A MINI PROJECT REPORT

ON

IOT Based Air Quality Monitoring

BACHELOR OF TECHNOLOGY

In

Artificial Intelligence and Machine Learning

Submitted By

Ayush Bajpai (2201921640015)
Pranshu Sharma (2201921640035)
Aditya Pathak (2201921640005)

Under the Supervision of

Dr. Santosh Kumar Srivastava

G.L. BAJAJ INSTITUTE OF TECHNOLOGY & MANAGEMENT, GREATER NOIDA

Affiliated to
DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW

2024-2025

Certificate

This is to certify that the Mini Project report entitled **IoT-Based Real-Time Air Quality Monitoring and Alert System** done by **Ayush Bajpai (2201921640015), Aditya Pathak (2201921640005), and Pranshu Sharma (2201921640035)** is an original work carried out by them in Department of Computer Science and Engineering (AIML), G.L. Bajaj Institute of Technology & Management, Greater Noida under my supervision. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Date: 30th November 2024

Dr. Santosh Kumar Srivastava **Dean / Head of Department**
Associate Professor

Acknowledgement

The merciful guidance bestowed to us by the almighty made us stick out this project to a successful end. We humbly pray with sincere heart for his guidance to continue forever.

We pay thanks to our project guide **Dr Santosh Kumar Srivastava** who has given guidance and light to us during this project. His versatile knowledge has helped us in critical times during the span of this project.

We pay special thanks to our Head of the Department **Dr. Naresh Kumar** who has been always present as a support and helped us in all possible ways during this project.

We also take this opportunity to express our gratitude to all those people who have been directly and indirectly with us during the completion of the project.

We want to thank our friends who have always encouraged us during this project.

Pranshu Sharma
Aditya Pathak
Ayush Bajpai

Abstract

The rapid urbanization and industrialization of modern societies have led to an alarming rise in environmental pollution, particularly in urban areas, necessitating advanced, real-time monitoring solutions to mitigate its adverse effects. This project introduces an innovative IoT-based air quality monitoring system designed to provide accurate, comprehensive, and actionable pollution data at an affordable cost. By leveraging the capabilities of low-cost yet highly effective sensors, edge computing for real-time processing, and mobile technologies for seamless access, this system bridges critical gaps in traditional air quality monitoring frameworks. At its core, the system integrates ESP32 microcontrollers, which serve as the computational backbone, alongside multi-gas sensors capable of detecting a range of harmful pollutants with precision. The inclusion of cloud-based analytics enhances the system's ability to not only visualize pollutant levels but also deliver predictive insights, enabling proactive measures. This comprehensive approach aims to empower policymakers, researchers, and the general public with real-time data, facilitating informed decision-making to combat urban pollution effectively. By addressing limitations in scalability, cost, and accessibility found in existing monitoring solutions, this project represents a significant step forward in environmental monitoring technologies.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Problem Definition

Air pollution has become a pressing concern worldwide, with severe implications for public health and the environment. Monitoring air quality effectively and providing real-time data to individuals is essential for taking preventive measures. However, existing systems often lack affordability, accessibility, and user-friendly interfaces, making them unsuitable for widespread use, especially in urban and rural areas.

The key problems are:

1. Lack of real-time and accessible air quality monitoring for everyday users.
2. High cost and complexity of traditional air quality monitoring equipment.
3. Limited integration with IoT platforms for interactive user engagement and data visualization.
4. Difficulty in combining temperature, humidity, and air quality metrics in a single, compact, and portable system.

This project aims to bridge the gap by providing a cost-effective, IoT-based air quality monitoring system that delivers real-time data to a mobile app. It empowers users to make informed decisions to safeguard their health and contribute to environmental awareness.

1.2 Project Overview

The "IoT Based Air Quality Monitoring" system is designed to provide a cost-effective, real-time solution for monitoring air quality along with environmental parameters such as temperature and humidity. The system employs an ESP32 microcontroller as the central unit, interfaced with sensors like the MQ135 for air quality and DHT11 for temperature and humidity.

The collected data is processed by the ESP32 and relayed to a cloud platform via Arduino's IoT Cloud. Users can access this data through an intuitive Android application, created using Arduino's Cloud IDE and displayed using the Arduino IoT Remote app. The app features interactive widgets, such as meters and graphs, which provide a graphical representation of real-time and historical data, ensuring a user-friendly experience.

The system's core functionalities include:

1. **Real-Time Monitoring:** Continuous measurement of air quality, temperature, and humidity.
2. **Cloud Connectivity:** Data is stored and accessed through Arduino IoT Cloud.
3. **Mobile App Interface:** A dynamic GUI displaying environmental metrics in a visually appealing manner.
4. **Cost-Effectiveness:** Designed with affordable components to ensure accessibility.
5. **Scalability:** Capable of integrating additional sensors or features in the future.

This project serves as an effective tool for individuals and organizations to monitor environmental conditions, make informed decisions, and raise awareness about air quality issues.

1.3 Existing System

Currently, air quality monitoring is predominantly conducted using large-scale, sophisticated equipment deployed by governmental and private organizations. These systems, while accurate, are often characterized by several limitations:

1. **High Cost:** Traditional air quality monitoring systems are expensive and require significant investments, making them inaccessible for individual or small-scale use.
2. **Limited Coverage:** Fixed monitoring stations are typically limited to specific locations, leaving vast areas unmonitored, particularly rural or less-developed regions.
3. **Complex Operation:** These systems demand professional expertise for setup, calibration, and maintenance, making them impractical for general users.

4. **Lack of Real-Time Accessibility:** Data from conventional systems may not always be available in real-time, creating a delay in awareness and response.
5. **Inflexibility:** Traditional systems are not designed to integrate additional parameters such as temperature or humidity, which are crucial for comprehensive environmental monitoring.

Although some portable air quality monitors exist, they often compromise on accuracy, connectivity, or user interface. Furthermore, few systems offer seamless integration with IoT platforms or mobile applications, which limits their functionality for modern, tech-savvy users.

These gaps highlight the need for an affordable, user-friendly, and IoT-enabled system capable of providing real-time air quality data directly to users' devices.

1.4 Proposed System

The proposed system addresses the limitations of existing air quality monitoring systems by leveraging IoT technology and a user-friendly interface. It is a cost-effective and scalable solution designed for real-time monitoring and data visualization of environmental conditions.

Key features of the proposed system include:

1. **IoT-Enabled Monitoring:** The system uses the ESP32 microcontroller to collect data from sensors and transmit it to the cloud via Arduino IoT Cloud for real-time accessibility.
2. **Multi-Parameter Measurement:** The system combines the capabilities of the MQ135 sensor to measure air quality and the DHT11 sensor to monitor temperature and humidity, providing a comprehensive environmental overview.
3. **Mobile Application Integration:** Users can access the data through an Android app created using Arduino's IoT Cloud IDE. The app provides interactive widgets like AQI meters and temperature gauges for easy understanding of real-time data.
4. **Cost-Effectiveness:** By using affordable components like the ESP32, MQ135, and DHT11, the system remains budget-friendly and accessible to a wide range of users.
5. **User-Friendly Interface:** The Arduino IoT Remote app ensures that users can visualize and understand the data intuitively without technical expertise.

6. **Scalability:** The modular design allows for the addition of more sensors or integration with other IoT systems in the future.

1.5 Unique Features of the Proposed System

The proposed "IoT Based Air Quality Monitoring" system incorporates several unique features that set it apart from traditional and existing systems:

1. **Seamless IoT Integration:**
   The system leverages the ESP32 microcontroller and Arduino IoT Cloud to enable real-time data collection, storage, and access. This integration ensures continuous monitoring and accessibility from anywhere.
2. **Compact and Portable Design:**
   Unlike bulky traditional systems, the proposed solution is lightweight, compact, and easy to deploy in various locations, including homes, offices, and outdoor spaces.
3. **Comprehensive Environmental Monitoring:**
   By combining the MQ135 sensor for air quality and the DHT11 sensor for temperature and humidity, the system provides a holistic view of environmental conditions.
4. **Interactive Mobile Application:**
   The Android app created with Arduino's Cloud IDE and IoT Remote app enables users to visualize data in a user-friendly GUI with widgets such as AQI meters, temperature gauges, and real-time graphs.
5. **Cost-Effectiveness:**
   The use of affordable components ensures that the system is accessible to individuals, small businesses, and educational institutions, promoting widespread adoption.
6. **Customizable and Scalable:**
   The modular design allows easy integration of additional sensors or features, making it suitable for varied use cases and future upgrades.
7. **User-Friendly Interface:**
   The system is designed with non-technical users in mind, providing an intuitive interface

both in the hardware setup and mobile app.

