Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Artificial Intelligence (AI) and Data Sciences

Engineering Clinics

Project Report

IoT Based Weather Monitoring System

BACHELOR OF TECHNOLOGY

(Artificial Intelligence and Data Science.)

SUBMITTED BY :

Ayush Gupta, Ayush Sharma, Ayush Shukla, Bablesh Kumar
2420690, 240691, 2420692, 2420692, 2420693
October 2025

Under the Guidance of :

Er. Shubham Sharma (J4224)
Assistant Professor

Department of Artificial Intelligence and Data Science Chandigarh Engineering College Jhanjeri Mohali – 140307

Table of Contents

Chapter 1 : Abstract and Acknowledgement

1.1 Abstract

Weather monitoring is a critical field that impacts vital sectors like agriculture, aviation, and disaster management. However, traditional meteorological systems are often expensive, sparsely distributed, and provide data with significant delays. This creates a significant information gap, as they fail to provide the real-time, hyperlocal data needed for efficient decision-making.

This project, "**IoT Based Weather Monitoring System**," aims to overcome these limitations by developing a low-cost, scalable, and autonomous solution. The system integrates Internet of Things (IoT) and cloud technologies to provide accessible, real-time atmospheric data.

The core of the system is a standalone IoT node built using an **ESP32 microcontroller** and a suite of environmental sensors (**DHT22** for temperature and humidity, and **BMP280** for barometric pressure). This hardware node collects data and transmits it wirelessly over Wi-Fi using the **MQTT protocol** to the **ThingSpeak cloud platform**. The ThingSpeak platform is used for data aggregation, storage, and processing, providing a restful API for data retrieval.

1.2 Acknowledgement

We wish to express our sincere gratitude to our project guide, Er. Shubham Sharma, Assistant Professor, Department of AI&DS, for his invaluable guidance, constant encouragement, and insightful feedback throughout the duration of this project. His expertise and support were critical in navigating the technical challenges and successfully completing this work.

We are also thankful to the **Department of Artificial Intelligence and Data Science, Chandigarh Engineering College, Jhanjeri**, for providing us with the necessary resources, infrastructure, and a conducive environment that made this project possible.

Finally, we extend our heartfelt thanks to our friends, peers, and families for their unwavering support, valuable discussions, and encouragement, which motivated us to bring this project to a successful conclusion.

Chapter 2 : Declaration

We, **Ayush Gupta, Ayush Sharma, Ayush Shukla,** and **Bablesh Kumar,** hereby declare that the work presented in this project report entitled "**IoT Based Weather Monitoring System**" is the outcome of our own original work. This work has been carried out under the guidance of Er. Shubham Sharma (Assistant Professor) at Chandigarh Engineering College, Jhanjeri (Mohali).

This report, or any part thereof, has not been submitted, either in part or in full, for the award of any other degree, diploma, or certificate at this or any other university or institution. All sources of information and data have been duly acknowledged and cited.

Chapter 3 : Introduction

3.1 Background and Motivation

In an era increasingly affected by climatic volatility, the demand for precise and timely weather information has never been higher. Weather monitoring is a field of immense critical importance, directly influencing the safety, efficiency, and profitability of numerous sectors. From a farmer planning, irrigation schedules and protecting crops, to an airline plotting safe flight paths, to a city's disaster management team preparing for an impending storm, access to accurate weather data is fundamental.

However, the traditional infrastructure for meteorological observation, while sophisticated, was designed for a different era. It relies on a network of large, expensive, and sparsely distributed weather stations. The data from these stations is highly accurate for large-scale forecasting but often fails to capture localized weather phenomena, known as microclimates. Furthermore, the data processing and dissemination from these systems often involve significant time lags.

3.2 The Importance of Real-Time Weather Data

The value of weather data is directly proportional to its timeliness and granularity. Real-time data allows for proactive and immediate decision-making, which can be the deciding factor in outcomes across various domains:

- **Agriculture:** Farmers can optimize irrigation, pesticide application, and harvesting schedules based on immediate local conditions, directly impacting crop yields and resource conservation.

- **Aviation:** Real-time data on barometric pressure, temperature, and humidity is crucial for flight planning and ensuring safety, especially during take-off and landing.
- **Disaster Management:** The ability to monitor sudden changes in atmospheric conditions in real-time is essential for issuing timely warnings for flash floods, storms, or other weather-related emergencies.
- **Public Life:** Urban commuters, event planners, and even individuals can make more informed daily decisions with access to live weather updates specific to their immediate vicinity, rather than a generalized city-wide forecast.

