions (si vous aviez besoin de fonctions côté serveur, ce n'est pas le cas pour ce projet précis car EmailJS est utilisé côté client), vous auriez besoin du forfait Blaze. Cependant, pour l'envoi direct depuis le client comme dans ce projet, le forfait Spark (gratuit) est suffisant. Néanmoins, il est toujours bon d'être conscient des limitations.

3- Configurer Firebase Authentication:

- Dans la console Firebase de votre projet, allez dans la section "Authentication" (Authentification).
- Cliquez sur "Démarrer".
- Dans l'onglet "Méthode de connexion", activez le fournisseur "E-mail/Mot de passe".

4- Configurer Firebase Realtime Database:

- Dans la console Firebase de votre projet, allez dans la section "Realtime Database".
- Cliquez sur "Créer une base de données".
- Choisissez un emplacement pour votre base de données.
- Pour les règles de sécurité, sélectionnez le "mode de test" pour commencer, puis nous les mettrons à jour.

5- Obtenir les informations de configuration de votre application Firebase:

- Dans la console Firebase, allez dans les "Paramètres du projet" (icône d'engrenage à côté de "Présentation du projet").
- Faites défiler jusqu'à la section "Vos applications".
- Cliquez sur l'icône pour ajouter une nouvelle application web.
- Enregistrez votre application et copiez l'objet `firebaseConfig`. Cet objet contient votre `apiKey`, `authDomain`, `databaseURL`, etc. Vous en aurez besoin à l'étape suivante.

5.2- Configuration des Fichiers du Projet

1- Créer la structure de répertoires :

Créez les dossiers et fichiers comme suit :

```

1. .
2.   └── public/
3.     ├── index.html
4.     ├── style.css
5.     └── scripts/
6.       ├── index.js
7.       └── auth.js
8.   └── firebase.json
9.   └── database.rules.json
10.  └── 404.html

```

2- Mettre à jour : `public/scripts/index.js` :

- Ouvrez `public/scripts/index.js`.
- Remplacez l'objet `firebaseConfig` existant par celui que vous avez copié depuis votre console Firebase.

```

// Firebase configuration
const firebaseConfig = {
  apiKey: "VOTRE_API_KEY", // Remplacez par votre clé API
  authDomain: "VOTRE_AUTH_DOMAIN",
  databaseURL: "VOTRE_DATABASE_URL",
  projectId: "VOTRE_PROJECT_ID",
  storageBucket: "VOTRE_STORAGE_BUCKET",
  messagingSenderId: "VOTRE_MESSAGING_SENDER_ID",
  appId: "VOTRE_APP_ID",
  measurementId: "VOTRE_MEASUREMENT_ID"
};

```

3- Mettre à jour : `database.rules.json`:

- a. Copiez le contenu du fichier `database.rules.json` fourni dans votre projet.
- b. Ces règles garantissent que seuls les utilisateurs authentifiés peuvent lire et écrire leurs propres données sous leur `uid` (User ID) respectif dans le nœud `UserData`.

```
{
  "rules": {
    "UserData": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

4- Mettre à jour `firebase.json`:

- Copiez le contenu du fichier `firebase.json` fourni dans votre projet.
- Ce fichier configure Firebase Hosting pour servir les fichiers du répertoire public et spécifie l'emplacement des règles de la base de données.