8. **Real-Time Alerts and Updates:**
   The system can be programmed to send notifications or alerts to the user in case of hazardous air quality levels, enabling quick action.

These unique features make the proposed system an ideal solution for real-time air quality monitoring, blending functionality with affordability and ease of use.

Chapter 2

Requirement Analysis and System Specification

2.1 Introduction

Requirement analysis is a critical phase in the system development lifecycle, focusing on understanding the system's needs and defining its specifications. For the "IoT Based Air Quality Monitoring" system, requirements are identified based on the functionalities and performance goals essential for real-time environmental monitoring. This chapter outlines the functional, data, and performance requirements, the SDLC model adopted, and key diagrams illustrating the system's architecture.

2.2 Functional Requirements

The functional requirements define the primary operations the system must perform:

1. **Sensor Data Collection:**
   - The system must collect air quality data using the MQ135 sensor.
   - It must measure temperature and humidity using the DHT11 sensor.

2. **Data Processing and Transmission:**
   - The ESP32 microcontroller should process the data and send it to Arduino IoT Cloud.

3. **Mobile Application Integration:**
   - The Arduino IoT Remote app must display data through widgets (e.g., AQI meter, temperature gauge).

4. **Real-Time Updates:**
    - The system should provide real-time updates on environmental parameters.

5. **Alert Mechanism (Optional):**
    - Users may be notified when air quality crosses predefined hazardous levels.

2.3 Data Requirements

The data requirements specify the type and flow of information within the system:

1. **Input Data:**
    - Air quality levels from the MQ135 sensor.
    - Temperature and humidity readings from the DHT11 sensor.

2. **Output Data:**
    - Real-time environmental data displayed on the mobile app through widgets.

3. **Cloud Storage:**
    - The system must store collected data in Arduino IoT Cloud for future reference and analysis.

4. **Communication Protocols:**
    - Use of MQTT or HTTP protocols for seamless data transmission between the ESP32 and Arduino IoT Cloud.

2.4 Performance Requirements

1. **Accuracy:**
   - The system must provide accurate readings with minimal error from the sensors.

2. **Real-Time Processing:**
   - Data should be processed and transmitted to the mobile app within 1-2 seconds of collection.

3. **Reliability:**
   - The system should function consistently under various environmental conditions.

4. **Scalability:**
   - Must support additional sensors or expanded features without significant reconfiguration.

5. **Energy Efficiency:**
   - The ESP32 should operate with minimal power consumption, suitable for battery-powered setups.

2.5 SDLC Model to be Used

The project follows the **Iterative SDLC Model** due to its flexibility and adaptability:

1. **Requirement Gathering and Analysis:**
   - Initial requirements are identified, with scope for modifications in subsequent iterations.

2. **Design and Prototyping:**
   - A prototype is developed to validate functionality and design.

3. **Implementation:**
   - The final system is implemented and integrated.

4. **Testing and Evaluation:**
   - Iterative testing ensures reliability and performance optimization.

5. **Deployment and Maintenance:**
   - The system is deployed, with periodic updates for enhancements.

2.6 Use Case Diagram

The use case diagram depicts the interactions between the user and the system. Below is the description:

- **Actors:**
  1. User (App Interface)
  2. Sensors (MQ135, DHT11)
  3. Cloud Platform (Arduino IoT Cloud)

- **Use Cases:**
  1. Collect Environmental Data.
  2. Process and Send Data to Cloud.
  3. Display Data on Mobile App.
  4. Notify User of Critical Air Quality Levels.

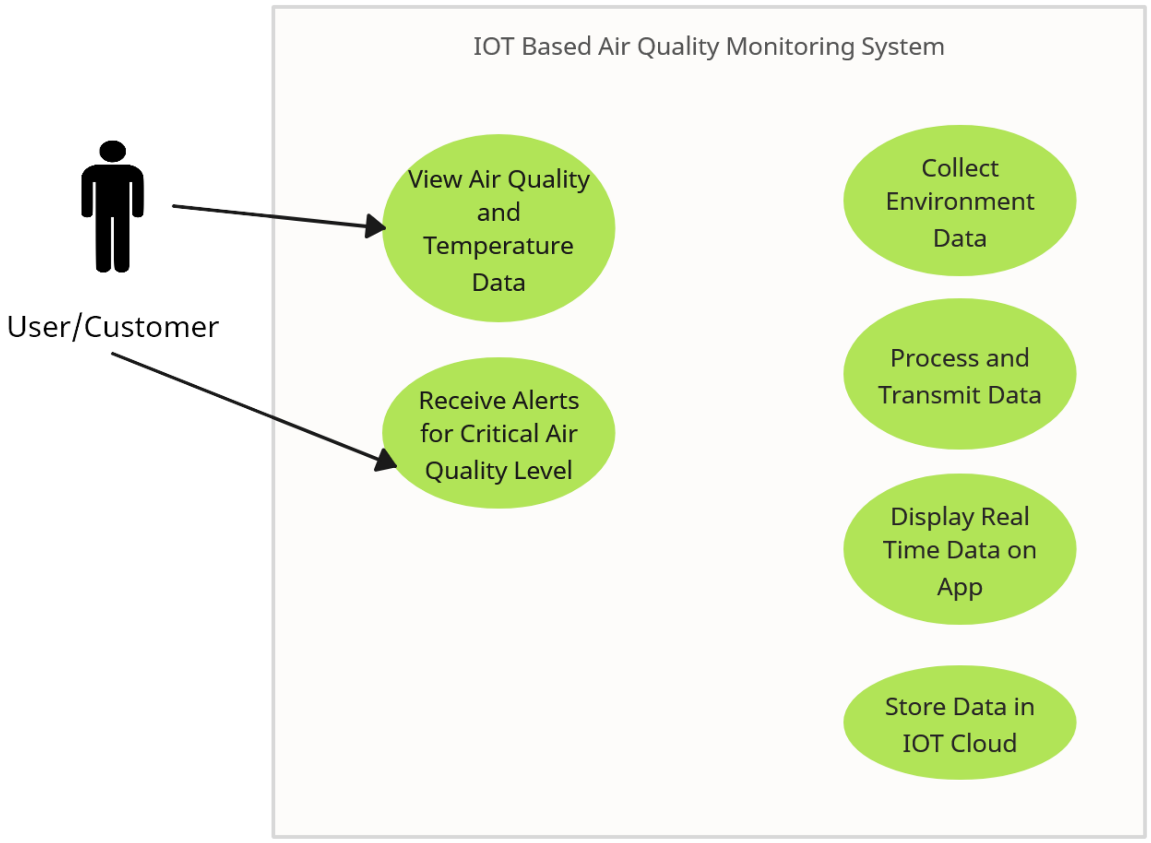

IOT Based Air Quality Monitoring System
User/Customer
View Air Quality and Temperature Data
Receive Alerts for Critical Air Quality Level
Collect Environment Data
Process and Transmit Data
Display Real Time Data on App
Store Data in IOT Cloud

Figure 2.1: Use Case Diagram of the System

Chapter 3

System Design

3.1 Introduction

The system design phase is a critical step in translating the identified requirements into a blueprint for development and implementation. For the "IoT Based Air Quality Monitoring" project, the design focuses on creating a seamless integration of hardware and software components to ensure efficient data collection, processing, transmission, and visualization. This chapter presents the design framework and methodologies used to develop a robust and scalable system.

The primary objective of system design is to define the architecture, components, modules, interfaces, and data flow of the system. This phase ensures that each system component interacts effectively with others to achieve the desired functionality. The design phase bridges the gap between theoretical requirements and practical implementation, providing a clear roadmap for development.

