Department of Electrical and Electronic Engineering
Course Name: Internet of Things-1
Course Code: EEE 4523
Sec: A

IOT Project : Smart EV Charging Station Monitoring & Safety Management

Submitted By,

Team-Copilot

MD. AL AMIN ID:021 222 029
YAMIN AHMED ID:021 223 0024
MAHMUD MOLLA MIAD ID: 021 223 0062

Submitted to,
Prof. Dr. Siddique Mohammad Lutful Kabir
Professor at United International University

Project Description:

We build an IOT ecosystem with some esp32 and sensors (ir and flame) and connect those esp32 to wifi and send data to a realtime database, and database send those data to the website .This ecosystem provide the owner to monitor his charging station from a website. Here he can shutdown the EV station by clicking a button from website. On the other hand,the customer of EV station can know which station is free and which station has less traffic , so that he can save his time. As more people use electric cars, it becomes important to know which charging spots are free and to react quickly if there is an emergency.

The project description and Procedure:

Step 1: Hardware Setup & Integration

- Occupancy Sensing: We install IR (Infrared) sensors in each charging bay and waiting lane. When a car blocks the sensor, it signals that the slot is "Busy."
- Safety Sensing: We install Flame sensors around the electrical equipment to detect any sign of fire.
- Control Circuit: We connect a Relay Module to the main power line of the station, allowing the ESP32 to cut the power when commanded.

Step 2: Connecting to the Cloud (Firebase)

- The ESP32 is programmed to connect to the local WiFi.
- Here we use Firebase Realtime Database as a bridge.
- Every few seconds, the ESP32 "pushes" the status of the IR and Flame sensors to the cloud.
- At the same time, it "listens" for any commands (like Shutdown or LED control) coming from the website.

Step 3: Developing Website for owner and customer

- We built a web-based dashboard that fetches data from Firebase.

- The website visualizes each station (e.g., Basundhara, Rampura, Satarkul) with clear indicators.

- We programmed a "Shutdown" button on the interface. When clicked, it changes a value in the database, which the ESP32 sees instantly and triggers the relay to turn off the power.

Step 4 : Implementing the Emergency Alert System
To ensure 24/7 safety, we created a background Python script.

- If a fire is detected, the script immediately uses an API to place a **voice call** to the owner's phone and send a Telegram alert. This ensures the owner is notified even if they aren't looking at the website.

Step 5 : Management Analytics
We implemented a secondary Manager Summary system using Python to process the raw data into business insights:

- Active Load Tracking: The system calculates the total network load .

- Revenue Monitoring: It tracks charging sessions and calculates estimated revenue based on a fixed rate (e.g., $0.20 per minute).

- Network Summary: A dedicated management view provides a bird's-eye view of all stations, showing which are online and how many cars are currently charging across the entire city.

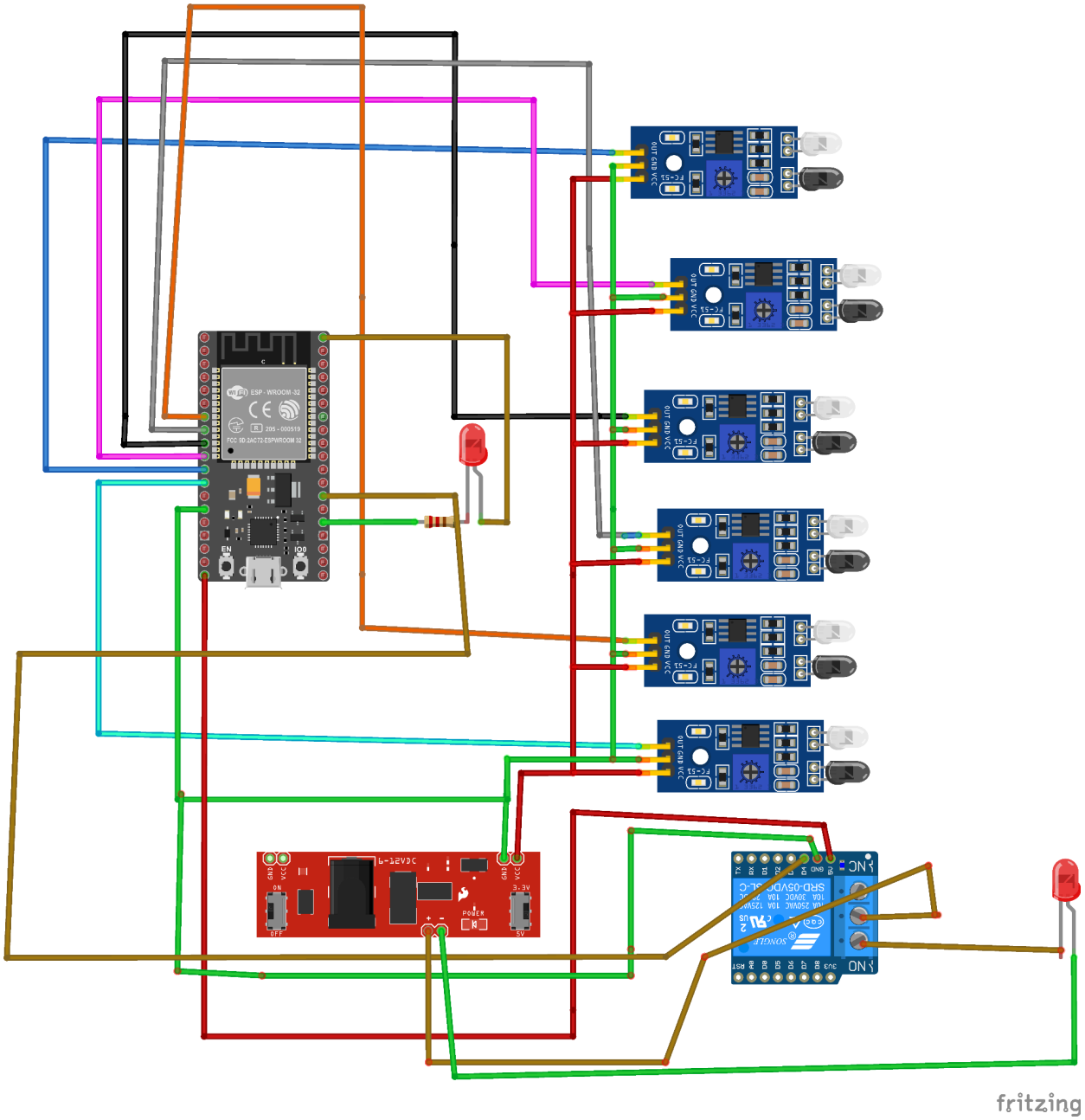
Technical Summary

- Microcontroller: ESP32 (WiFi Enabled)

