Air Quality Monitoring System

Final Project

CS539: Internet of Things

Instructor: **Dr. Jagpreet Singh**

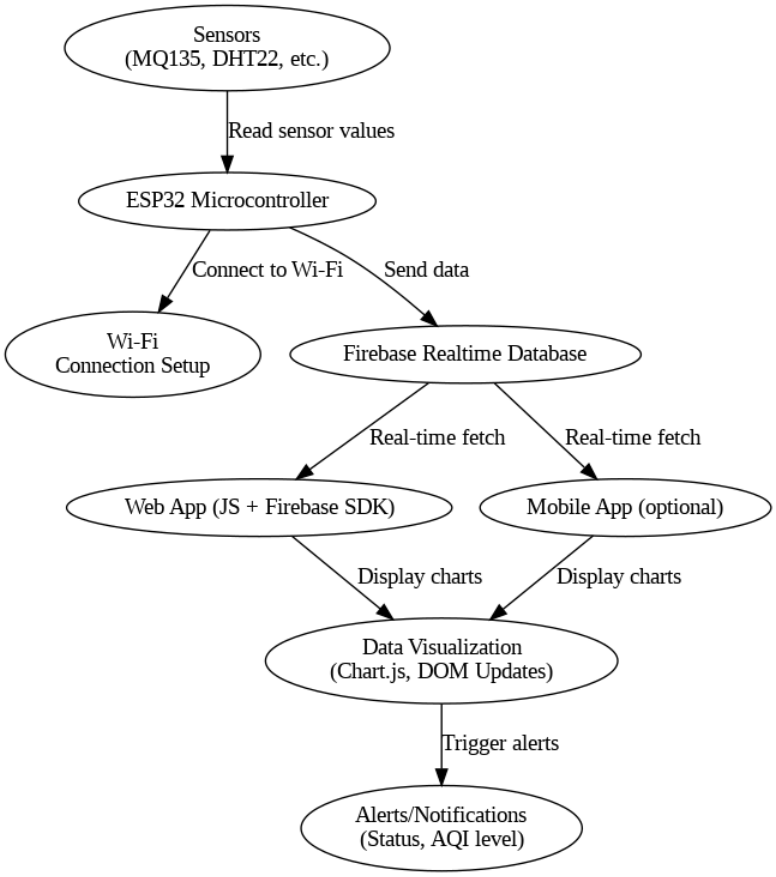
Name	**Entry No.**
Anshika	2021CSB1069
Ghulam Haider	2024CSM1008
Md Rizwan Ahmad	2024AIM1005

1 | Overview and System Architecture

This project implements an **ESP32-based Air Quality Monitoring System** that measures environmental parameters like **CO_2 and CH_4 (methane)**, along with **temperature and humidity**. The system is capable of:

- **Real-time data acquisition** via sensors.
- **WiFi-based connectivity** for cloud communication.
- **Web server** hosting on the ESP32 to display live data locally.
- **Firebase integration** for remote logging, analytics, and monitoring.
- **Time synchronization** and logging using the NTP protocol.

The architecture combines **embedded sensing**, **web technologies**, and **cloud-based database systems** to deliver a smart, connected IoT solution.



2 | Communication Protocols and Technologies Used

Protocol / Technology	Purpose
I²C / Analog	Sensor interfacing (CO_2, CH_4, DHT22)
WiFi (802.11 b/g/n)	Internet connectivity
HTTP	Web server access (GET request from browser)
NTP (Network Time Protocol)	Synchronizing system time
Firebase Realtime Database (REST API)	Cloud database read/write
JSON	Data format for cloud upload

3 | WiFi Functions – Connectivity Management

Purpose:

Handle all operations related to **connecting and maintaining** WiFi access for cloud and server features.

Key Functionalities:

- **Initial Connection**:
  - Attempts connection to a predefined SSID and password.
  - Displays IP address and signal strength once connected.
- **Reconnection Logic**:
  - Automatically checks if the ESP32 is disconnected.
  - If so, it tries to reconnect every 30 seconds.

  - On reconnection, it reinitializes Firebase.

- **Status Display**:
  - Converts numeric WiFi status codes to human-readable logs for diagnostics.

Significance:

Ensures the device remains **robust and self-healing** even under unstable network conditions. Cloud and web functions are dependent on this connectivity layer.

4 | Firebase Functions – Cloud Integration

Purpose:

Enable **cloud-based storage and analysis** of sensor data using Google Firebase's Realtime Database.