Key considerations in the design process include:

1. **Scalability:** The system is designed to accommodate future enhancements, such as integrating additional sensors or features.

2. **Reliability:** Each component is selected and configured to ensure consistent and accurate operation under varying environmental conditions.

3. **Modularity:** The design separates hardware and software functionalities into distinct modules, simplifying testing, debugging, and upgrades.

4. **Usability:** The system is user-centric, with an emphasis on a clear, intuitive interface for data visualization through the Android app.

5. **Cost-Effectiveness:** Affordable components and technologies are chosen without compromising performance.

6. **Real-Time Functionality:** The system design ensures minimal latency in collecting, processing, and displaying environmental data.

3.2 Design Approach

The design approach for the "IoT Based Air Quality Monitoring" system is based on a **function-oriented design methodology**. This methodology focuses on breaking down the system into smaller, manageable functional modules that perform specific tasks. Each module is designed to accomplish a well-defined function, ensuring that the system is modular, scalable, and easy to maintain.

3.2.1 Rationale for Function-Oriented Design

The function-oriented approach was chosen for the following reasons:

1. **Simplified Development:**
   The system's tasks, such as data collection, processing, and transmission, are mapped to individual functional modules, making development straightforward.

2. **Focus on Core Operations:**
   Each module targets a specific operation, such as sensor data acquisition, data processing by ESP32, and real-time data transmission to the cloud.

3. **Ease of Testing and Debugging:**
   Modular design simplifies the identification and rectification of issues within specific system components.

4. **Flexibility and Scalability:**
   Additional features, such as integrating more sensors or improving alert mechanisms, can be implemented without overhauling the entire system.

5. **Interoperability:**
   The use of standard communication protocols (e.g., MQTT) ensures compatibility between the system's components and with external platforms.

3.3 Design Diagrams

System design diagrams are essential to visualize the structure, functionality, and interactions within the system. For the "IoT Based Air Quality Monitoring" project, multiple diagrams are utilized to represent the architecture, data flow, and functionality of the system. These diagrams provide a comprehensive understanding of how the components work together to achieve the desired objectives.

3.3.1 System Architecture Diagram

The system architecture diagram provides an overview of the system's components and their interactions. The architecture is designed to achieve seamless communication between the hardware, cloud, and user interface.

Key Components of the Architecture:

1. **Sensors:**
   - The MQ135 sensor is responsible for detecting air pollutants and determining the

Air Quality Index (AQI).

    - The DHT11 sensor measures temperature and humidity to provide additional environmental data.

2. **Controller:**

    - The ESP32 microcontroller serves as the system's brain, collecting sensor data, processing it, and transmitting it to the cloud.

3. **Cloud Platform:**

    - Arduino IoT Cloud stores and manages the data sent by the ESP32. It provides an interface for the user to access the data through the Arduino IoT Remote app.

4. **Mobile Application:**

    - The Android app (Arduino IoT Remote) fetches the data from the cloud and displays it in a user-friendly format using widgets like meters and gauges.

Process Flow in the Architecture:

- The sensors continuously collect environmental data.
- The ESP32 processes the raw data from the sensors and transmits it to Arduino IoT Cloud over Wi-Fi.
- The mobile app retrieves this data from the cloud in real-time and displays it through various graphical widgets.

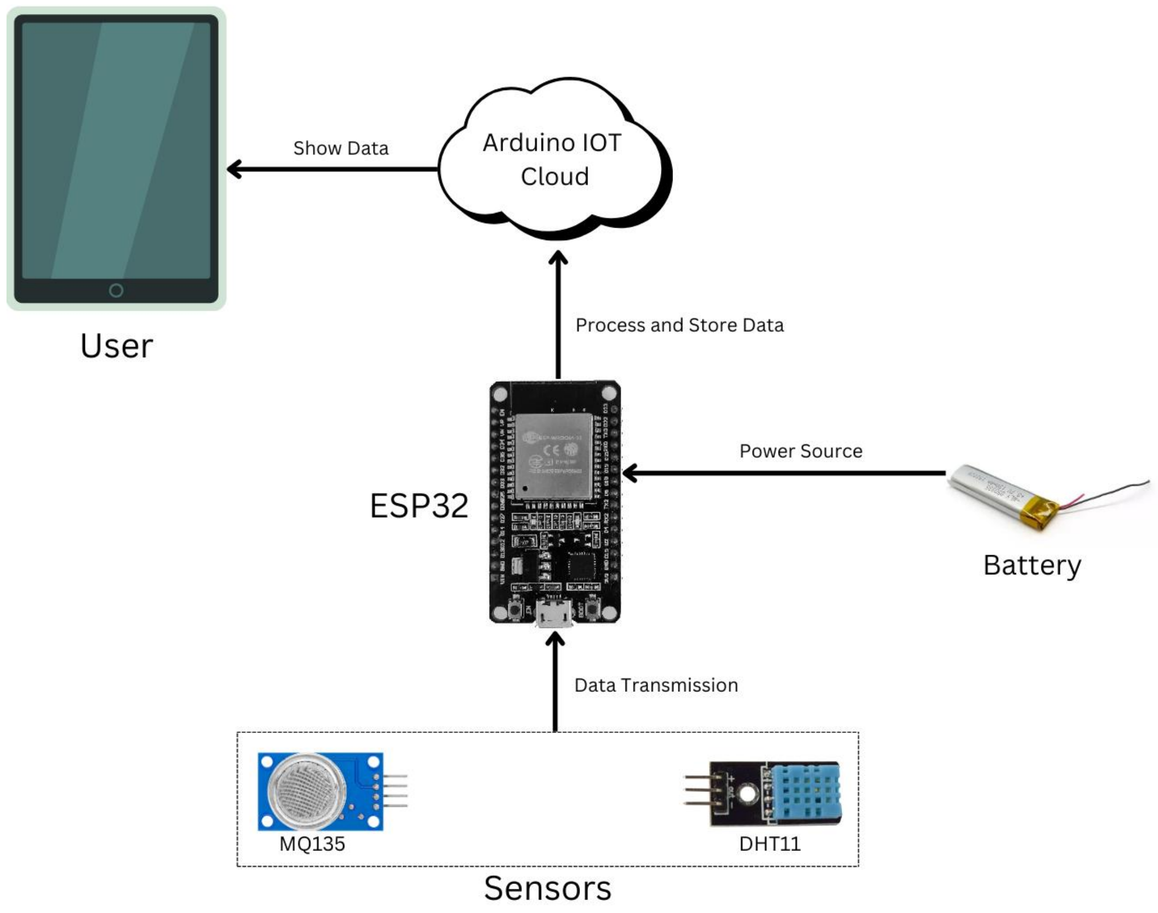


Figure 3.1: System Architecture Diagram of the System

3.3.2 Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) models the flow of data through the system, focusing on how data is collected, processed, transmitted, and visualized. A Level 1 DFD for this system includes the following elements:

1. **Data Sources:**

   The MQ135 and DHT11 sensors act as the data sources, providing air quality, temperature, and humidity data.

2. **Data Processing:**

   The ESP32 processes the raw data from the sensors, applying calibration and formatting it into a structured format suitable for transmission.

3. **Data Transmission:**

   The processed data is transmitted from the ESP32 to Arduino IoT Cloud using the MQTT protocol.

4. **Data Storage:**

   Arduino IoT Cloud stores the data in real-time and makes it accessible for retrieval by the mobile app.

5. **Data Visualization:**

   The mobile app displays the retrieved data using widgets such as AQI meters and temperature/humidity gauges.

