Project Report on

LIVECAST WEATHER STATION

Submitted by

Swadev Mohapatra (2001229140)
Suresh Kumar Barik (2001229139)
Pratyasa Mohanty (2001229105)

Project Report submitted in partial fulfillment of the requirements for the award of degree of B.Tech. in Computer Science & Engineering under DRIEMS University

DRIEMS UNIVERSITY

2020 - 2024

Under the Guidance of

Prof. Rojalin Dash

Asst. Professor, Dept. of CSE

Department of Computer Science and Engineering

School of Engineering & Technology, Tangi, Cuttack

DRIEMS UNIVERSITY

Department of Computer Science & Engineering

School of Engineering and Technology, Tangi, Cuttack - 745022

CERTIFICATE

This is to certify that, this is a Bonafide Project report, titled "Livecast Weather Station", done satisfactorily by Swadev Mohapatra (2001229140), Suresh Kumar Barik (2001229139) and Pratyasa Mohanty (2001229105) in partial fulfilment of requirements for the degree of B.Tech. in Computer Science & Engineering under DRIEMS University.

This Project report on the above-mentioned topic has not been submitted for any other examination earlier before in this institution and does not form part of any other course undergone by the candidate.

Signature
External Examiner

Prof. Rojalin Dash
Asso. Professor,
Dept. of CSE
Guide

Prof. Surajit Mohanty
Asso. Professor,
Head Dept. of CSE

ACKNOWLEDGEMENT

We express indebtedness to our guide **Prof. Rojalin Dash**, of the Computer Science & Engineering department who spared her valuable time to go through manuscript and offer her scholar advice in the writing. Her guidance, encouragement and all out help have been invaluable to us. There is short of words to express our gratitude and thankfulness to her.

We are grateful to all the teachers of Computer Science & Engineering department, DRIEMS, for their encouragement, advice, and help.

At the outset, we would like to express our sincere gratitude to **Prof. Surajit Mohanty**, H.O.D of Computer Science & Engineering department for his moral support extended towards us throughout the duration of this project.

We are also thankful to our friends who have helped us directly or indirectly for the success of this project.

Swadev Mohapatra (2001229140)
Suresh Kumar Barik (2001229139)
Pratyasa Mohanty (2001229105)
Department of Computer Science & Engineering

ABSTRACT

The "LiveCast Weather Station" is an innovative web-based platform designed to provide real-time and forecasted environmental data, specifically tailored for applications in smart farming and the agriculture industry. The system integrates a 14-day weather forecast, detailed air quality metrics, and astronomical data about the sun and moon using the WeatherAPI service. Users can input their location through city names, GPS-based geolocation, or manual latitude and longitude, allowing for personalized and precise data retrieval.

The frontend of the application is developed using Next.js, offering a responsive and dynamic user interface, while the backend is powered by Node.js and Express.js, acting as a middleware to securely handle API requests. A unique component of the project is its integration with an IoT-based local weather station comprising a NodeMCU microcontroller connected to DHT11 (temperature and humidity), BMP180 (barometric pressure), LDR (light intensity), and FC-37 (rain detection) sensors. The data collected is uploaded in real-time to ThingSpeak for visualization and analysis.

This project demonstrates a practical use case in smart agriculture, where farmers can leverage accurate environmental data to make informed decisions. The solution is scalable, cost-effective, and contributes to the development of sustainable and data-driven farming practices.

Keywords:-
weather forecast, air quality forecast, astronomy data forecast, real-time data, data visualization, GPS-based location, WeatherAPI, geolocation, Next.js, Node.js, Express.js, IoT, NodeMCU, DHT11, BMP180, LDR light sensor, FC-37 rain sensor, C++, ThingSpeak, smart farming.

CONTENTS

LIST OF FIGURES

PAGE NO.

CHAPTER 1

1.1 INTRODUCTION

The rapid advancement of technology has significantly influenced traditional industries, and agriculture is no exception. With the growing need for sustainable and efficient farming practices, integrating smart technologies into agriculture has become a crucial step forward. The “LiveCast Weather Station” project is a web-based platform designed to provide detailed and real-time environmental data to support the agriculture industry, particularly in the domain of smart farming.