- Sensors: IR Sensors (Occupancy), Flame Sensors (Fire Detection)

- Actuators: Relays (Power Control), LEDs (Status)
- Database: Firebase Realtime Database
- Alerts: Python, Telegram API, and Voice Call Integration

Circuit diagram:



Component:
1. ESP32
2. IR sensor
3. Flame Sensor
4. Led
5. Register
5. 5 V and 3.3 V power supply

CODE USED :

main/main.cpp:

```
#include <iostream>
#include <string>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"

#include "driver/gpio.h"

#include "esp_firebase/app.h"
#include "esp_firebase/rtdb.h"

#include "wifi_utils.h"
#include "firebase_config.h"

using namespace ESPFirebase;

// ========================================
// CONFIGURATION
// ========================================
#define DEVICE_ID "ESP32_04"

// Output Pins
```

```
#define RELAY_GPIO GPIO_NUM_4    // Relay (0 = ON, 1 = OFF)
#define LED_GPIO   GPIO_NUM_2    // Internal LED

// Input Pins (Sensors)
#define IR1_GPIO GPIO_NUM_27
#define IR2_GPIO GPIO_NUM_26
#define IR3_GPIO GPIO_NUM_25
#define IR4_GPIO GPIO_NUM_33
#define IR5_GPIO GPIO_NUM_32
#define IR6_GPIO GPIO_NUM_14
#define FLAME_GPIO GPIO_NUM_13

static const char *TAG = "EV_STATION_INTEGRATED";

/* ------------ Hardware Initialization ------------ */
void hardware_init()
{
    // Configure Sensors as Inputs
    gpio_config_t io_conf = {};
    io_conf.pin_bit_mask = (1ULL << IR1_GPIO) | (1ULL << IR2_GPIO) |
                           (1ULL << IR3_GPIO) | (1ULL << IR4_GPIO) |
                           (1ULL << IR5_GPIO) | (1ULL << IR6_GPIO) |
                           (1ULL << FLAME_GPIO);
    io_conf.mode = GPIO_MODE_INPUT;
    io_conf.pull_up_en = GPIO_PULLUP_ENABLE;
    gpio_config(&io_conf);

    // Configure Relay and LED as Outputs
    gpio_reset_pin(RELAY_GPIO);
    gpio_set_direction(RELAY_GPIO, GPIO_MODE_OUTPUT);
    gpio_reset_pin(LED_GPIO);
    gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);

    // Initial States
    gpio_set_level(RELAY_GPIO, 0); // Default Relay ON
    gpio_set_level(LED_GPIO, 0);   // Default LED OFF
}

/* ------------ Main Application ------------ */
extern "C" void app_main(void)
```

```
{
    // 1. Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    // 2. WiFi Connect
    wifiInit(SSID, PASSWORD);
    ESP_LOGI(TAG, "WiFi connected. Device ID: %s", DEVICE_ID);

    // 3. Firebase Login
    user_account_t account = {USER_EMAIL, USER_PASSWORD};
    FirebaseApp app(API_KEY);
    app.loginUserAccount(account);
    RTDB db(&app, DATABASE_URL);

    // 4. Setup Hardware
    hardware_init();

    std::string path = "/devices/" + std::string(DEVICE_ID);

    while (true)
    {
        // --- STEP A: Fetch Control Data from Firebase ---
        Json::Value device_node = db.getData(path.c_str());

        // Control LED via "Sensor" key (1 = LED ON)
        // Check for both Integer and String formats to prevent errors
        if (device_node["Sensor"].asInt() == 1 ||
device_node["Sensor"].asString() == "1") {
            gpio_set_level(LED_GPIO, 1);
        } else {
            gpio_set_level(LED_GPIO, 0);
        }

        // Control Relay via "Shutdown" key (1 = Relay OFF)
```

```
        int shutdown_val = 0;
        if (device_node["Shutdown"].asInt() == 1 ||
device_node["Shutdown"].asString() == "1") {
            shutdown_val = 1;
            gpio_set_level(RELAY_GPIO, 1); // Logic HIGH turns Relay OFF
        } else {
            shutdown_val = 0;
            gpio_set_level(RELAY_GPIO, 0); // Logic LOW turns Relay ON
        }

        // --- STEP B: Read Sensor Data ---
        Json::Value sync_data;
        sync_data["ir1"] = (gpio_get_level(IR1_GPIO) == 0);
        sync_data["ir2"] = (gpio_get_level(IR2_GPIO) == 0);
        sync_data["ir3"] = (gpio_get_level(IR3_GPIO) == 0);
        sync_data["ir4"] = (gpio_get_level(IR4_GPIO) == 0);
        sync_data["ir5"] = (gpio_get_level(IR5_GPIO) == 0);
        sync_data["ir6"] = (gpio_get_level(IR6_GPIO) == 0);
        sync_data["flame_detected"] = (gpio_get_level(FLAME_GPIO) == 0);

        // --- STEP C: Maintain Controls in Firebase ---
        // We include these in putData so our manual dashboard changes
aren't overwritten by the ESP32
        sync_data["Sensor"] = device_node["Sensor"];
        sync_data["Shutdown"] = shutdown_val;

        // --- STEP D: Sync to Firebase ---
        db.putData(path.c_str(), sync_data);

        ESP_LOGI(TAG, "Sync: LED=%d, Shutdown=%d, Path=%s",
                 device_node["Sensor"].asInt(), shutdown_val,
path.c_str());

        vTaskDelay(pdMS_TO_TICKS(3000));
    }
}
```