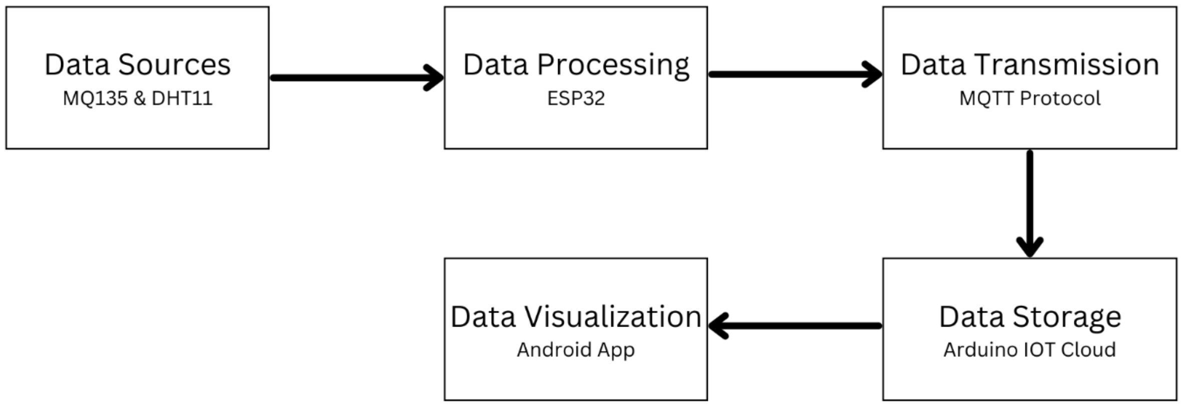


Figure 3.2: Data Flow Diagram of the System

3.3.3 Deployment Diagram

The deployment diagram represents the physical deployment of the system, highlighting hardware placement and communication channels.

Deployment Setup:

- Sensors are installed in the environment where air quality needs to be monitored.

- The ESP32 is placed near the sensors to ensure reliable data collection.

- Wi-Fi connectivity is established between the ESP32 and the cloud.

- The mobile app is deployed on the user's smartphone for data access.

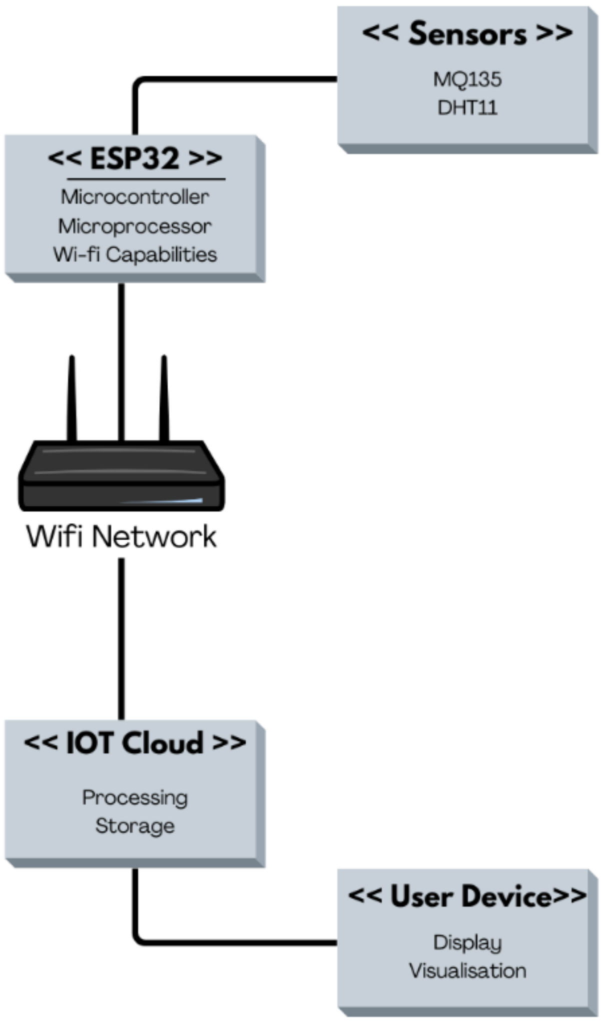


Figure 3.3: Deployment Setup of the System

3.3.4 Wiring Diagram

The wiring diagram illustrates the connections between the ESP32 microcontroller, the MQ135 air quality sensor, and the DHT11 temperature and humidity sensor. It provides a clear representation of how the components are integrated to ensure accurate data collection.

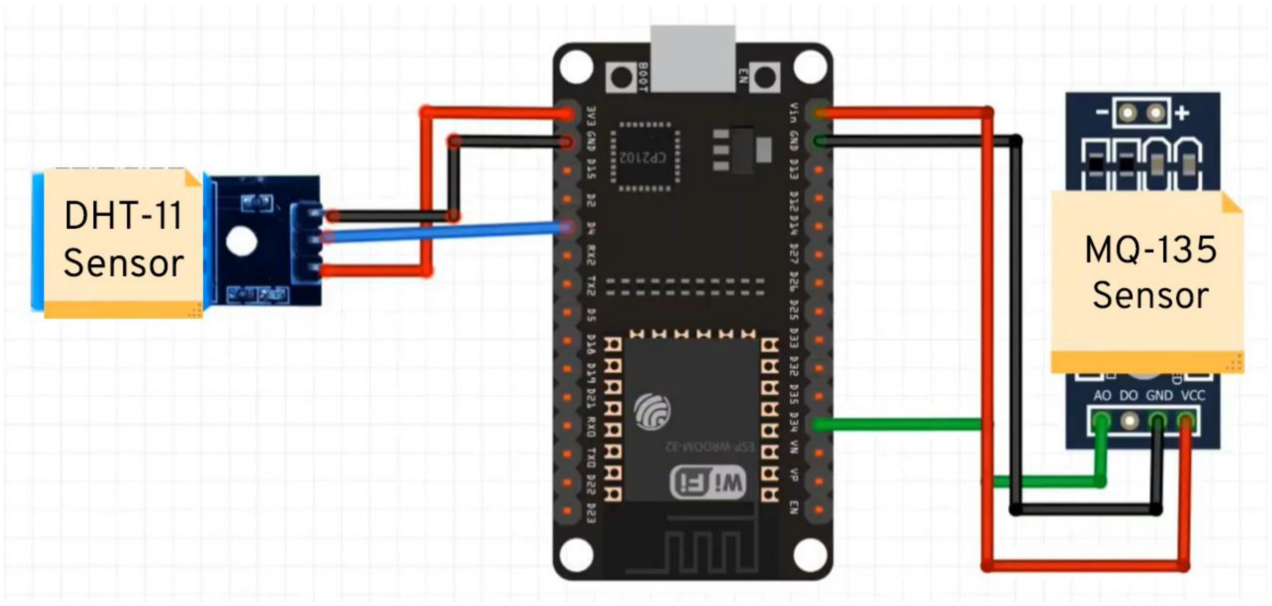

DHT-11 Sensor
MQ-135 Sensor
BOOT
EN
CP2102
ESP-WROOM-32
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
Vin GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
AO DO GND VCC

Figure 3.4: Wiring Diagram of the System

3.4 User Interface Design

The user interface (UI) design focuses on creating an intuitive and user-friendly platform for users to interact with the "IoT Based Air Quality Monitoring" system. The UI is primarily implemented using the **Arduino IoT Remote app**, which communicates with the Arduino IoT Cloud to fetch and display real-time and historical data.

The UI design ensures that users can easily monitor air quality, temperature, and humidity parameters through graphical widgets and visual representations. This section elaborates on the design of the Android app's interface and its key features.

3.4.1 Objectives of UI Design

The UI design for this project aims to achieve the following objectives:

1. **Simplicity:** Create an interface that is easy to navigate, even for non-technical users.
2. **Clarity:** Present sensor data in a clear and visually appealing manner.
3. **Responsiveness:** Ensure the app updates the displayed data in real-time with minimal latency.
4. **Accessibility:** Design the UI to be accessible across different screen sizes and devices.
5. **User Engagement:** Provide visual elements like graphs, meters, and notifications to keep users engaged.

3.4.2 UI Features and Layout

The UI is divided into several sections, each tailored to meet a specific user requirement.

1. Dashboard:

- Displays real-time data fetched from the Arduino IoT Cloud.
- Key elements include:
  - **Air Quality Index (AQI):** A meter or gauge that visually represents the AQI level.
  - **Temperature Display:** A digital gauge or numeric widget showing the current temperature in °C.
  - **Humidity Display:** A bar or circular gauge indicating the current humidity percentage.

2. Historical Data Section:

- Displays graphs and charts for past air quality, temperature, and humidity readings.
- Users can select a time range (e.g., last hour, day, or week) to view trends and patterns.

3. Alerts and Notifications (Optional):

- Sends push notifications to the user when air quality exceeds predefined thresholds.
- Alerts are displayed in a dedicated section within the app.