This system is built using a combination of web development technologies and Internet of Things (IoT) components. It utilizes the WeatherAPI service to fetch a 14-day detailed weather forecast, real-time air quality index, and astronomical data such as sunrise, sunset, moon phases, and moonrise/moonset timings. The platform accepts location input from users through city, town, or village names, GPS-based geolocation using the browser’s navigator.geolocation API, or direct latitude and longitude coordinates.

On the hardware side, the project includes a custom-built IoT weather station featuring a NodeMCU microcontroller programmed using C++ in the Arduino IDE. It connects with sensors like DHT11 (temperature and humidity), BMP180 (barometric pressure), LM393 LDR (light intensity), and FC-37 (rain detection). This data is sent to a third-party cloud platform, ThingSpeak, which allows real-time monitoring and visualization. The website itself is developed using Next.js for the frontend, ensuring a fast and responsive interface, while the backend is powered by Express.js and Node.js to handle API calls and data processing. Overall, the project bridges technology and agriculture, offering a practical solution for informed, data-driven farming decisions.

1.2 LITERATURE SURVEY

1. **" IoT-Based Real-Time Weather Monitoring and Reporting System " :-**

The existing literature on IoT-based real-time weather monitoring systems underscores a growing trend toward the deployment of interconnected sensor networks for comprehensive environmental data collection. These systems leverage platforms such as ThingSpeak, which allow for the seamless processing, visualization, and real-time display of meteorological data. Due to their affordability and efficiency, these systems are proving essential for continuous weather analysis and short- to long-term forecasting. Furthermore, their integration into broader environmental and pollution monitoring initiatives provides critical data that can be used to inform public health decisions and implement timely protective measures. Their relevance is increasingly apparent in urban and industrial zones, where environmental quality directly affects population well-being. [1]

2. **" IoT Based Weather Monitoring System ":-**

The incorporation of Internet of Things (IoT) technologies into environmental monitoring has significantly advanced the development of intelligent ecosystems that can self-monitor and interact dynamically with changing climatic conditions. By embedding a variety of sensors, these systems can collect real-time data on temperature, humidity, air pressure, and other vital parameters, transmitting it through Wi-Fi networks to centralized platforms. This approach greatly enhances the speed and reliability of environmental monitoring processes. Additionally, IoT's versatility is highlighted through its application in sustainable agriculture, particularly in vertical farming environments using hydroponics and aquaponics. These methods benefit from continuous environmental feedback, optimizing crop yield in confined urban spaces and contributing to food security and resource efficiency. [2]

3. **" Internet of Things (IOT) based Weather Monitoring System ":-**

Research into IoT-enabled environmental monitoring systems demonstrates the significant improvements they bring in terms of accuracy, scalability, and interactivity. These systems utilize a network of sensors that continuously capture and analyze climatic data, enabling smart environments to respond proactively to changing conditions. The transmission of this data through wireless connections to cloud-based platforms such as Google Sheets ensures

broad accessibility and supports data-driven research and policy planning. Their cost-effective nature makes them especially advantageous for densely populated urban areas and pollution-sensitive industrial zones, offering a scalable solution for maintaining public health and environmental safety. These systems are setting new benchmarks for how urban planning and ecological management can be conducted more effectively using real-time data insights. [3]

4. **" An IOT Based Weather Monitoring System " :-**

Recent academic and technical studies highlight the transformative potential of IoT technology in modern weather monitoring applications. These systems, designed for affordability and energy efficiency, are equipped with a range of sensors that detect and report key climatic variables such as temperature, humidity, barometric pressure, carbon dioxide levels, and rainfall intensity. Their modular design allows for easy integration of additional components, making them adaptable to specific monitoring needs. With enhanced weather prediction capabilities, these systems contribute to more precise climate modeling and help determine suitable environmental conditions for human habitation and agriculture. Their ability to scale across different regions further reinforces their utility in achieving sustainable environmental management goals, particularly in areas vulnerable to climatic variability. [4]