3.3 Limitations of Traditional Systems

Traditional weather monitoring, primarily managed by governmental meteorological departments, has long been the gold standard. However, this approach is beset by several inherent limitations that this project seeks to address:

1. **High Cost:** The establishment and maintenance of a single professional-grade weather station are extremely expensive, involving sophisticated instrumentation and significant upkeep.
2. **Sparse Coverage:** Due to the high cost, these stations are sparsely distributed, often many miles apart. This results in data that represents a broad geographical average, failing to capture critical local variations (microclimates).
3. **Data Latency:** Data from these stations is typically processed centrally and disseminated through official channels with a notable time lag. This non-automated, manual collection process makes the

data unsuitable for applications requiring immediate intelligence.

4. **Inaccessibility:** The raw data is often not directly accessible to the public or small-scale users, who must rely on processed forecasts.

3.4 Proposed System Overview

This project introduces an **Internet of Things (IoT) based approach** to overcome the limitations of traditional systems. The primary goal is to design, develop, and deploy a low-cost, scalable, and autonomous weather monitoring system capable of providing hyperlocal, real-time atmospheric data.

The proposed system is composed of three main layers:

1. **Hardware Node:** A standalone IoT node is developed, serving as the core sensing unit. It is equipped with sensors to measure key environmental parameters: **temperature, humidity, and barometric pressure**.

2. **Cloud Backend:** The data collected by the node's microcontroller is transmitted wirelessly to a cloud server. This server is responsible for aggregating, storing, and processing the high-frequency time-series data.

3. **End-User Application:** The aggregated data is then made accessible to end-users through a custom-built **mobile application**. This application provides an intuitive interface with real-time gauges and historical charts, effectively democratizing weather data.

3.5 Scope of the Project

The scope of this project is to design and implement a functional prototype

that validates the feasibility of this IoT-based approach.

3.5.1 In-Scope

- Design and assembly of a low-cost hardware node using an ESP32 microcontroller and sensors for temperature, humidity, and pressure.
- Firmware development for data acquisition, Wi-Fi connectivity, and cloud transmission using MQTT/HTTP.
- Configuration of the ThingSpeak cloud platform for data storage and API access.
- Testing and validation of the system for data accuracy, transmission reliability, and power efficiency.

3.5.2 Out-of-Scope

- The initial prototype focuses on stationary monitoring and does not include sensors for wind speed or wind direction.
- The project does not include the development of a large-scale, grid-deployed network of nodes, but rather proves the concept for such a network.
- The project focuses on data collection and visualization; the development of complex AI-based predictive models is left as future work.

Chapter 4 : Literature Survey

4.1 Conventional Weather Monitoring Techniques

The field of meteorology has traditionally been the domain of government agencies and large research institutions. This conventional approach relies on a global network of highly sophisticated, expensive, and non-automated weather stations.

These stations, while providing highly accurate and reliable data, suffer from several drawbacks as noted in Chapter 1. The primary issue is **spatial density**. Due to the high deployment and maintenance costs, stations are placed many kilometres apart. This sparsity means the data collected is an average over a large area and cannot capture localized, rapidly changing weather events, which are often the most dangerous.

Furthermore, the data dissemination process is often slow. Data is collected, processed centrally, and then broadcast, leading to significant delays. For applications in precision agriculture or immediate disaster response, this time lag renders the data obsolete. These methods, while reliable for large-scale, long-term forecasting, are insufficient for applications requiring immediate, localized weather intelligence.

4.2 Advancements with IoT and Embedded Systems

The last decade has seen a revolution in electronics, catalysed by the emergence of **low-cost microcontrollers and sensors**. This has enabled a paradigm shift from centralized, expensive systems to decentralized, affordable, and "smart" devices.

The "Internet of Things" (IoT) describes this network of physical devices embedded with sensors, software, and connectivity, allowing them to exchange data over the internet. This technology is perfectly suited to address the shortcomings of traditional weather monitoring.