4. Settings Menu:

- Allows users to configure preferences such as:
    - Threshold levels for AQI alerts.
    - Update intervals for real-time data.
    - Connectivity settings for the ESP32 and the cloud.

3.4.3 Widgets Used

The UI uses interactive widgets provided by the Arduino IoT Remote app to display data effectively. Each widget is linked to a variable in the ESP32 code, which communicates with the Arduino IoT Cloud.

Key Widgets:

1. **Gauge Widget for AQI:**
    - Displays air quality levels in a range (e.g., 0 to 500).
    - Uses colour coding (green, yellow, red) to indicate safe, moderate, or hazardous conditions.

2. **Numeric Widget for Temperature:**
    - Shows the current temperature in real-time.
    - Units are displayed in °C for clarity.

3. **Bar Widget for Humidity:**
    - Displays the percentage of humidity.

    - Provides a visual representation of humidity levels.

4. **Graph Widget for Historical Data:**
    - Plots a line or bar graph showing trends for AQI, temperature, and humidity over time.

3.5 Database Design

The database design is an essential component of the "IoT Based Air Quality Monitoring" system, as it enables the structured storage and retrieval of sensor data. In this project, the database is managed via the **Arduino IoT Cloud**, which acts as the backend for storing real-time and historical data transmitted by the ESP32 microcontroller. This section discusses the structure, organization, and design considerations for the database.

3.5.1 Objectives of Database Design

The database design aims to:

1. **Ensure Data Integrity:** Accurate and consistent storage of air quality, temperature, and humidity data.

2. **Support Real-Time Updates:** Handle continuous updates from the ESP32 microcontroller without delays.
3. **Facilitate Data Retrieval:** Enable efficient access to real-time and historical data for visualization on the mobile app.

4. **Provide Scalability:** Allow for the addition of new variables, such as more sensors or metrics, without significant redesign.

5. **Ensure Security:** Protect user data and sensor readings during storage and transmission.

3.5.2 Database Model

The database uses a **cloud-based structured data model** hosted on the Arduino IoT Cloud. Data is organized in the form of **linked variables**, which represent sensor outputs and other metrics.

Key Variables Stored in the Database:

1. **Air Quality Index (AQI):**
    - Data Type: Float
    - Range: 0–500
    - Description: Represents the AQI level calculated from the MQ135 sensor readings.
2. **Temperature:**
    - Data Type: Float
    - Unit: Degrees Celsius (°C)
    - Description: Stores the ambient temperature data collected from the DHT11 sensor.
3. **Humidity:**
    - Data Type: Float
    - Unit: Percentage (%)
    - Description: Stores humidity data from the DHT11 sensor.
4. **Timestamps:**
    - Data Type: DateTime
    - Description: Indicates the exact time each data entry is recorded, enabling historical analysis.

3.5.3 Database Schema Design

Although Arduino IoT Cloud abstracts much of the database complexity, the logical schema can be visualized as follows:

Variable Name	Data Type	Description	Example Value
AQI	Float	Air Quality Index	120.5
Temperature	Float	Ambient Temperature (°C)	25.3
Humidity	Float	Relative Humidity (%)	65.0
Timestamp	DateTime	Record Timestamp	2024-12-03 14:32

Table 3.1: Database Schema of IOT Cloud

3.5.7 Limitations and Future Enhancements

1. **Storage Limitations:**
   - Arduino IoT Cloud may impose limits on the amount of historical data stored, requiring periodic backups or migration to external databases for long-term storage.
2. **Customization Constraints:**
   - Customizing database structures beyond Arduino IoT Cloud's predefined system might require third-party platforms.

Future Enhancements:

- Integration with a dedicated database, such as Firebase or AWS, for enhanced data storage and processing capabilities.
- Implementation of advanced data analytics for predictive insights.

Chapter 4

Implementation

I

4.1 Introduction

The implementation phase is crucial for translating the theoretical design and concepts into a working system. For the "IoT Based Air Quality Monitoring" project, the primary goal during implementation was to integrate various hardware components, such as the ESP32 microcontroller, MQ135 air quality sensor, and DHT11 temperature and humidity sensor, with the Arduino IoT Cloud to collect and transmit real-time data.

This focuses on the key aspects of the implementation, including the setup of the hardware, the development of software for the ESP32, and the integration of the system with the Arduino IoT Cloud for seamless communication and data visualization. The process also involves configuring the Android application to display the data in real-time, allowing users to monitor air quality metrics such as AQI, temperature, and humidity through a user-friendly interface.

4.2 Tools/Technologies Used

The implementation of the "IoT Based Air Quality Monitoring" project involves a combination of hardware components, software tools, and cloud technologies that enable real-time data acquisition, processing, and visualization. Below is a breakdown of the key tools and technologies utilized throughout the project.

4.2.1 Hardware Components

1. **ESP32 Microcontroller:**
    - **Role:** The ESP32 is the central microcontroller used for processing the data from the sensors and communicating with the cloud.
    - **Features:**
        - Dual-core processor for efficient handling of tasks.
        - Built-in Wi-Fi and Bluetooth connectivity for seamless data transmission.
        - Low power consumption, suitable for IoT applications.

2. **MQ135 Gas Sensor:**
    - **Role:** The MQ135 sensor is used to detect air quality by measuring the concentration of various gases in the environment, including ammonia (NH3), benzene (C6H6), and CO2.
    - **Features:**
        - Analog output for the detection of gas concentration.
        - Sensitivity to a wide range of gases that affect air quality, including pollutants and harmful compounds.
        - Data is processed to generate the Air Quality Index (AQI).

3. **DHT11 Temperature and Humidity Sensor:**
    - **Role:** The DHT11 sensor is used to measure the temperature and humidity of the surrounding environment.
    - **Features:**
        - Digital output for accurate temperature and humidity readings.
        - Affordable and widely used in IoT applications.
        - Operating range: Temperature: 0°C to 50°C, Humidity: 20% to 90% RH.

4. **Power Supply:**
    - **Role:** Ensures that the system components receive the necessary power to operate. The ESP32 requires a 5V input, while the sensors operate on 3.3V and 5V, respectively.
    - **Components:**
        - USB power adapter or battery to power the ESP32 and sensors.
        - Voltage regulator for stable voltage supply to components.

4.2.2 Software Tools

1. **Arduino IDE:**
    - **Role:** The Arduino IDE is used for writing and uploading the code to the ESP32 microcontroller.
    - **Features:**
        - An easy-to-use environment for developing, testing, and deploying code.
        - Extensive libraries for sensor integration and cloud communication.
        - Cross-platform support for Windows, macOS, and Linux.

2. **Arduino IoT Cloud:**
    - **Role:** The Arduino IoT Cloud acts as the backend for the system, storing the sensor data and providing a platform for real-time monitoring.
    - **Features:**
        - Cloud-based data storage for easy access and management of data.
        - Secure and scalable solution for IoT applications.
        - Integration with various devices through MQTT or HTTP protocols.
        - Easy-to-use dashboard for visualizing sensor data and creating cloud variables.

3. **Arduino IoT Remote App:**
    - **Role:** The Arduino IoT Remote app enables real-time visualization of sensor data on a mobile device.
    - **Features:**
        - Customizable widgets for displaying data from the cloud, such as AQI, temperature, and humidity meters.
        - Real-time updates from the Arduino IoT Cloud.
        - Cross-platform support (Android and iOS).

4. **Libraries and APIs Used:**
    - **WiFi.h:** Manages the Wi-Fi connection between the ESP32 and the internet.
    - **ArduinoIoTCloud.h:** Facilitates communication between the ESP32 and the Arduino IoT Cloud.
    - **Adafruit_Sensor.h:** Provides an interface for interacting with the DHT11 sensor.
    - **MQ135.h:** Provides a simplified method for reading values from the MQ135 sensor.

5. **MQTT Protocol:**
    - **Role:** MQTT (Message Queuing Telemetry Transport) is used for lightweight, low-latency communication between the ESP32 and the Arduino IoT Cloud.
    - **Features:**
        - Efficient data transmission, ideal for IoT applications.
        - Supports real-time communication with minimal data overhead.
        - Ensures reliable message delivery even in low-bandwidth environments.