5. **" Real Time Weather Monitoring System Using Iot ":-**

The scholarly and technical literature on IoT-based weather monitoring systems emphasizes their low-cost structure and high utility in capturing environmental data through multi-sensor arrays. These systems typically operate on a client-side architecture model, which streamlines data collection, local processing, and real-time reporting. In regional contexts like Gorakhpur, such systems have demonstrated significant value by offering location-specific weather data that can inform vital sectors such as agriculture, transportation, and industrial operations. By improving the accuracy and availability of environmental information, these systems empower decision-makers with the insights necessary to enhance productivity, manage resources more efficiently, and ensure safety during adverse weather conditions. Their adaptability and reliability make them a critical component in smart city and rural development initiatives. [5]

1.3 PROBLEM DEFINITION

Agriculture remains one of the most essential sectors globally, yet it faces numerous challenges due to unpredictable weather patterns, climate change, and lack of access to real-time environmental data. Farmers, especially in rural or resource-limited regions, often rely on traditional methods or outdated forecasts that do not provide the precision needed for modern, efficient farming. These limitations can lead to poor crop management, reduced yields, and economic losses.

Another significant issue is the absence of localized data. General weather forecasts available online may not accurately reflect microclimatic conditions in specific areas, especially in villages or remote farmlands. Additionally, most farmers do not have access to affordable, real-time monitoring tools that can track environmental parameters such as temperature, humidity, rainfall, and air quality, which are critical for effective agricultural planning.

The "LiveCast Weather Station" project aims to address these issues by offering a web-based solution that combines global forecast data with local sensor readings from an IoT weather station. By integrating modern web technologies and affordable sensor hardware, this project delivers personalized, accurate, and real-time weather and environmental data. It empowers farmers with the information needed to make better decisions, reduce risks, and promote sustainable smart farming practices.

1.4 MOTIVATION

The motivation behind this work is to equip farmers with accessible and real-time environmental data, enabling them to make more accurate and timely decisions in their agricultural activities. By integrating IoT with web-based technologies, the project enhances the ability to monitor and respond to changing weather patterns. This combination not only supports more efficient and informed farming practices but also contributes to sustainable agriculture by helping to mitigate the impact of random climate conditions.

1.5 OBJECTIVE

- To develop a web platform that provides detailed, location-based weather forecasts, air quality, and astronomical data using WeatherAPI, enhancing decision-making for agriculture and smart farming.

- To integrate an IoT-based weather station that uploads real-time environmental data to ThingSpeak, supporting continuous monitoring of local climate conditions crucial for efficient and sustainable farming practices.

1.6 SCOPE AND DIRECTION

The "LiveCast Weather Station" project involves building an integrated web and IoT-based system that provides real-time and forecasted environmental data, primarily supporting agriculture and smart farming. It allows users to input locations via city/town names, GPS-based geolocation, or latitude-longitude coordinates. Powered by WeatherAPI, the website displays a 14-day weather forecast, air quality index, and astronomical data like sun and moon phases.

The project also includes a custom IoT weather station with sensors such as DHT11, BMP180, LM393 LDR, and FC-37 rain sensor, controlled by a NodeMCU microcontroller. These sensors collect local temperature, humidity, pressure, light intensity, and rainfall data, which is uploaded to ThingSpeak for real-time monitoring and visualization.

Developed using Next.js for the frontend and Node.js with Express.js for the backend, the platform is fast, scalable, and secure. Future enhancements may include SMS/app alerts, AI-based predictions, and multilingual support, extending its value for diverse agricultural needs.