main/firebase_config.h:

```
// Wifi Credentials
#define SSID "********"
#define PASSWORD "**********"

// Read readme.md to properly configure api key and authentication

// create a new api key and add it here
#define API_KEY "AIzaSyBBacn2C_AmruoHZcb3vmZIXysTQRPrvIk"
// Copy your firebase real time database link here
#define DATABASE_URL
"https://bd-ev-ch-default-rtdb.asia-southeast1.firebasedatabase.app/"

#define USER_EMAIL "*****************"   // This gmail does not exist
outside your database. it only exists in the firebase project as a user
#define USER_PASSWORD "12345678"
```

main/wifi_utils.h

```
#include "esp_event.h"
#include "esp_log.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "freertos/task.h"
#include "nvs_flash.h"
#include <string.h>
#include "lwip/err.h"
#include "lwip/sys.h"

#include "esp_wifi_types.h"
#ifdef __cplusplus
extern "C" {
#endif
```

```
#define EXAMPLE_ESP_MAXIMUM_RETRY 5

static EventGroupHandle_t s_wifi_event_group;

#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT BIT1

#define WIFI_TAG "wifi station"

static int s_retry_num = 0;

static void event_handler(void *arg, esp_event_base_t event_base, int32_t
event_id, void *event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
    {
        esp_wifi_connect();
    }
    else if (event_base == WIFI_EVENT &&event_id ==
WIFI_EVENT_STA_DISCONNECTED)
    {
        if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY)
        {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(WIFI_TAG, "retry to connect to the AP");
        }
        else
        {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(WIFI_TAG, "connect to the AP fail");
    }
    else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
    {
        ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
        ESP_LOGI(WIFI_TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
```

```
    }
    else if (event_base == WIFI_EVENT && event_id ==
WIFI_REASON_BEACON_TIMEOUT)
    {
        ESP_LOGE("WiFi", "WIFI_REASON_BEACON_TIMEOUT");
        // Handle wrong password scenario here
    }
}

void wifi_init_sta(const char* ssid, const char* password) {
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
ESP_EVENT_ANY_ID, &event_handler, NULL, &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
IP_EVENT_STA_GOT_IP, &event_handler, NULL, &instance_got_ip));
    wifi_config_t wifi_config = {0};
    memcpy(wifi_config.sta.ssid, ssid, strlen(ssid));
    memcpy(wifi_config.sta.password, password, strlen(password));

    wifi_config.sta.threshold.authmode = WIFI_AUTH_OPEN;
    wifi_config.sta.pmf_cfg = {true, false};
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(WIFI_TAG, "wifi_init_sta finished.");

    /* Waiting until either the connection is established
(WIFI_CONNECTED_BIT) or
```

```
    * connection failed for the maximum number of re-tries
(WIFI_FAIL_BIT). The
    * bits are set by event_handler() (see above) */
    EventBits_t bits =
xEventGroupWaitBits(s_wifi_event_group,WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
pdFALSE, pdFALSE, portMAX_DELAY);

    /* xEventGroupWaitBits() returns the bits before the call returned,
hence we
    * can test which event actually happened. */
    if (bits & WIFI_CONNECTED_BIT)
    {
        ESP_LOGW(WIFI_TAG, "connected to ap SSID: %s || password: %s",
ssid, password);
    }
    else if (bits & WIFI_FAIL_BIT)
    {
        ESP_LOGE(WIFI_TAG, "Failed to connect to SSID: %s || password:
%s", ssid, password);
    }
    else
    {
        ESP_LOGE(WIFI_TAG, "UNEXPECTED EVENT");
    }

    /* The event will not be processed after unregister */
    ESP_ERROR_CHECK(esp_event_handler_instance_unregister(IP_EVENT,
IP_EVENT_STA_GOT_IP, instance_got_ip));
    ESP_ERROR_CHECK(esp_event_handler_instance_unregister(WIFI_EVENT,
ESP_EVENT_ANY_ID, instance_any_id));
    vEventGroupDelete(s_wifi_event_group);
}
void wifiInit(const char* ssid, const char* password) {
    // Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
```

```
    }
    ESP_ERROR_CHECK(ret);
    ESP_LOGI(WIFI_TAG, "ESP_WIFI_MODE_STA");
    wifi_init_sta(ssid, password);
}



#ifdef __cplusplus
}
#endif
```

Python code which is running at any server or pc :

```
import firebase_admin
from firebase_admin import credentials, db
import requests
import time

# --- CONFIGURATION ---
TELEGRAM_USER = "@alamindac"
DATABASE_URL =
"https://bd-ev-ch-default-rtdb.asia-southeast1.firebasedatabase.app"

# Cooldown: Wait 60 seconds before calling again for the same station
CALL_COOLDOWN = 60
last_call_time = {}

STATION_NAMES = {
   "ESP32_01": "Station One Basundhara",
   "ESP32_02": "Station Two Rampura",
   "ESP32_03": "Station Three Satarkul",
   "ESP32_04": "Station Four Natunbazar"
}
```

```
# --- FIREBASE SETUP ---
if not firebase_admin._apps:
    cred = credentials.Certificate("serviceAccountKey.json")
    firebase_admin.initialize_app(cred, {'databaseURL': DATABASE_URL})
print("✅ Connected to Firebase")

def trigger_telegram_call(station_id):
    global last_call_time
    current_time = time.time()

    if station_id not in last_call_time or (current_time - last_call_time[station_id]) >
CALL_COOLDOWN:
        name = STATION_NAMES.get(station_id, station_id)
        message = f"Emergency Alert! Fire detected at {name}."
        url =
f"http://api.callmebot.com/start.php?user={TELEGRAM_USER}&text={message}&lang=
en-US-Standard-B&rpt=2"

        try:
            print(f"☎️ TRIGGERING CALL FOR: {name}")
            response = requests.get(url)
            if response.status_code == 200:
                last_call_time[station_id] = current_time
                print(f"📞 Call successful for {name}")
        except Exception as e:
            print(f"❌ Call Error: {e}")
    else:
        # Calculate remaining seconds for the cooldown
        remaining = int(CALL_COOLDOWN - (current_time - last_call_time[station_id]))
        print(f"⏳ Cooldown active for {station_id} ({remaining}s left)")

def fire_listener(event):
    # Log everything to the console so we can monitor
    print(f"LOG: Path={event.path} Data={event.data}")

    # CASE 1: The event path itself contains the station and field (e.g.
/ESP32_01/flame_detected)
    if "flame_detected" in event.path and event.data == True:
        station_id = event.path.split('/')[1]
```

```
        trigger_telegram_call(station_id)

    # CASE 2: The event path is just the station (e.g. /ESP32_03) and the data is a
dictionary
    elif isinstance(event.data, dict) and event.data.get("flame_detected") == True:
        station_id = event.path.strip('/')
        # Handle cases where path is just '/'
        if not station_id:
            # If the whole 'devices' node was updated, loop through it
            for sid, values in event.data.items():
                if isinstance(values, dict) and values.get("flame_detected") == True:
                    trigger_telegram_call(sid)
        else:
            trigger_telegram_call(station_id)

# Watch the 'devices' folder
db.reference('devices').listen(fire_listener)

print("🔥 MONITORING ACTIVE - SYSTEM STABILIZED")
while True:
    time.sleep(1)
```