4.2.3 Development Environment

1. **Cloud Integration:**
    - Arduino IoT Cloud provides a cloud platform to store and access sensor data. The system is configured to automatically sync data from the ESP32 to the cloud, ensuring that the data is available for monitoring at any time.

    - **Dashboard:** The IoT Cloud dashboard allows users to create custom widgets for visualizing the data, such as AQI gauges, temperature graphs, and humidity meters.

2. **Mobile Application:**
    - The Arduino IoT Remote app is used to display real-time sensor data. This app is available on both Android and iOS platforms, making it accessible to a wide range of users.

    - **Widgets:** Users can create widgets for monitoring various metrics like AQI, temperature, and humidity.

4.2.4 Communication and Data Flow

1. **ESP32 and Sensor Communication:**
    - The ESP32 is connected to the sensors (MQ135 and DHT11) to read the air quality, temperature, and humidity values.
    - The data is processed and formatted to be transmitted to the Arduino IoT Cloud.
2. **Cloud Communication:**
    - The ESP32 communicates with the Arduino IoT Cloud using the MQTT protocol.

This allows for real-time updates, ensuring that the data displayed in the mobile app is up-to-date.

- The Arduino IoT Cloud securely stores the data and allows users to access it through the app or web interface.

3. **Real-time Data Visualization:**
    - The Arduino IoT Remote app continuously fetches the latest data from the cloud, updating the user interface in real-time. Users can monitor air quality, temperature, and humidity from anywhere with an internet connection.

4.3 Coding Standards of the Programming Language Used

In the development of the "IoT Based Air Quality Monitoring" project, adhering to consistent coding standards was essential for ensuring readability, maintainability, and efficiency. The code was written primarily in C/C++ using the Arduino IDE for the ESP32 microcontroller. Following standardized coding practices throughout the project allowed for easier debugging, modification, and future updates.

4.3.1 Modular Programming

Modular programming divides the program into distinct sections or modules, each responsible for a specific task. This helps in managing the complexity of the project and improves the reusability of code.

- **Sensor Modules:** Separate functions or files were created for handling each sensor (e.g., MQ135, DHT11). This allowed for a clean separation of concerns and made it easier to modify or replace individual sensors without affecting other parts of the code.
- **Cloud Communication Module:** A dedicated section of the code was responsible for handling communication between the ESP32 and the Arduino IoT Cloud, ensuring that the data from the sensors was transmitted in real time.

4.3.2 Variable Naming Conventions

Proper naming conventions improve code readability and make it easier to understand the program’s functionality at a glance. The following naming conventions were adhered to:

- **Descriptive Variable Names:** Variable names were chosen to clearly describe their function.
  - Example:
    - airQualityValue for storing the air quality index (AQI) value.
    - temperatureReading for storing the temperature data from the DHT11 sensor.
    - humidityLevel for storing the humidity percentage.

- **Consistent Naming Style:** Camel case (camelCase) was used for variable names, where the first word is lowercase, and subsequent words are capitalized (e.g., sensorData, sensorValue).

4.3.3 Comments and Documentation

Clear comments and documentation were included in the code to explain the logic behind each part of the program. This is particularly important for future developers or for anyone reviewing the code.

- **Inline Comments:** Short comments were added next to lines of code explaining specific actions or logic.
- **Function Documentation:** Each function in the code included a brief description of its purpose, input parameters, and output.

4.3.4 Error Handling and Exception Management

Error handling ensures that the program can deal with unexpected situations gracefully without crashing. Robust error handling mechanisms were put in place to account for potential failures in sensor readings, connectivity issues, or cloud communication problems.

- **Sensor Error Handling:** The program checks if the sensors return valid data. For example, if the DHT11 sensor returns a "NaN" (Not a Number) value, the program will handle it by retrying or logging an error.

- **Wi-Fi Connectivity:** The program checks the Wi-Fi connection status and retries if the ESP32 fails to connect to the network.

- **Cloud Communication:** If the connection to the Arduino IoT Cloud fails, the program will attempt to reconnect and print an error message.

4.3.5 Code Structure and Formatting

Maintaining consistent formatting throughout the code enhances readability and makes collaboration easier. The following practices were followed:

- **Indentation:** Proper indentation was used to structure code blocks, making it clear which parts of the code belong together. A consistent indentation level (2 spaces) was maintained.

- **Braces:** Opening and closing braces {} were used consistently to define code blocks. Even in single line if statements, braces were used to maintain clarity.

Chapter 5

Result & Discussion

I

5.1 Introduction

In this chapter, we present the results and discussions derived from the implementation of the "IoT Based Air Quality Monitoring" system. The goal of this project was to develop an integrated solution for real-time air quality monitoring using IoT technology. The system utilizes an ESP32 microcontroller, an MQ135 air quality sensor, and a DHT11 sensor for temperature and humidity measurement. Data is relayed to the Arduino IoT Cloud and visualized through the Arduino IoT Remote app, providing users with real-time information on the environmental conditions.

The performance of the system was evaluated by examining its functionality, accuracy, data transmission reliability, and user experience. In this section, we will showcase the key results, including the successful integration of sensors, cloud communication, and mobile application visualization. Additionally, the effectiveness of the system in providing accurate and timely air quality data will be discussed, along with any challenges encountered during the implementation.

5.2 Snapshots of System

In this section, we present visual snapshots of the IoT-based air quality monitoring system, including the sensor setup, data visualization on the Arduino IoT Cloud dashboard, and the mobile application interface. These images provide a clear view of how the system functions in real-time and how users can interact with the data through the mobile app.

5.2.1 Sensor Setup

The sensor setup involves the ESP32, MQ135 air quality sensor, and DHT11 temperature and humidity sensor. Below is an image of the physical setup of the system, which includes the sensors connected to the ESP32 on a breadboard.

Snapshot of the hardware setup:

- The ESP32 is positioned centrally, with the MQ135 sensor for air quality placed on one side and the DHT11 sensor for temperature and humidity placed next to it.
- The system is powered via a USB adapter, with wires connecting the sensors to the corresponding pins on the ESP32.