Pioneering work in this area has explored the use of popular embedded platforms like **Arduino and Raspberry Pi** for environmental data collection. These platforms are low-cost, flexible, and supported by large open-source communities, making them ideal for rapid prototyping. Researchers have successfully demonstrated the feasibility of building functional monitoring systems using these tools, proving that accurate data collection is possible on a small budget.

4.3 Review of Existing Systems

Our survey of existing literature revealed several projects with similar goals.

- **Patel and Jain (2018)** demonstrated a functional prototype for a low-cost weather monitoring system. Their work used an Arduino and a DHT11 sensor for measuring temperature and humidity. This research was foundational in establishing the feasibility of using low-cost components for data collection.
- Kumar, et al. (2019) proposed an "IoT-Based Smart Weather Monitoring System using NodeMCU". This work is particularly relevant as it utilizes a Wi-Fi-enabled microcontroller (NodeMCU, which uses the ESP8266) to send data to the cloud, moving beyond simple local logging.
- Oszczypala, et al. (2017) presented an "IoT-based environmental data monitoring system" at the TSP conference, highlighting the broader academic and industrial interest in using IoT for comprehensive environmental analysis, not just weather.

These studies and others like them have firmly established the viability of using embedded systems for data acquisition.

4.4 Research Gap

While the reviewed literature confirms the feasibility of IoT-based data collection, we identified a common gap: many projects stop at the proof-of-concept stage. They often end with **local data logging** (e.g., to an SD

card) or a **basic web page display** on a local network.

These systems lack the two most critical components for a truly practical solution:

1. **A Robot Cloud Backend:** A scalable system for data aggregation, long-term storage, and API-based access.
2. **A Polished End-User Application:** A user-friendly mobile or web app that makes the data accessible and useful to a non-technical user.

This project directly addresses this gap. We extend the concepts of local data collection by focusing on integration with a robust cloud platform (ThingSpeak) and developing a complete, end-to-end solution that includes a user-friendly mobile application. The goal is not just to collect data, but to make it immediately accessible and actionable.

Chapter 5 : System Requirements

5.1 Hardware Requirements

The following hardware components are required for the *development and deployment* of the IoT node:

Component	**Specification**	**Purpose**
Microcontroller	ESP32 Development Board	The core processing unit ("brain") of the system. Chosen for its built-in Wi-Fi, low power consumption, and dual-core processor.
Temp/Humidity Sensor	DHT22 (or AM2302)	Digital sensor to accurately measure ambient temperature and relative humidity. Chosen over DHT11 for its higher precision.
Pressure Sensor	BMP280 (or BMP180)	Digital sensor to measure barometric pressure. Essential for altitude calculations and localized weather prediction.
Prototyping	Breadboard and Jumper Wires	For assembling and testing the circuit without soldering.

Component	Specification	Purpose
Power	5V Power Supply	To power the ESP32 and sensors (e.g., via USB, battery pack).

5.2 Software Requirements

The following software tools and platforms are required for development:

Component	Specification	Purpose
Firmware IDE	Arduino IDE	Used to program the ESP32 microcontroller in C++.
App Dev IDE	Android Studio	The official IDE for developing the native Android application.
Cloud Platform	ThingSpeak IoT Platform Account	The cloud backend for data aggregation, storage, visualization, and API access.
Libraries	ESP32 Core for Arduino	Provides board definitions and core functions.
	Adafruit DHT Sensor Library	To interface with the DHT22 sensor.

Component	Specification	Purpose
	Adafruit BMP280 Library	To interface with the BMP280 sensor via I2C/SPI.
	ThingSpeak MQTT/HTTP Library	To simplify data transmission to the cloud.

5.3 Functional Requirements

Functional requirements define the specific behaviors and functions of the system.

ID	Requirement	Description
FR-01	Data Acquisition	The system's hardware node *must* accurately read data from the temperature, humidity, and barometric pressure sensors
FR-02	Configurable Polling	The system *must* allow the data reading interval to be configured in the firmware (e.g., every 60 seconds)
FR-03	Wi-Fi Connectivity	The hardware node *must* be able to connect to a predefined Wi-Fi network using an SSID and password.