Code Explanation:

Explanation of main/main.cpp

Station Control Logic:

1. Hardware Initialization (hardware_init)

Before the station starts monitoring, it must configure its physical pins (GPIOs):

Sensor Inputs: Six IR sensors and one Flame sensor are configured as GPIO_MODE_INPUT. They use "Pull-up" resistors to ensure stable readings and prevent electrical noise.

Control Outputs: The Relay (power control) and the internal LED (status indicator) are configured as GPIO_MODE_OUTPUT.

Default State: Upon startup, the station is programmed to be "ON" by default (Relay set to Low). This ensures the charging service stays active even if the internet is briefly interrupted during boot-up.

2. System Setup (app_main)

When the device powers on, it follows a strict 4-step sequence:

1. Memory Start: It initializes the NVS (Non-Volatile Storage) to handle system configurations.

2. WiFi Handshake: It connects to the internet using the wifiInit utility.

3. Firebase Authentication: The station logs in using a unique API Key and User Account to ensure only authorized devices can send data to your database.

4. Device Identification: Each station identifies itself (e.g., DEVICE_ID "ESP32_04”,”ESP_03”), so the dashboard knows exactly which location (Basundhara, Rampura, etc.) is being updated.

3. The Infinite Sync Loop (Real-Time Operation)

Once connected, the code runs a continuous loop every 3 seconds to perform the following tasks:

Step A: Remote Command Processing The station "pulls" data from the Firebase path /devices/ESP32_04.

• Shutdown Control: If the website’s "Shutdown" value is set to 1, the ESP32 flips the relay to cut power to the station.

• Visual Indicator: The "Sensor" key in the database controls the station's local LED, allowing the owner to flash lights for maintenance or identification.

Step B: Sensor Data Acquisition The ESP32 "reads" the physical state of the charging bays:

• IR Sensors (ir1–ir6): It checks 6 different spots to see if a car is present. Logic 0 means the sensor is blocked (car detected), which the code converts to a True value for the database.

• Fire Watch: It continuously checks the Flame sensor. If fire is detected, it flags flame_detected as True.

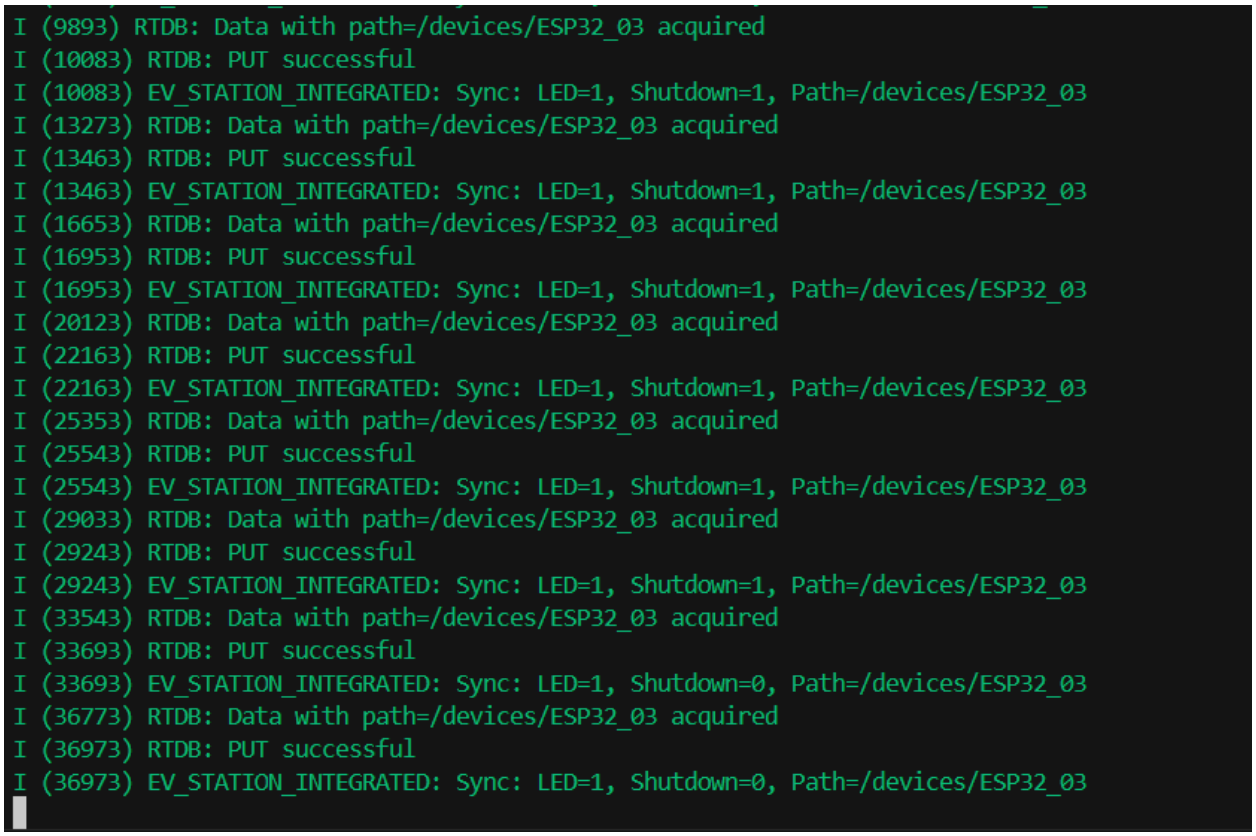
Step C: Data Synchronization (putData) Finally, the station "pushes" all this information back to the cloud. It packages the sensor results and the current power status into a JSON object and updates Firebase. This ensures that what you see on the website is an exact live reflection of the physical station.