CHAPTER 2

TECHNOLOGIES USED

2.1 DEVELOPMENT TOOLS

2.1.1 VS CODE

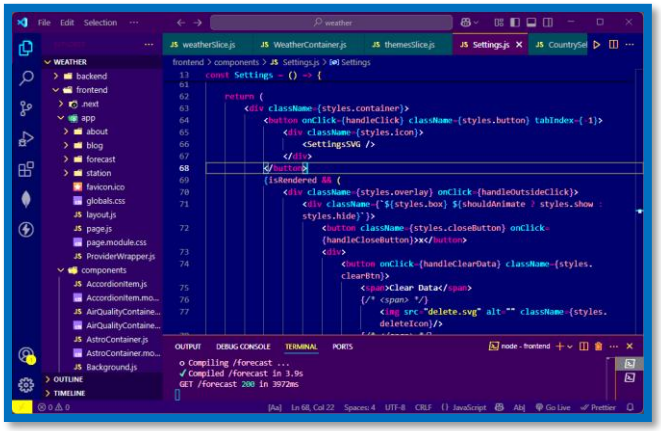

Fig 2.1: Vs Code

- Rich HTML, CSS, JS Support: VS Code provides built-in support for HTML, CSS, and JavaScript, including intelligent code completion and live error checking. This improves development speed and reduces errors while building front-end components of websites and web applications efficiently.

- Live Server Extension: The Live Server extension allows real-time preview of web pages in the browser. Developers can instantly see changes made to HTML, CSS, or JavaScript, significantly enhancing productivity by eliminating the need for constant manual refreshing.

- Node.js and Framework Integration: VS Code supports back-end technologies like Node.js, Express, and frameworks like React or Next.js. With terminal access and debugging tools, developers can run servers, manage APIs, and handle full-stack development from a single environment.

2.1.2 ARDUINO IDE

Fig 2.2: Arduino IDE

- Beginner-Friendly Interface: Its intuitive interface makes it easy for beginners to get started with IoT projects, including weather monitoring systems.

- Extensive Library Support: Arduino IDE provides a vast collection of libraries for sensors, communication modules, and other peripherals commonly used in weather monitoring applications, simplifying development.

- Cross-Platform Compatibility: It runs on multiple operating systems, ensuring compatibility with various hardware platforms, including NodeMCU boards used in IoT projects.

- Integrated Development Environment: Arduino IDE offers an all-in-one development environment with code editor, compiler, and uploader tools, streamlining the development process.

- Community Support: Arduino has a large and active community of developers and enthusiasts who contribute tutorials, guides, and troubleshooting assistance, aiding beginners and experienced developers alike.

2.2 WEB TECHNOLOGIES (FRONTEND)

2.2.1 HTML, CSS, JS, JSX

Fig 2.3: HTML, CSS, JS, JSX

- HTML: It provides the foundational structure of web pages by defining elements like headings, paragraphs, buttons, and forms, ensuring proper content layout and semantic organization in the application.

- CSS: It is used for styling the website, including layout, colors, fonts, and responsiveness, enhancing the visual appearance and user experience across various devices and screen sizes.

- JavaScript (JS): It enables dynamic behavior and interactivity on the website, handling logic, user input, API calls, and real-time updates without requiring full page reloads.

- JSX: It (JavaScript XML) is a powerful syntax extension for JavaScript, predominantly used in React, that enables developers to write code that closely resembles HTML directly within JavaScript files. This hybrid syntax bridges the gap between logic and layout, making the process of building user interfaces much more intuitive and visually structured. By allowing HTML-like tags to be embedded alongside JavaScript expressions, JSX enhances code readability and maintainability, especially in complex component hierarchies. It simplifies the development workflow by reducing the need for traditional createElement calls, making it easier to visualize component structure and behavior.

2.2.2 Redux Toolkit, NPM Packages

Fig 2.4: Redux Toolkit

- Redux Toolkit simplifies state management in React applications by reducing boilerplate code and providing pre-configured functions for creating slices, reducers, and asynchronous logic using createAsyncThunk.

- It improves scalability and maintainability by organizing state logic in a structured way, enabling better debugging, testing, and efficient handling of global state across large, complex applications.

Fig 2.5: NPM

- NPM packages are reusable code libraries available through the Node Package Manager. They simplify development by providing pre-built solutions for various tasks like routing, state management, UI components, and API handling.