ID	Requirement	Description
FR-04	Data Transmission	The hardware node *must* securely transmit the collected sensor data (temperature, humidity, pressure) to the ThingSpeak cloud platform.
FR-05	Cloud Storage	The cloud platform *must* receive and store the time-stamped sensor data in distinct channels.
FR-06	Data API	The cloud platform *must* provide a RESTful API or retrieving the stored historical and real-time data.
FR-07	Real-time Visualization	The Android application *must* display the *latest* sensor values using gauges or text fields.
FR-08	Historical Visualization	The Android application *must* display *historical* data for each sensor in the form of interactive charts or graphs.
FR-09	Data Filtering	The application *should* allow users to filter historical data by time (e.g., last 24 hours, last 7 days).

5.4 Non-Functional Requirements

Non-functional requirements define the quality attributes and constraints of the system.

ID	Requirement	Description	Rationale / Metric
NFR-01	Reliability	The hardware node *must* be resilient to network failures.	Must implement reconnection logic to auto-reconnect to Wi-Fi if the connection drops.
NFR-02	Accuracy	Sensor readings *must* be reasonably accurate.	Validated readings against a commercial source must be within a ±5% deviation.
NFR-03	Performance	Data *must* be displayed in near real-time.	Total latency from data collection to app display should be under 2 minutes (limited by polling interval).
NFR-04	Power Efficiency	The hardware node *must* be power-efficient.	Must implement Deep Sleep mode to maximize battery life. Tested to last >5 days on a 3000mAh battery.
NFR-05	Scalability	The cloud architecture *must* be	ThingSpeak is designed to handle millions of data points from many devices,

ID	Requirement	Description	Rationale / Metric
		scalable.	meeting this requirement.
NFR-06	Usability	The Android application *must* be intuitive and easy to use.	The user interface must be clean, and data must be presented in an easily digestible visual format.
NFR-07	Cost	The hardware node *must* be low-cost.	The total bill of materials (BOM) for one node must be kept to a minimum (under $20).
NFR-08	Maintainability	The firmware code *must* be well-commented and modular.	To allow for future updates (e.g., adding new sensors).

Chapter 6 : System Design

6.1 System Architecture

The system is designed using a **three-layer architecture**, which separates the concerns of data sensing, data processing, and data presentation. This modular design is robust, scalable, and easy to maintain.

The three layers are:

1. **Hardware (Sensing) Layer:** The physical layer responsible for sensing the environment and transmitting data.
2. **Communication & Cloud Layer:** The middleware responsible for data transport, storage, and processing.
3. **Application (Presentation) Layer:** The user-facing layer that visualizes the data and provides insights.

The following sections detail the components and technologies used in each layer.

6.2 Hardware Layer

The Hardware Layer is the physical foundation of the project. It consists of the primary sensing unit responsible for data acquisition.

- **Microcontroller (MCU): ESP32** The **ESP32** is used as the core processing unit, or the "brain," of the IoT node. This powerful and low-cost MCU was chosen over other options (like Arduino Uno) for its critical built-in features:
    - **Integrated Wi-Fi:** Allows the device to connect to the internet directly without needing an external "shield" or module.
    - **Low Power Consumption:** Features an "Ultra Low Power" (ULP) co-processor and Deep Sleep modes, which are essential for battery-powered operation.
    - **Dual-Core Processor:** Provides ample processing power for

handling sensor readings, Wi-Fi management, and data encryption simultaneously.

- **Sensors:**
  - **DHT22 (Temperature & Humidity):** This is a digital sensor that provides calibrated measurements for ambient temperature and relative humidity. It was chosen for its reliability and higher accuracy compared to the cheaper DHT11.
  - **BMP280 (Barometric Pressure):** This sensor from Bosch measures atmospheric pressure. It communicates with the ESP32 via the I2C protocol. This data is essential for altitude compensation and as a key indicator for short-term weather prediction.

6.3 Communication & Cloud Layer

This layer acts as the bridge between the physical hardware and the end-user application. It handles data transmission, storage, and API access.

- **Communication Protocol: Wi-Fi & MQTT** The ESP32 collects sensor data and transmits it over a standard **Wi-Fi** (802.11b/g/n) network. For data transmission, the **MQTT** protocol is used. MQTT is an extremely lightweight, publish-subscribe messaging protocol. It is the industry standard for IoT because it is:
  - **Lightweight:** Has a very small packet header, minimizing network bandwidth.
  - **Reliable:** Provides quality-of-service (QoS) levels to ensure messages are delivered.
  - **Efficient:** Ideal for high-latency or unreliable networks, fitting our Wi-Fi flakiness challenge.
- **Cloud Platform: ThingSpeak** The **ThingSpeak IoT platform** is used as the cloud backend. The ESP32 "publishes" its sensor data to a specific ThingSpeak "channel". ThingSpeak was chosen for its simplicity and powerful features:
  - **Data Aggregation:** It easily aggregates data streams from

multiple devices.
- **Built-in Visualization:** Provides instant graphs and gauges for testing and monitoring data as it arrives.
- **MATLAB Analytics:** Allows for in-cloud data processing using MATLAB, which can be used for future predictive modelling.