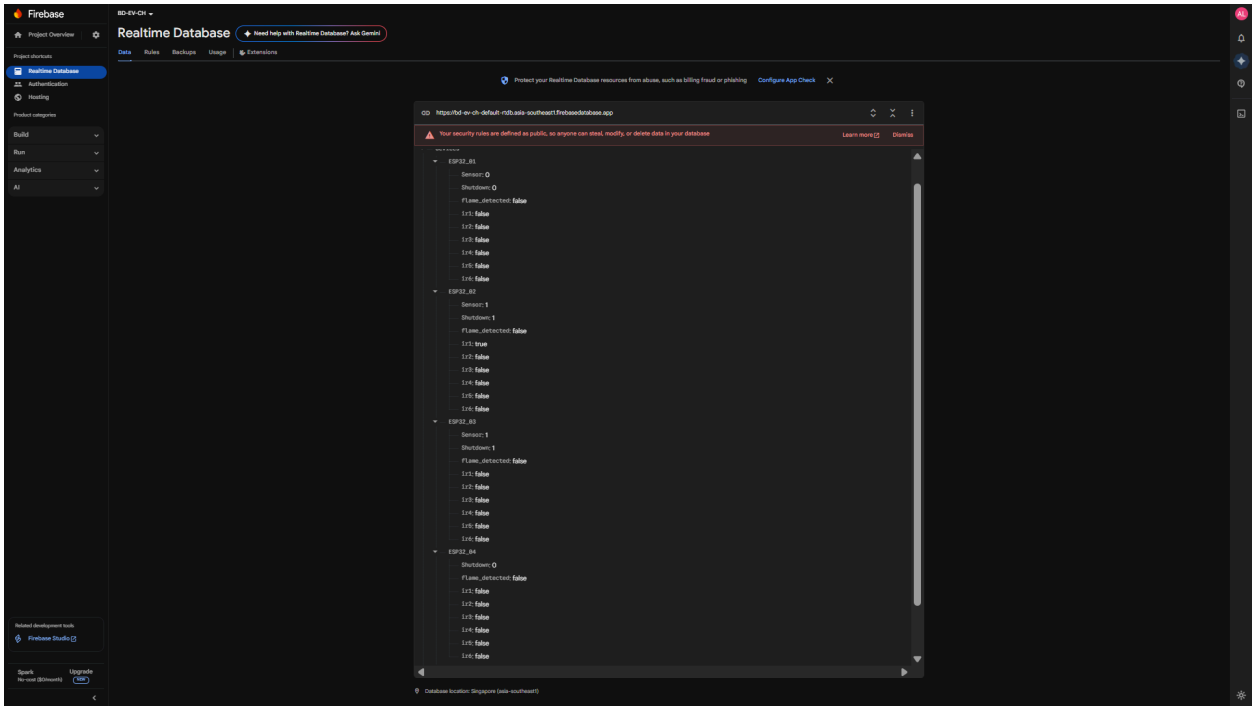
4. Reliability & Timing

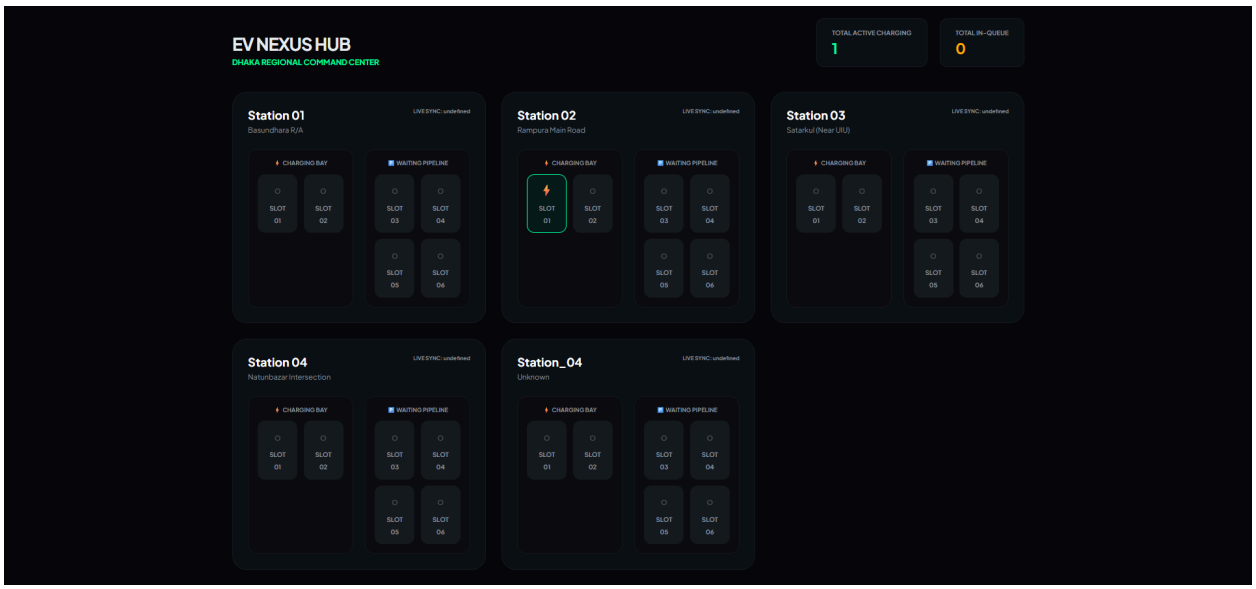
The system uses vTaskDelay to manage network traffic efficiently. By syncing every 3,000 milliseconds (3 seconds), the station provides "near-instant" updates while remaining stable and preventing the ESP32 from overheating during 24/7 operation.