```
{
  "hosting": {
    "public": "public",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ]
  },
  "database": {
    "rules": "database.rules.json"
  }
}
```

5- Copier les autres fichiers :

- Copiez le contenu de `public/index.html`, `public/style.css`, `public/scripts/auth.js` et `404.html` dans les fichiers correspondants de votre projet.

5.3- Configuration d'EmailJS

1- Créer un compte EmailJS:

- Allez sur le site d'EmailJS : <https://www.emailjs.com/>
- Inscrivez-vous pour un nouveau compte.

2- Ajouter un nouveau service d'e-mail:

- Dans votre tableau de bord EmailJS, cliquez sur "Add New Service".
- Sélectionnez le service que vous souhaitez utiliser (par exemple, Gmail, Outlook, etc.). Suivez les instructions pour l'authentifier.
- Notez votre **Service ID**.

3- Créer un nouveau modèle d'e-mail:

- Dans votre tableau de bord EmailJS, allez dans "Email Templates".
- Cliquez sur "Create New Template".

- Configurez votre modèle d'e-mail. Assurez-vous d'inclure des variables de remplacement pour le sujet ({{subject}}), le message ({{message}}) et l'e-mail du destinataire ({{to_email}}) comme dans le code `index.js`.
- Exemple de contenu de modèle :
 - **Subject:** {{subject}}
 - **Body:** Hello, {{message}}
- Notez votre **Template ID**. Le code utilise `template_nu0p69n`.

4- Obtenir votre Public Key (User ID):

- Dans votre tableau de bord EmailJS, allez dans "Account".
- Votre "Public Key" (également appelé User ID) est affichée ici. Notez-la. Le code utilise `FOY90D2TQq6L0rT84`.

5- Mettre à jour `public/scripts/index.js` avec vos clés EmailJS:

- Ouvrez `public/scripts/index.js`.
- Trouvez la ligne `emailjs.init("FOY90D2TQq6L0rT84");` et remplacez `FOY90D2TQq6L0rT84` par votre **Public Key (User ID)**.
- Trouvez les appels à `emailjs.send("service_bf82mnr", "template_nu0p69n", ...)` et remplacez `"service_bf82mnr"` par votre **Service ID** et `"template_nu0p69n"` par votre **Template ID**.

```

let currentAlertEmail = "omejri417@gmail.com"; // Peut être une valeur
par défaut
emailjs.init("VOTRE_EMAILJS_PUBLIC_KEY"); // Remplacez par votre clé
publique EmailJS

// ...

emailjs.send("VOTRE_EMAILJS_SERVICE_ID", "VOTRE_EMAILJS_TEMPLATE_ID", {
  to_email: currentAlertEmail,
  subject: "Utilisateur connecté",
  message: "bon retour !"
});

// ...

emailjs.send("VOTRE_EMAILJS_SERVICE_ID", "VOTRE_EMAILJS_TEMPLATE_ID", {
  to_email: currentAlertEmail,
  subject: "Alerte : Gaz détecté",
  message: "⚠️ Attention : un gaz a été détecté par votre station météo
!"
});

// ...

emailjs.send("VOTRE_EMAILJS_SERVICE_ID", "VOTRE_EMAILJS_TEMPLATE_ID", {
  to_email: currentAlertEmail,
  subject: "Alerte : flamme détecté",
  message: "⚠️ Attention : un flamme a été détecté par votre station
météo !"
});

```

5.4- Installation et Déploiement

1- Installer Firebase CLI:

- Si vous ne l'avez pas déjà fait, installez Node.js.
- Ouvrez votre terminal ou invite de commande.
- Installez la Firebase CLI globalement en exécutant :

```
npm install -g firebase-tools
```

2- Se connecter à Firebase:

- Dans votre terminal, accédez au répertoire racine de votre projet (là où se trouvent `firebase.json` et `public/`).
- Connectez-vous à votre compte Firebase :

```
firebase login
```

- Suivez les instructions pour vous authentifier via votre navigateur.

3- Initialiser le projet Firebase:

- Dans votre terminal, dans le répertoire racine de votre projet, initialisez Firebase :

```
firebase init
```

- Répondez aux questions :

- "Which Firebase features do you want to set up for this directory?": Sélectionnez "**Firestore**" (si vous voulez l'utiliser à l'avenir), "**Realtime Database**" et "**Hosting**" en utilisant la barre d'espace pour sélectionner et Entrée pour confirmer.
- "Please select a Firebase project for this directory": Choisissez le projet Firebase que vous avez créé précédemment.
- "What file do you want to use for Database Rules?": Appuyez sur Entrée pour accepter `database.rules.json` (le fichier que vous avez déjà).
- "What do you want to use as your public directory?": Tapez `public` et appuyez sur Entrée.
- "Configure as a single-page app (rewrite all urls to /index.html)": Tapez `N` (non) car nous avons un `404.html` séparé.
- "Set up automatic builds and deploys with GitHub?": Tapez `N` (non) pour l'instant, à moins que vous ne vouliez configurer l'intégration continue.
- "File `public/index.html` already exists. Overwrite?": Tapez `N` (non).

4- Déployer l'application :

- Dans votre terminal, dans le répertoire racine de votre projet, déployez votre application sur Firebase :