6.4 Application Layer

This is the presentation layer, where raw data is transformed into useful, actionable information for the end-user.

- **Development Platform: Android Studio** An Android application is developed using **Android Studio**. A native Android app was chosen to provide a responsive, user-friendly experience and to accommodate future features like push notifications (e.g., "Frost warning for your location!").
- **Application Functionality** The app serves as the primary user interface. Its core functions are:
  - **API Data Fetching:** The app periodically makes HTTP GET requests to the ThingSpeak RESTful API to retrieve the latest data in JSON format.
  - **Data Visualization:** It parses the JSON data and displays it on a clean dashboard.
  - **Real-time Gauges:** Shows the *most recent* temperature, humidity, and pressure values for a quick, at-a-glance check.
  - **Historical Charts:** Uses a charting library to plot the data over time, allowing users to identify trends.

6.5 Work Plan

Stage 1: Hardware Assembly and Prototyping

- **Tasks:** Source all required components (ESP32, DHT22, BMP280, breadboards). Assemble the initial prototype on a breadboard. Write basic test scripts to ensure each sensor is functioning and providing

readings.

- **Outcome:** A functional hardware prototype.

Stage 2: Microcontroller Programming (Firmware)

- **Tasks:** Develop the main firmware using the Arduino IDE. Implement Wi-Fi connectivity logic, including auto-reconnect. Integrate sensor libraries. Implement the MQTT data publishing protocol. Add error handling for sensor failures and power-saving deep-sleep cycles.
- **Outcome:** A stable, autonomous firmware that collects and transmits data.

Stage 3: Cloud and Application Development

- **Tasks:** Set up the ThingSpeak channel, fields, and API keys. Begin development of the Android application in Android Studio. Design the UI/UX for the dashboard. Implement the networking logic in the app to fetch and parse JSON data from the ThingSpeak API.
- **Outcome:** A functional ThingSpeak channel receiving data and a mobile app capable of displaying it.

Stage 4: System Integration and Testing

- **Tasks:** Deploy the complete end-to-end system. Run continuous operation tests (e.g., a 48-hour test). Validate sensor accuracy against a commercial weather source. Identify and resolve bugs (e.g., sensor noise, Wi-Fi drops).
- **Outcome:** A fully validated and debugged prototype system.

Chapter 7 : Implementation

7.1 Overview of Technology Stack

The implementation of this project relies on a stack of technologies spanning hardware, firmware, cloud services, and mobile development.

- **Hardware:** ESP32 Development Board , DHT22 Temperature/Humidity Sensor , BMP280 Barometric Pressure Sensor.
- **Firmware:** C/C++ via the Arduino IDE. Key libraries include WiFi.h (for connectivity), PubSubClient (for MQTT), and sensor-specific libraries (e.g., Adafruit_BMP280).
- **Cloud Platform:** ThingSpeak IoT Platform. Used for data ingestion, storage, and API provision.
- **Communication Protocol:** MQTT for publishing data from the ESP32, and REST/HTTP for data retrieval by the Android app.
- **Application:** Native Android (Java/Kotlin) developed in Android Studio. Key libraries include Retrofit (for API calls) and MPAndroidChart (for data visualization).

7.2 Microcontroller Programming (Firmware)

The firmware was developed in C/C++ using the Arduino IDE. The code is structured into several key logical blocks:

1. **Initialization (setup()):**
    - Initializes the serial monitor for debugging.
    - Initializes the DHT22 and BMP280 sensors.
    - Calls a setup_wifi() function, which connects the ESP32 to the predefined Wi-Fi SSID and password.
    - Sets up the MQTT client with the ThingSpeak server address, channel ID, and API key.