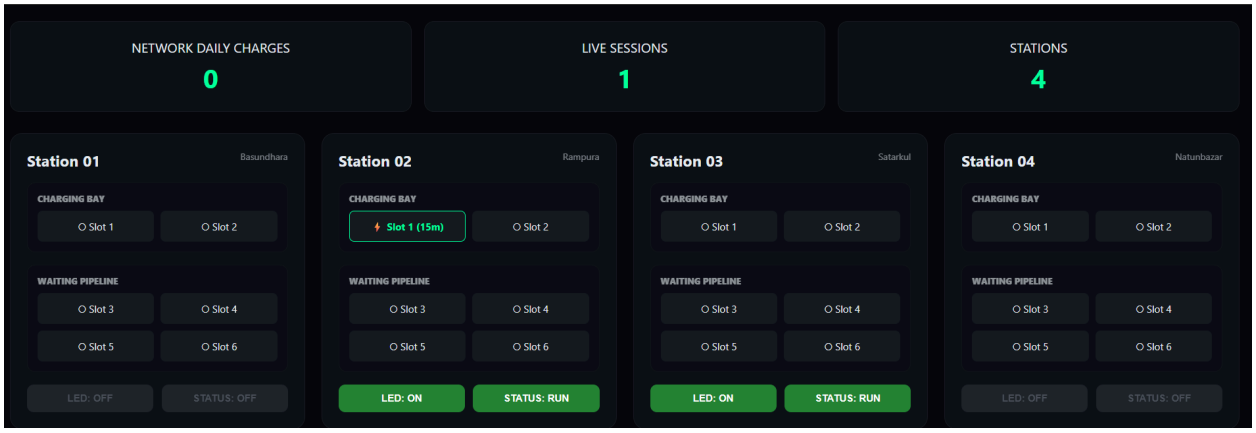
Results:

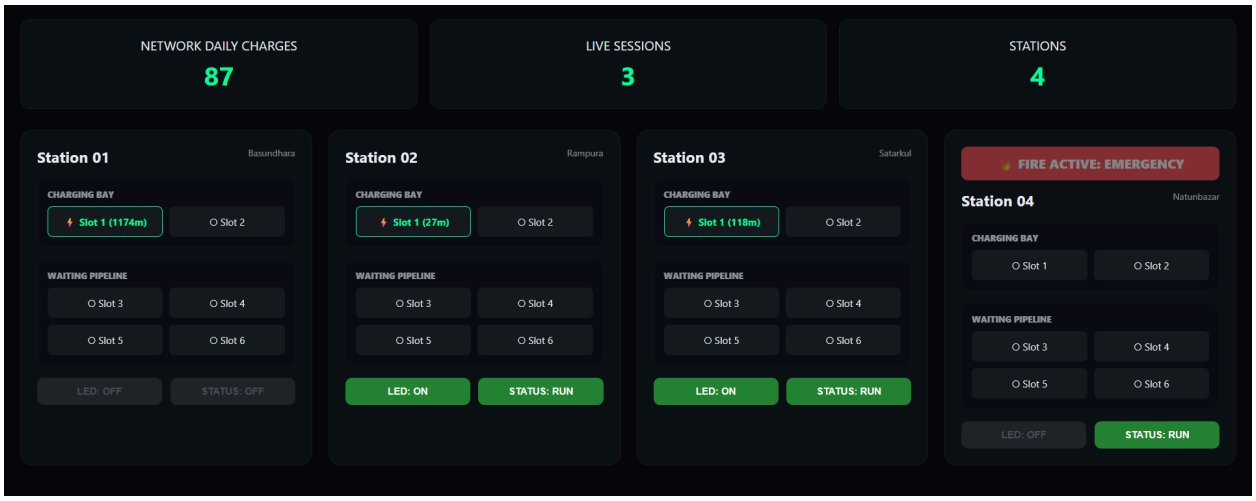
First ESP32 connect to wifi then try to log in firebase .If it is successfull then it send data to its path . Here our path=/devices/ESP32_01. After this it sync data from firebase, mainly firebase sends data to the ESP32 . Thus firebase update its database. The website continuously monitors the firebase real time database and update its dashboard. If the owner of ev click the led/shutdown button , the firebase automatically update those data to its realtime database. As we know, after a successful put to firebase, firebase send data to esp32 ,so the the command come from the owner after the successful put. And esp32 will execute the desired action. We run a python script in the pc .which continuously monitor is there any fire occur, if fire occur then it will send a phone call to the specific number . This python script will say which EV charging station set on fire.Such, if fire occur at EV station 3 in satarkul then it will exactly say ,Emergency Alert! Fire detected at Station Three Satarkul. The website of the EV charging station will automatically count how many car get charged.




Firebase
Realtime Database
Need help with Realtime Database? Ask Gemini
Data
Rules
Backups
Usage
Extensions
Protect your Realtime Database resources from abuse, such as billing fraud or phishing
Configure App Check
https://bd-ev-ch-default-rtdb.asia-southeast1.firebasedatabase.app
Your security rules are defined as public, so anyone can steal, modify, or delete data in your database
Learn more
Dismiss
ESP32_01
Sensor: 0
Shutdown: 0
flame_detected: false
ir1: false
ir2: false
ir3: false
ir4: false
ir5: false
ir6: false
ESP32_02
Sensor: 1
Shutdown: 1
flame_detected: false
ir1: true
ir2: false
ir3: false
ir4: false
ir5: false
ir6: false
ESP32_03
Sensor: 1
Shutdown: 1
flame_detected: false
ir1: false
ir2: false
ir3: false
ir4: false
ir5: false
ir6: false
ESP32_04
Shutdown: 0
flame_detected: false
ir1: false
ir2: false
ir3: false
ir4: false
ir5: false
ir6: false
Database location: Singapore (asia-southeast1)


EV NEXUS HUB
DHAKA REGIONAL COMMAND CENTER
TOTAL ACTIVE CHARGING
1
TOTAL IN-QUEUE
0
Station 01
Basundhara R/A
LIVE SYNC: undefined
CHARGING BAY
WAITING PIPELINE
SLOT 01
SLOT 02
SLOT 03
SLOT 04
SLOT 05
SLOT 06
Station 02
Rampura Main Road
Station 03
Satarkul (Near UIU)
Station 04
Natunbazar Intersection
Station_04
Unknown