- NPM packages used in this project are:
  1. react-loader-spinner (6.1.6): Customizable loading spinners
  2. react-select (5.10.1): Stylish customizable select dropdown
  3. react-toastify (11.0.5): Toast notifications for React
  4. react-top-loading-bar (3.0.2): Top-loading progress bar
  5. world-countries (5.1.0): Country data in JSON

2.3 WEB TECHNOLOGIES (BACKEND)

2.3.1 Express Js

Fig 2.6: Express Js

- Minimal and Fast Framework: Express.js is a lightweight Node.js framework that simplifies building web servers and APIs. It offers a minimal structure, allowing developers to create robust back-end services quickly without dealing with unnecessary complexity or boilerplate code.

- Middleware Support: Express.js supports middleware functions that handle requests, responses, and errors. This modular approach enables clean code organization, easier debugging, and flexible control over the application's request pipeline, making development more manageable and scalable.

- Routing Made Simple: It provides a powerful and flexible routing system that maps HTTP requests to specific functions. This helps in creating dynamic web pages and RESTful APIs with clearly defined endpoints for handling GET, POST, PUT, and DELETE operations.

- Full Stack Compatibility: Express.js integrates seamlessly with front-end frameworks like React or Next.js and databases like MongoDB. This makes it ideal for full-stack development, enabling developers to manage server-side logic and data interactions within a single JavaScript ecosystem.

2.3.2 Node Js

Fig 2.7: Node Js

- Event-Driven Architecture: Node.js uses a non-blocking, event-driven model that handles multiple requests efficiently. This makes it ideal for real-time applications like weather dashboards or IoT systems, where fast data processing and continuous connectivity are crucial.

- Single Language Stack: Developers can use JavaScript for both front-end and back-end with Node.js. This unification streamlines development, reduces context switching, and allows seamless communication between server and client—perfect for building full-stack applications like livecast weather platforms.

- Huge Ecosystem and NPM: Node.js comes with npm, a vast library of reusable packages. These packages help speed up development by offering ready-made modules for API calls, database integration, or sensor communication in IoT and smart farming applications.

- Excellent Scalability: Node.js handles large-scale applications efficiently by managing many simultaneous connections with minimal overhead. This scalability makes it suitable for data-intensive projects like smart weather systems that require continuous data flow from multiple sensors or users.

2.4 API

Fig 2.8: API

- API: An API (Application Programming Interface) allows different software systems to communicate by exposing defined endpoints. It enables access to specific features or data without exposing the underlying code or internal logic.

- REST API: A REST API follows the principles of Representational State Transfer, using standard HTTP methods like GET, POST, PUT, and DELETE. It enables stateless communication between client and server over URLs, commonly using JSON for data exchange.

- JSON: A JSON (JavaScript Object Notation) is a lightweight, human-readable data format used for structuring data. It's language-independent and widely used in APIs for sending and receiving structured information between servers and web applications.

- x-www-form-urlencoded: This format encodes form data into key-value pairs separated by '&' and uses '=' between keys and values. It's primarily used in HTTP POST requests when submitting HTML forms or interacting with APIs expecting URL-encoded data.

2.5 IOT TECHNOLOGIES (HARDWARE)

2.5.1 NodeMCU

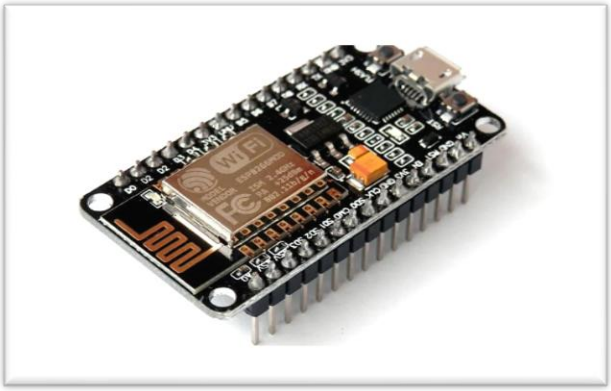

Fig 2.9: NodeMCU

NodeMCU ESP8266 is a popular choice for IoT projects like weather monitoring due to its versatility, connectivity options, and ease of use. Here's an overview of its features and advantages in point form:

- Integrated Wi-Fi Connectivity: NodeMCU ESP8266 comes with built-in Wi-Fi connectivity, allowing seamless integration with local networks and internet services. This enables weather monitoring systems to transmit data to online servers or cloud platforms for storage and analysis.