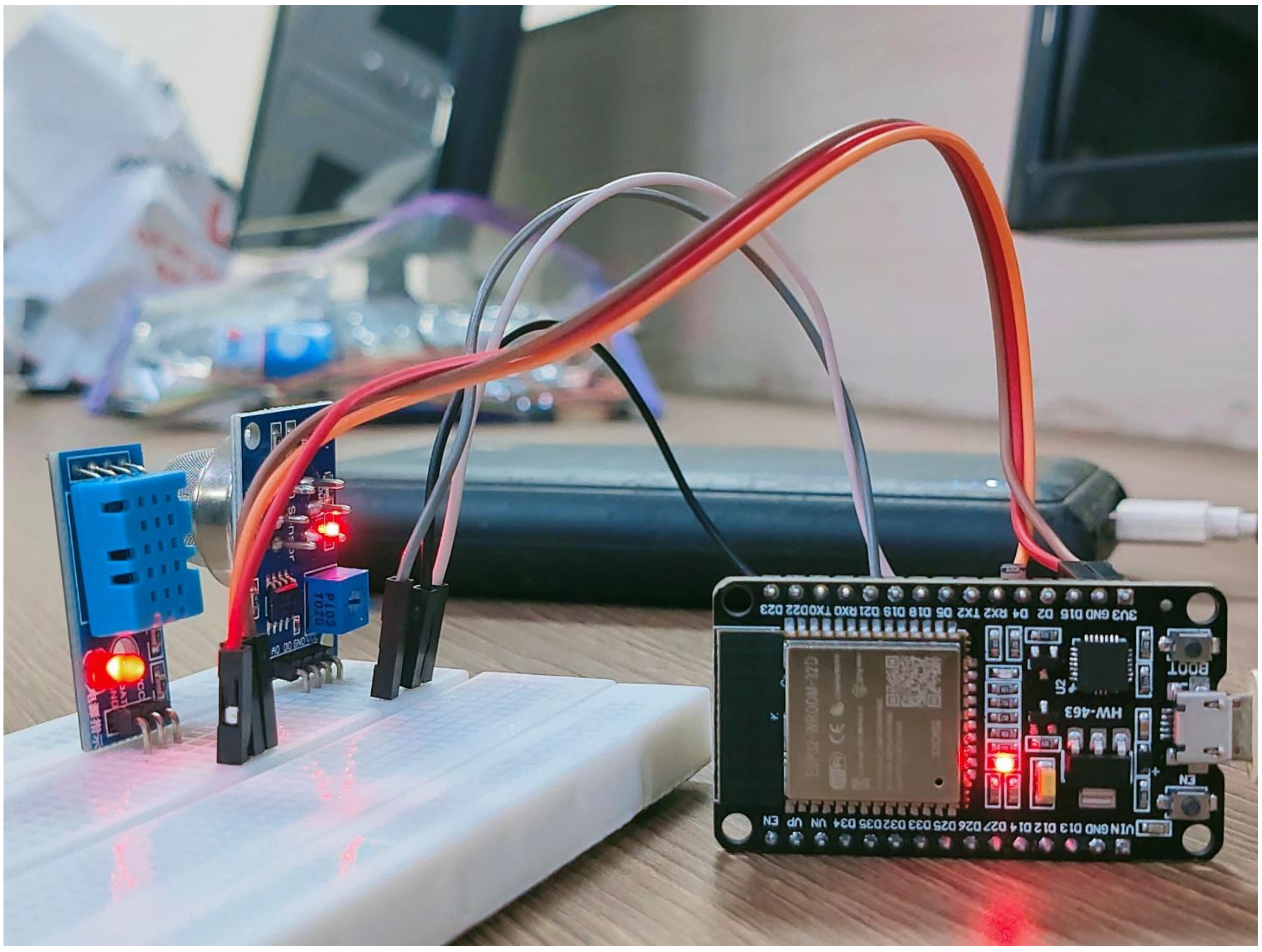


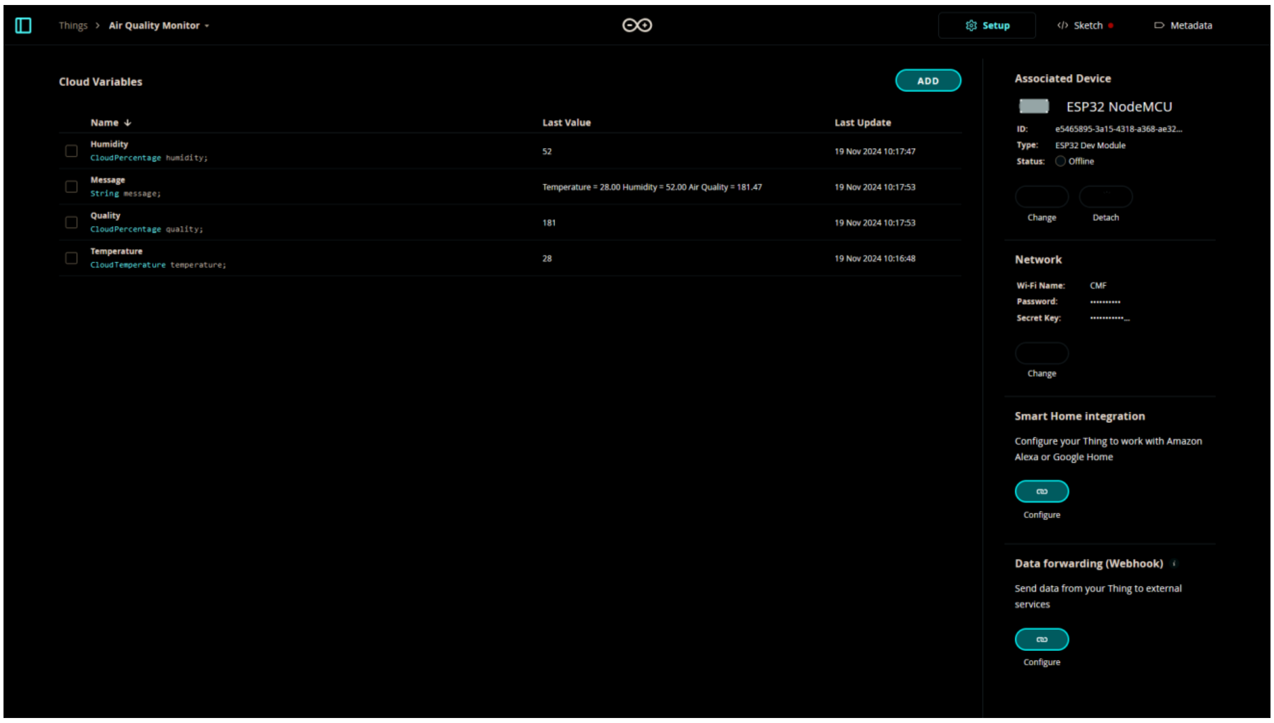
Figure 5.1: Snapshot of the Hardware Setup

5.2.2 Arduino IoT Cloud Dashboard

The Arduino IoT Cloud provides a web-based dashboard that displays the real-time sensor data. The cloud dashboard updates periodically to reflect the values for air quality, temperature, and humidity, which are read from the sensors. Users can access this dashboard to view the status of the environment remotely.

Snapshot of the Arduino IoT Cloud Dashboard:

- The dashboard contains various widgets representing different variables, such as:
  - **AQI Meter:** Displays the real-time Air Quality Index (AQI) based on the data from the MQ135 sensor.
  - **Temperature Meter:** Shows the temperature data retrieved from the DHT11 sensor.
  - **Humidity Meter:** Visualizes the humidity levels in the environment.
- The data is updated automatically, and the widgets reflect changes as the system collects fresh sensor readings.



Cloud Variables

Name	Last Value	Last Update
Humidity `CloudPercentage humidity;`	52	19 Nov 2024 10:17:47
Message `String message;`	Temperature = 28.00 Humidity = 52.00 Air Quality = 181.47	19 Nov 2024 10:17:53
Quality `CloudPercentage quality;`	181	19 Nov 2024 10:17:53
Temperature `CloudTemperature temperature;`	28	19 Nov 2024 10:16:48

Associated Device

ESP32 NodeMCU

ID:	e5465895-3a15-4318-a368-ae32...
Type:	ESP32 Dev Module
Status:	Offline

Network

Wi-Fi Name:	CMF
Password:	••••••••••
Secret Key:	•••••••••••...

Smart Home integration

Configure your Thing to work with Amazon Alexa or Google Home

Data forwarding (Webhook)

Send data from your Thing to external services

Figure 5.2: Snapshot of the Arduino IOT Cloud Dashboard

5.2.3 Mobile Application Interface

The Arduino IoT Remote app provides an easy-to-use mobile interface for monitoring real-time data. The app connects to the Arduino IoT Cloud to retrieve data from the ESP32, which is then displayed on various widgets designed for visual appeal and ease of interpretation.

Snapshot of the Mobile Application Interface:

- The main screen of the app shows several widgets representing sensor data.
    - **AQI Widget:** A gauge or meter that visually indicates the air quality index.
    - **Temperature and Humidity Widgets:** Meters showing the current temperature and humidity levels.
- The mobile app also allows for real-time updates, ensuring that the data is current and providing users with an interactive way to monitor environmental conditions from their mobile devices.

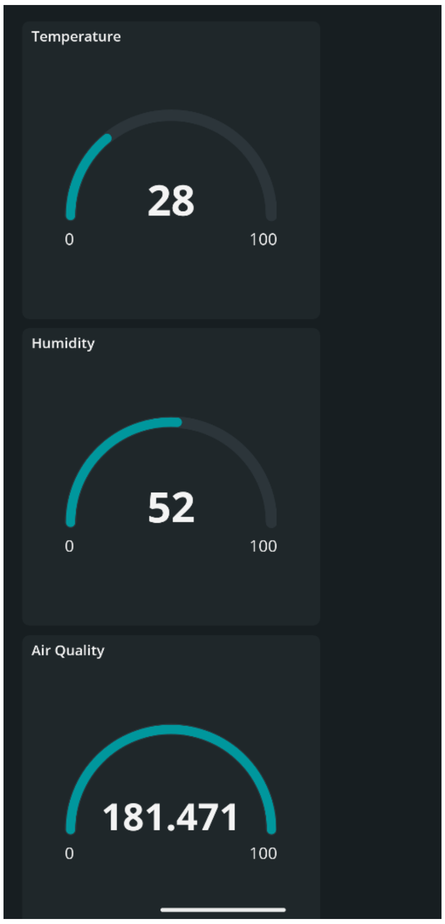


Figure 5.3: Snapshot of the Mobile Application Interface

5.3 Snapshots of Database Tables

Focus on the data stored in the Arduino IoT Cloud and its structure. The database holds the sensor data, including air quality (AQI), temperature, and humidity, which is logged continuously in real-time. The data is stored and can be accessed remotely for analysis, monitoring, and future use.