Thonny - C:\Users\alami\Downloads\EV_HTML_04ta_ESP32\python\manager_view.py @ 12 : 42

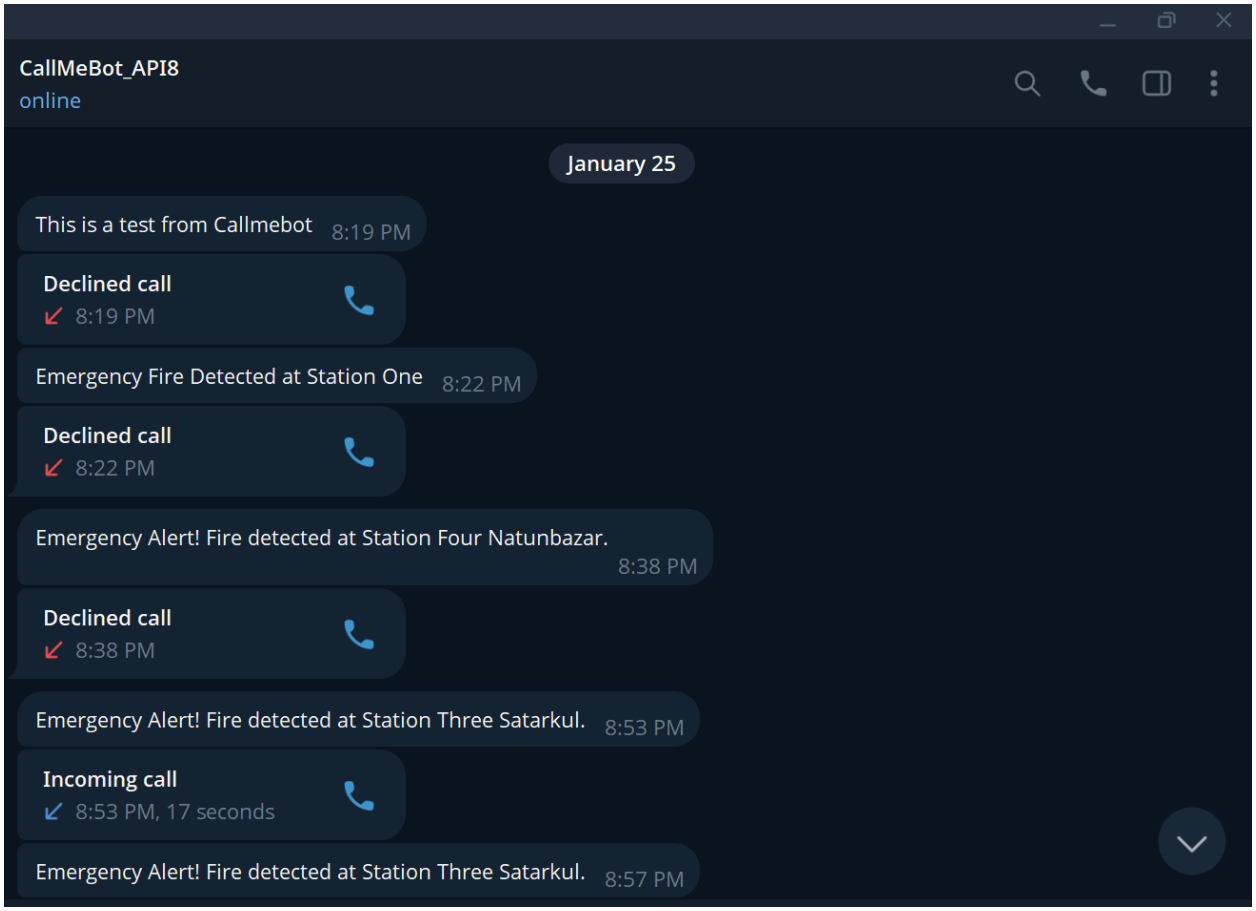
File Edit View Run Tools Help

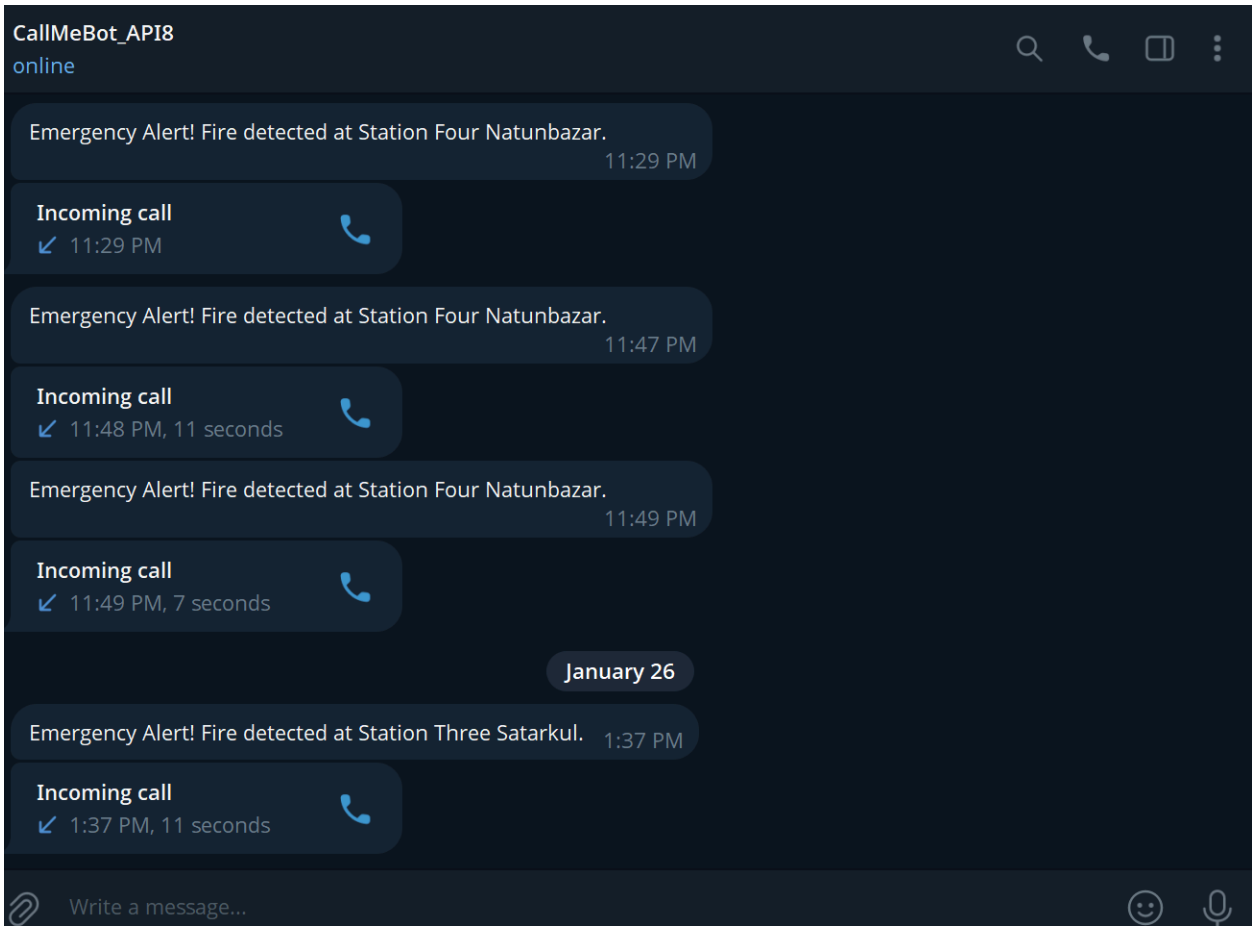
actual_fire_alerm_telegram.py | LAST.py | manager_view.py

```python
import firebase_admin
from firebase_admin import credentials, db
import time
import os

# --- CONFIGURATION ---
CH_RATE = 0.20  # Price per minute of charging
STATIONS = ["ESP32_01", "ESP32_02", "ESP32_03", "ESP32_04"]

if not firebase_admin._apps:
    cred = credentials.Certificate("serviceAccountKey.json")
    firebase_admin.initialize_app(cred, {
        'databaseURL': 'https://bd-ev-ch-default-rtdb.asia-southeast1.firebasedatabase.app'
    })
```