Key Functionalities:

- **Initialization (initFirebase)**:
  - Authenticates with Firebase.
  - Tests connection by writing an initial status.
  - Registers device metadata like MAC address, IP, and timestamps.
- **Structured Upload (uploadToFirebase)**:
  - Uploads sensor readings in a structured format under /data/MAC/timestamp.
  - Also updates the latest reading under /latest/MAC.
- **Qualitative Assessment (updateQualityStatus)**:

- Uploads status like "good", "moderate", or "poor" for both CO_2 and CH_4.
- Derives a combined overall air quality label.

Significance:

The cloud layer allows **remote monitoring**, **data logging**, and **future scalability** with analytics and visualization dashboards.

5 | Utility Functions – Support Layer

Purpose:

Provide **helper functions** that make sensor data more meaningful and manage timestamps.

Key Functionalities:

- **Classification Logic**:
  - getAirQualityClass and getMethaneLevelClass: Classify gas levels into labels (e.g., "good", "moderate", "poor").
  - Used in Firebase uploads and web display for semantic clarity.
- **Text Conversion**:
  - Converts numeric readings into human-readable formats like “Normal” or “High”.
- **Time Management**:
  - Functions like getFormattedTime() and getEpochTime() fetch current time from NTP and format it.
  - Used for timestamps in logs and Firebase.
- **Serial Logging**:

  - printLocalTime() displays the system time to the Serial Monitor for debugging.

Significance:

Bridges **raw sensor data with human context**, enhances UX, and supports time-synchronized operations.

6 | Experimental Setup

- **Hardware**:
  - ESP32 NodeMCU
  - MQ135 (for CO_2 and CH_4 approximation)
  - DHT22 (for temperature and humidity)
  - 5V regulated power supply
- **Software**:
  - Arduino IDE with Firebase ESP32 Library
  - Firebase Console for Database setup
  - Browser for local web interface
- **Network**:
  - 2.4 GHz WiFi
  - Static IP or mDNS for easier access (optional)

7 | Web + App Frontend

Key Technologies:

- **JavaScript (Vanilla):** For DOM manipulation and Firebase SDK.
- **Firebase SDK:** Reads real-time updates from the database.
- **Chart.js:** Dynamically plots environmental data like temperature, humidity, CO_2, CH_4.

UI Components:

- **Dashboard Panels:** Show real-time values and status for CO_2 and CH_4.
- **Node Cards:** Represent individual ESP32 nodes (Node A, Node B).
- **Date & Status Labels:** Show current time and sensor health.
- **Charts:**
  - envChart: Temperature and humidity per node.
  - historyChart: Historical trends for CO_2 and CH_4 with dual y-axes.

Data Fetch Logic:

- Firebase ref("/data") is queried to fetch real-time node data.
- On value change, UI updates:
  - DOM elements like co2ValueElement, aqiValueElement, etc.
  - Line charts are updated using `chart.update()`.

History & Comparison:

- Users can select **time ranges** (day/week/month).
- Can compare **Node A vs Node B** data visually.
- Optional: Add cloud functions or Firebase queries to enable filtering.

Android App

Technology Stack:

- Framework: React Native with Expo
- Platform: Android
- App Type: WebView wrapper (native shell that loads your web app)
- Web App Source: Your existing HTML/JS-based air quality monitoring dashboard hosted online (or locally bundled if needed)

Purpose of the App:

The app allows users to view real-time air quality sensor data directly from your Firebase-connected website, but now as a mobile app with:

- A native Android app feel (with icon, splash screen, APK)
- Fullscreen WebView loading your website
- Access via home screen like a regular app
- Real-time updates via Firebase (inherited from the web app)
- App installation on Android phones (.apk or .aab format)

Features Inherited from the Web App:

- Live air quality data (CO2, PM2.5, etc.)
- Real-time charts and dashboards
- Firebase-connected IoT sensor readings
- Interactive visuals (JavaScript charts, animations)
- Possibly multiple sensor nodes (as per your Firebase schema)

What’s Inside the App Code:

App.js includes:

- A full-screen WebView
- URL to your hosted dashboard (e.g., https://your-app.netlify.app)
- JavaScript and DOM support for interactive visuals
- Status bar handling for Android

Packaging Options:

- Development: Expo Go app via QR code
- Final Delivery: Standalone .apk file built via EAS CLI and installable on any Android phone
- Optionally publish to Google Play via the .aab file

Installation Method:

- APK version: installable directly from file (e.g., via USB or QR code)
- AAB version: suitable for Google Play Store submission