Since the Arduino IoT Cloud does not use a traditional relational database, it operates using cloud variables that are automatically created and updated with each sensor reading. However, these cloud variables serve as placeholders for database records, where data is stored in a format similar to database tables.

5.3.1 Cloud Variables Representation

In the Arduino IoT Cloud, the following cloud variables are created to represent the sensor readings:

1. **Air Quality Index (AQI) Variable:**
   - This variable stores the air quality index, calculated from the readings of the MQ135 sensor. It reflects the level of pollutants in the air and is continuously updated.

Cloud Variable: airQualityValue
Data Type: float
Range: 0 - 500 (AQI scale)

2. **Temperature Variable:**
   - The temperature value is retrieved from the DHT11 sensor. This variable stores the temperature in degrees Celsius.

Cloud Variable: temperature
Data Type: float
Range: 0°C - 50°C

3. **Humidity Variable:**
   - The humidity value is also retrieved from the DHT11 sensor. This variable stores the percentage of humidity in the environment.

Cloud Variable: humidity

Data Type: float

Range: 20% - 90% (Relative Humidity)

These cloud variables act as real-time placeholders for the sensor data that is being updated. Though not implemented in a traditional SQL database, the functionality provided by the Arduino IoT Cloud is equivalent to database records being created and updated for each variable.

5.3.3 Example of Cloud Data Table

The following is an example of what the data might look like in a traditional database table format:

Timestamp	AQI Value	Temperature (°C)	Humidity (%)
2024-12-01 10:00:00	72	22.5	58
2024-12-01 10:05:00	75	22.7	57
2024-12-01 10:10:00	80	23.0	55
2024-12-01 10:15:00	70	22.8	56

Table 5.1: Cloud Data Table Depicted as Traditional Database Table

This table represents how data would appear if logged in a conventional database system. In the Arduino IoT Cloud, this data is automatically captured in the form of cloud variables, and users can access it in real-time or for historical analysis.

Chapter 6

Conclusion, Limitation & Future Scope

I

6.1 Conclusion

The "IoT Based Air Quality Monitoring" project successfully integrates Internet of Things (IoT) technology with air quality monitoring. By using an ESP32 microcontroller, the MQ135 air quality sensor, and the DHT11 sensor, the system measures critical environmental parameters such as air quality (AQI), temperature, and humidity. This data is then transmitted to the Arduino IoT Cloud, where it is visualized in real-time through the Arduino IoT Remote app, providing users with immediate access to crucial information.

The key achievements of the project are as follows:

- **Real-time Data Monitoring:** The system provides continuous, real-time monitoring of environmental parameters. The data is instantly updated and made available to users through the IoT Cloud and mobile app.

- **User-Friendly Interface:** The use of the Arduino IoT Remote app ensures that even non-technical users can easily monitor air quality, temperature, and humidity with a simple, interactive interface.

- **Cloud Integration:** The integration with the Arduino IoT Cloud enables efficient storage, retrieval, and analysis of environmental data. It ensures that the system is scalable and can be expanded for additional sensors or functionality.

- **Cost-Effective and Scalable Solution:** The system uses low-cost components like the ESP32 and DHT11 sensor, making it affordable. The modular design allows the system to be easily extended with more sensors or features in the future.

In conclusion, this project demonstrates the potential of IoT in environmental monitoring, offering an efficient, accessible, and scalable solution for real-time air quality monitoring.

6.2 Limitations

While the system is functional and effective for real-time air quality monitoring, there are several limitations that should be addressed in future iterations:

1. **Sensor Accuracy:**
    - The MQ135 air quality sensor, though widely used in IoT applications, has a limited accuracy range. It may not provide highly precise measurements, especially for low or very high concentrations of pollutants. Calibration and more accurate sensors could improve the data's reliability.
    - Similarly, the DHT11 sensor, while cost-effective, has a lower accuracy compared to other temperature and humidity sensors like the DHT22.

2. **Limited Cloud Storage and Features:**
    - The Arduino IoT Cloud offers basic functionality, but it lacks advanced features such as data analytics or long-term storage. This limits the ability to perform detailed data analysis or store large datasets for extended periods.
    - The cloud platform's data logging capability is also limited in comparison to more sophisticated cloud services.

3. **Network Reliability:**
    - The system's reliance on Wi-Fi connectivity could pose a limitation in areas with unstable or poor network conditions. If the network connection is lost, the system will be unable to transmit data to the cloud, impacting the real-time monitoring capability.

4. **Power Consumption:**
    - The ESP32, while power-efficient compared to other microcontrollers, may still consume significant power during continuous operation. This may limit its applicability in battery-powered deployments, particularly in remote or off-grid locations.

5. **Mobile App Dependency:**
    - The system relies on the Arduino IoT Remote app for data visualization. If users experience issues with the app or are unable to connect to the cloud, it may limit their ability to access the data.

6.3 Future Scope

Despite the limitations mentioned, the project offers numerous opportunities for enhancement and expansion. The following future improvements can be considered:

1. **Integration of Additional Sensors:**
    - To improve the accuracy and reliability of environmental monitoring, additional sensors can be incorporated into the system. For example, integrating sensors for particulate matter (PM2.5, PM10), carbon dioxide (CO2), and volatile organic compounds (VOCs) would provide a more comprehensive assessment of air quality.

- Advanced sensors like the CCS811 for CO2 and TVOCs or the PMS5003 for particulate matter could be used to provide more accurate data.

2. **Data Analytics and Visualization:**
    - Future versions of the system can incorporate advanced data analytics capabilities to process and analyze the historical data collected by the sensors. Cloud platforms like AWS or Google Cloud can be used to store large datasets and perform machine learning-based predictions on air quality trends.
    - Enhanced data visualization techniques, such as time-series graphs or heatmaps, could provide deeper insights into air quality patterns and help users make informed decisions.

3. **Mobile Application Enhancement:**
    - The mobile app can be enhanced to provide more features, such as notifications when air quality levels exceed predefined thresholds or the ability to export data for offline analysis.
    - Incorporating additional languages and accessibility features would make the app more user-friendly for a wider audience.

4. **Energy Efficiency Improvements:**
    - To reduce power consumption, the system could be optimized by utilizing sleep modes on the ESP32 and minimizing Wi-Fi transmission intervals. This would make the system more suitable for battery-powered, long-term deployments.
    - Solar-powered options or the use of low-power sensors could help make the system more sustainable and suitable for outdoor monitoring.

5. **Advanced Cloud Solutions:**
    - Moving to a more robust cloud platform, such as AWS IoT or Microsoft Azure, could provide better scalability, more advanced analytics tools, and improved data storage. These platforms offer the ability to handle large volumes of data and provide more detailed insights into environmental patterns.
    - Incorporating the capability for multiple user accounts, user roles, and historical data analysis would enhance the system's capabilities.

6. **Real-World Deployments:**
    - The system could be deployed in urban or industrial environments to monitor air quality in real-time. By expanding the network of IoT-based air quality monitors, cities could create a network of sensors that contribute to smart city initiatives aimed at improving public health through air quality awareness.

6.4 Final Thoughts

The IoT-based air quality monitoring system provides an effective and scalable solution for real-time environmental monitoring. Despite some limitations, it demonstrates the power of IoT technology to create affordable and accessible tools for monitoring air quality. With future enhancements, the system can become even more powerful, reliable, and suitable for large-scale environmental monitoring applications. This project serves as a foundation for future work in the field of smart cities and environmental IoT solutions.