Shell

```
>>> %Run LAST.py
 Connected to Firebase
LOG: Path=/ Data={'ESP32_01': {'Sensor': 0, 'Shutdown': 0, 'flame_detected': False, 'ir1': False, 'ir2': False, 'ir3': False, 'ir4': False, 'ir
5': False, 'ir6': False}, 'ESP32_02': {'Sensor': 1, 'Shutdown': 1, 'flame_detected': False, 'ir1': True, 'ir2': False, 'ir3': False, 'ir4': Fal
se, 'ir5': False, 'ir6': False}, 'ESP32_03': {'Sensor': 1, 'Shutdown': 1, 'flame_detected': False, 'ir1': False, 'ir2': False, 'ir3': False, 'i
r4': False, 'ir5': False, 'ir6': False}, 'ESP32_04': {'Shutdown': 0, 'flame_detected': False, 'ir1': False, 'ir2': False, 'ir3': False, 'ir4':
False, 'ir5': False, 'ir6': False}, 'Station_04': {'Sensor': 0, 'Shutdown': 1, 'flame_detected': False, 'ir1': False, 'ir2': False, 'ir3': Fals
e, 'ir4': False, 'ir5': False, 'ir6': False, 'shutdown': 0}}🔥 MONITORING ACTIVE - FAILSAFE ENABLED

LOG: Path=/ESP32_03 Data={'Sensor': 1, 'Shutdown': 1, 'flame_detected': True, 'ir1': False, 'ir2': False, 'ir3': False, 'ir4': False, 'ir5': Fa
lse, 'ir6': False}
☎ TRIGGERING CALL FOR: Station Three Satarkul
⚠ CallMeBot Busy (FLOOD). Switching to Text Backup...
📩 Backup Text Alert Sent!
LOG: Path=/ESP32_03 Data={'Sensor': 1, 'Shutdown': 1, 'flame_detected': False, 'ir1': False, 'ir2': False, 'ir3': False, 'ir4': False, 'ir5': F
alse, 'ir6': False}
```


CallMeBot_API8
online
January 25
This is a test from Callmebot 8:19 PM
Declined call
8:19 PM
Emergency Fire Detected at Station One 8:22 PM
Declined call
8:22 PM
Emergency Alert! Fire detected at Station Four Natunbazar.
8:38 PM
Declined call
8:38 PM
Emergency Alert! Fire detected at Station Three Satarkul. 8:53 PM
Incoming call
8:53 PM, 17 seconds
Emergency Alert! Fire detected at Station Three Satarkul. 8:57 PM


CallMeBot_API8
online
Emergency Alert! Fire detected at Station Four Natunbazar.
11:29 PM
Incoming call
11:29 PM
Emergency Alert! Fire detected at Station Four Natunbazar.
11:47 PM
Incoming call
11:48 PM, 11 seconds
Emergency Alert! Fire detected at Station Four Natunbazar.
11:49 PM
Incoming call
11:49 PM, 7 seconds
January 26
Emergency Alert! Fire detected at Station Three Satarkul.
1:37 PM
Incoming call
1:37 PM, 11 seconds
Write a message...

Discussion: By using the Firebase Realtime Database, the system achieved a latency of less than one second, which is essential for providing drivers with accurate, up-to-the-minute data on station availability.

A major success of the project is the automated emergency response. Unlike traditional stations that rely on manual alarms, this system uses a Python-based watchdog to trigger instant voice calls and Telegram alerts during a fire. This dual-channel approach ensures a reliable failsafe that can save lives and prevent property damage.While WiFi stability was an initial challenge, the implementation of robust auto-reconnect logic ensures the system stays online 24/7 without needing manual resets. Overall, this IOT system is more efficient for drivers and safer for station owners.

Conclusion :

This system has successfully provides a complete IoT solution for modern charging stations. By combining real-time sensor monitoring, cloud-based control, and automated emergency alerts, the project makes EV charging safer for owners and more convenient for drivers. The system’s ability to reduce wait times through live traffic updates and prevent disasters through instant fire notifications proves that smart technology is essential for the future of green transportation