2. **Main Loop (loop()):**
    - Checks the Wi-Fi and MQTT connection status. If disconnected, it triggers a reconnection function.
    - Reads data from the DHT22 (temperature, humidity) and BMP280 (pressure).
    - Implements error handling to check for failed sensor readings.
    - Applies a rolling average filter to the readings to smooth out sporadic noise.
    - Formats the data into the string format required by the ThingSpeak MQTT API.
    - Publishes the formatted data string to the ThingSpeak channel.
    - Enters **Deep Sleep mode** for a 60-second interval. This turns off the CPU and Wi-Fi, dramatically reducing power consumption. After 60 seconds, the device resets and runs the setup() function again.

7.3 Cloud Platform Configuration (ThingSpeak)

The ThingSpeak platform was configured to receive and store the data.

1. **Channel Creation:** A new channel was created on the ThingSpeak platform.
2. **Field Configuration:** The channel was configured with three active fields to store the incoming data:
    - Field 1: Temperature (°C)
    - Field 2: Humidity (%)
    - Field 3: Pressure (hPa)
3. **API Key Generation:** ThingSpeak automatically generates API keys for the channel.
    - **Write API Key:** This key was programmed into the ESP32 firmware to authorize it to *publish* data to the channel.
    - **Read API Key:** This key was used in the Android application to authorize it to *fetch* data from the channel.

7.4 Application Development (Android)

The end-user application was developed using Android Studio.

1. **UI/UX Design:** The main screen features a clean dashboard.
    - **Real-time Gauges:** At the top, three prominent gauges (or text cards) display the *latest* values for temperature, humidity, and pressure.
    - **Historical Charts:** Below the gauges, three line charts display the historical trends for each parameter.
    - **Controls:** A "refresh" button and a time-filter (e.g., "24h", "7d") are provided for user control.
2. **Networking (Retrofit):** The Retrofit library was used to handle network operations. A ThingSpeakAPI Service interface was defined to describe the API endpoints.
    - A GET request is sent to the ThingSpeak API
    - Parameters are added to the request, such as the api_key (the Read API Key) and results (the number of data points to fetch).
3. **Data Parsing (JSON):** The API returns data in JSON format. This JSON object contains an array called feeds, where each feed is a time-stamped data point. The app parses this array.
4. **Visualization (MPAndroidChart):** The MPAndroidChart library was used to render the line graphs. The parsed feeds array is transformed into a DataSet object, which is then used to populate the charts, with timestamps on the X-axis and sensor values on the Y-axis.

Chapter 8 : Results and Discussion

8.1 Project Results Overview

The project was successful in achieving all its primary objectives. A functional, end-to-end prototype of an IoT-based weather monitoring system was designed, built, and validated.

The prototype **successfully operated continuously**, validating the core design principles of using an ESP32, ThingSpeak, and a custom Android application. The system reliably collected data from the DHT22 and BMP280 sensors, transmitted it to the cloud, and displayed it on the mobile dashboard.

The following sections detail the specific outcomes of the functional and performance testing.

8.2 Challenges Faced & Solutions Implemented

No project is without its challenges. The following three issues were identified during testing and successfully resolved.

- **Challenge: Wi-Fi Flakiness & Connection Drops**
  - **Problem:** The initial firmware was "optimistic," assuming the Wi-Fi would always be available. During testing, if the router was reset, the ESP32 would crash or hang, failing to send any more data.
  - **Solution:** We implemented **robust re-connection logic** in the firmware's main loop. Before any attempt to publish, the code now checks the WiFi.status() and mqtt.connected(). If either check fails, it triggers a non-blocking reconnect() function that continuously attempts to re-establish the connection before proceeding.
- **Challenge: Sensor Drift & Noise**
  - **Problem:** We observed that the DHT22 and BMP280 sensors

would occasionally (e.g., 1 in 100 readings) return a completely nonsensical value or NaN (Not a Number). This would corrupt the data stream and create large, ugly spikes in the graphs.