```
firebase deploy
```

- Cela déploiera vos règles de base de données et votre application web sur Firebase Hosting. Une fois le déploiement terminé, vous obtiendrez une "URL d'hébergement" où votre application est accessible.

5.5- Interaction avec la Base de Données

Activer le projet (esp32 et stm32) pour recevoir les mesures des Capteurs.

Aussi pour tester, vous pouvez ajouter manuellement des données via la console Firebase :

1- Accédez à votre Realtime Database:

- Dans la console Firebase, allez dans la section "Realtime Database".

2- Créez les nœuds de données :

```
UsersData
  └── YOUR_UID
      ├── alertEmail: "your_alert_email@example.com"
      ├── co2: "450"
      ├── flame: "No flame detected"
      ├── gaz: "No Gaz detected"
      ├── humidity: 65.23
      ├── pressure: 1012.56
      ├── temperature: 25.78
      ├── button1: 0
      ├── button2: 0
      └── button3: 0
```

- Remplacez `YOUR_UID` par l'UID réel de l'utilisateur qui se connecte à l'application. Vous pouvez trouver l'UID d'un utilisateur dans la section "Authentication" de la console Firebase.
- Pour simuler des alertes :
 - Changez la valeur de `gaz` à " Gaz detected" pour déclencher l'alerte gaz.
 - Changez la valeur de `flame` à " flame detected" pour déclencher l'alerte flamme.
 - L'application devrait envoyer un e-mail à l'adresse définie dans `alertEmail` la première fois que ces valeurs changent vers l'état "détecté".

6- Fonctionnalités Clés du Code

- **index.js:**
 - Initialisation de Firebase et de ses services (`initializeApp`, `getAuth`, `getDatabase`).
 - Gestion de l'interface utilisateur (`setupUI`) : affiche/masque les éléments en fonction de l'état de connexion de l'utilisateur.
 - Lecture des données en temps réel (`onValue`) depuis Firebase Realtime Database pour chaque capteur (température, humidité, pression, gaz, flamme, CO2).
 - Mise à jour dynamique de l'interface utilisateur avec les dernières lectures des capteurs.
 - Gestion de l'entrée de l'e-mail d'alerte : Lit l'e-mail depuis la base de données et met à jour la base de données lorsque l'utilisateur modifie l'e-mail.
 - Intégration d'EmailJS pour envoyer des e-mails d'alerte en cas de détection de gaz ou de flamme, avec un mécanisme pour n'envoyer l'alerte qu'une seule fois par événement.
 - Gestion des boutons personnalisés : Lit leur état depuis la base de données et met à jour l'état dans la base de données lors d'un clic.
- **auth.js:**
 - Importe le module `auth` de `index.js`.
 - Écoute les changements d'état d'authentification (`onAuthStateChanged`) pour appeler `setupUI` et mettre à jour l'interface.
 - Gère la logique de connexion (`signInWithEmailAndPassword`) et de déconnexion (`signOut`).
- **style.css:**
 - Fournit les styles visuels pour le tableau de bord, y compris la disposition des cartes des capteurs, la barre de navigation et les formulaires.

7- Conclusion

En suivant ce guide, vous devriez être en mesure de reproduire et de comprendre le fonctionnement de ce projet de tableau de bord de station météo basé sur Firebase.

Conclusion générale

Ce projet de station météo intelligente illustre parfaitement l'intégration harmonieuse entre les systèmes embarqués, les technologies de communication sans fil et les services cloud. En combinant la puissance de traitement locale du **STM32**, la connectivité de l'**ESP32**, et les capacités de stockage et de synchronisation en temps réel de **Firebase**, le système permet une acquisition, une transmission et une visualisation fluide des données environnementales.

L'ajout de **EmailJS** pour l'envoi d'alertes par e-mail rend le système proactif, capable de notifier les utilisateurs en cas de détection de gaz ou de flamme, renforçant ainsi l'aspect sécuritaire de l'application. De plus, la conception d'un tableau de bord web interactif offre une interface utilisateur conviviale, permettant à chacun de consulter les mesures en temps réel et de contrôler des dispositifs à distance via des boutons virtuels.

Ce projet met donc en évidence :

La complémentarité entre le traitement local (STM32) et la connectivité (ESP32),

L'utilité des bases de données cloud pour la surveillance à distance,

Et l'intérêt croissant des solutions **IoT** dans les domaines de l'environnement, de la sécurité et de l'automatisation.

Enfin, ce travail ouvre la voie à de nombreuses perspectives d'évolution : ajout de capteurs supplémentaires, implémentation de l'intelligence artificielle pour l'analyse des données...