- **Solution:** We applied a two-part software-based solution. First, we added error handling to check for NaN or unrealistic values (e.g., temp < -40 or > 80), discarding them if found. Second, we applied a **rolling average (median filter)** to the last 3-5 readings to smooth out sporadic, minor fluctuations and minimize the impact of noise

- **Challenge: Power Optimization**
  - **Problem:** The initial prototype, without power management, drained the 3000mAh battery in under 10 hours. The ESP32's Wi-Fi radio is extremely power-hungry when active.
  - **Solution:** We re-architected the firmware to use the ESP32's **Deep Sleep cycle**. Instead of a continuous loop(), the device now wakes up, connects to Wi-Fi, reads sensors, publishes data, and then immediately tells the RTC (Real-Time Clock) to wake it up again in 60 seconds before entering deep sleep. This reduced the "on-time" to only ~4-5 seconds per minute, drastically extending battery life.

Chapter 9 : Conclusion and Future Work

9.1 Conclusion

This project has been an invaluable experience in applying the principles of Artificial Intelligence and Data Science to a tangible, real-world hardware environment. We successfully designed, built, and validated a complete, end-to-end **IoT Based Weather Monitoring System**.

The project achieved its core objective of creating a low-cost, scalable, and autonomous system that overcomes the primary limitations of traditional weather monitoring. By integrating a custom-built hardware node with a robust cloud platform and a user-friendly mobile application, we have demonstrated a practical solution for providing the **real-time, hyperlocal weather data** that modern applications in agriculture, logistics, and public safety demand.

The final prototype was thoroughly tested and proven to be reliable, accurate, and power-efficient. The challenges encountered with network reliability and sensor noise were systematically solved, resulting in a stable platform. This project serves as a strong proof-of-concept and a solid foundation for a larger, more comprehensive environmental monitoring network.

9.2 Future Enhancements

While the current prototype is a complete success, this project lays the groundwork for numerous exciting expansions. We aim to expand the project's capabilities in the next phase:

1. **Add More Sensors:** The most immediate enhancement is to integrate an **anemometer and a wind vane**. This would add crucial data on wind speed and direction, making the station much more comprehensive. Sensors for UV index, air quality (PM2.5), and rainfall (tipping bucket) could also be added.
2. **Solar Power Integration:** To make the node truly "deploy-and-forget" and suitable for remote agricultural fields, we will design a **complete solar power solution**. This would involve integrating a solar panel, a charging circuit, and a rechargeable battery (e.g., Li-ion) to allow for indefinite, off-grid operation.
3. **AI-Based Localized Prediction:** The true power of this project lies in the data it collects. With sufficient historical data from one or more nodes, we will use the collected time-series data to build and test a **localized, short-term weather prediction model**. Using AI/ML techniques (like LSTM networks) on the ThingSpeak platform's MATLAB integration, we could move from *monitoring* to *forecasting*.
4. **Mesh Networking:** For large-area deployment (like a large farm), Wi-Fi may not be available everywhere. Future work could explore using mesh networking protocols (e.g., LoRaWAN, Zigbee) to allow nodes to relay data back to a central gateway, creating a wide-area, low-power network.
5. **App Enhancements:** The Android app could be improved with push notifications for user-defined alerts (e.g., "Temperature below 5°C" or "Pressure dropping rapidly").

Chapter 10 : References

1. Adithan, M. and Gupta, A.B. (1996), "Manufacturing Technology", New Age, International Publishers, New Delhi.
2. Kumar, N., Kumar, S. and Prasad, R., 2019. "IoT-Based Smart Weather Monitoring System using NodeMCU", Proceedings of Second International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, pp. 245-249.
3. Oszczypala, M., Szybisty, A. and Wojas, A., 2017. "An IoT-based environmental data monitoring system", Proceedings of the 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, Spain, pp. 521-525.
4. Patel, K. and Jain, S., 2018. "A Low-Cost Weather Monitoring System Using Arduino and IoT", International Journal of Engineering Research & Technology, Vol. 6, No. 13, pp. 1-4.
5. Singh, S. and Shan, H. S. (2002) "Development of Magneto Abrasive Flow Machining Process", International Journal of Machine Tools & Manufacturing, vol.42, 2, 2002, pp. 953-959.
6. *Arduino IDE.* (n.d.). Retrieved from https://www.arduino.cc/en/main/software
7. *ThingSpeak IoT Analytics.* (n.d.). Retrieved from https://thingspeak.com
8. *ESP32 Datasheet.* (n.d.). Espressif Systems. Retrieved from https://www.espressif.com/en/products/socs/esp32
9. *DHT22 (AM2302) Datasheet.* (n.d.). Aosong.
10. *BMP280 Datasheet.* (n.d.). Bosch Sensortec.