- Low-Cost Solution: NodeMCU ESP8266 boards are cost-effective, making them suitable for hobbyists, students, and professionals on a budget. This affordability factor enables widespread adoption in IoT projects, including weather monitoring applications.

- Arduino Compatibility: NodeMCU ESP8266 is compatible with the Arduino programming environment, offering a familiar development platform for Arduino enthusiasts. This compatibility simplifies the development process and allows users to leverage existing Arduino libraries and resources.

- GPIO Pins for Sensor Integration: NodeMCU ESP8266 boards feature General Purpose Input/Output (GPIO) pins, enabling easy integration of various sensors,

such as temperature, humidity, and atmospheric pressure sensors, essential for weather monitoring applications.

- Small Form Factor: NodeMCU ESP8266 boards have a compact form factor, making them suitable for deployment in space-constrained environments typical of weather monitoring systems, such as outdoor weather stations or indoor weather monitoring devices.

- Real-Time Data Processing: The ESP8266 microcontroller on NodeMCU boards provides sufficient processing power for real-time data processing and analysis, allowing weather monitoring systems to generate insights and respond to environmental changes promptly.

- Web Server Capabilities: NodeMCU ESP8266 can function as a web server, enabling users to access weather data through web browsers or mobile devices. This feature facilitates remote monitoring and control of weather monitoring systems over local networks or the internet.

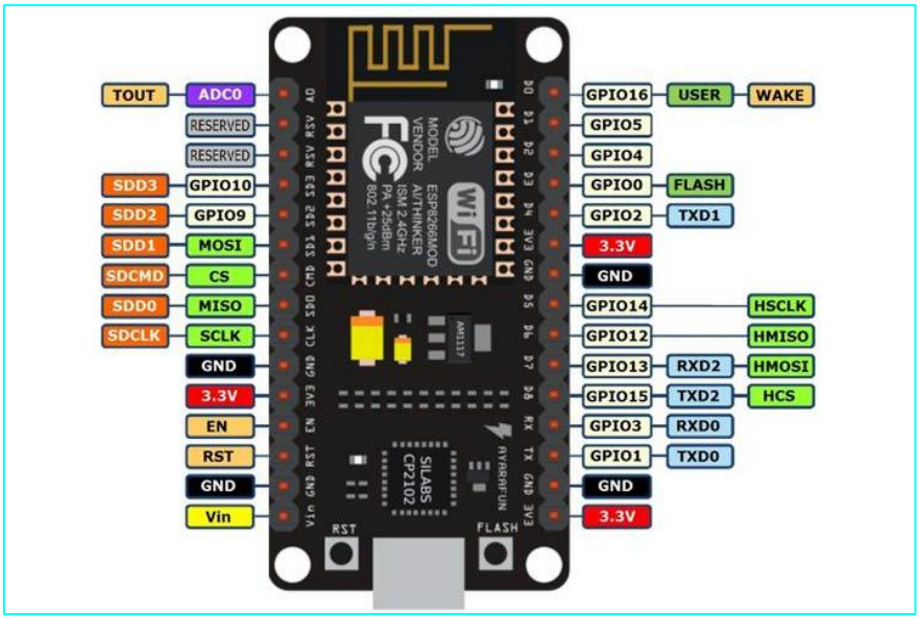

Fig 2.10: NodeMCU Diagram

2.5.2 DHT-11

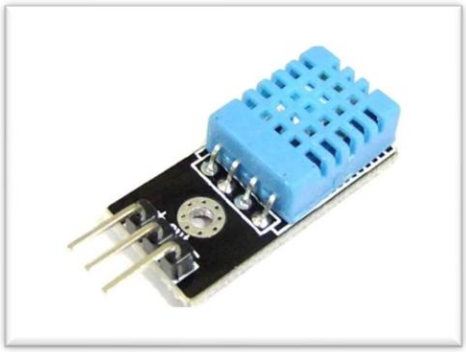

Fig 2.11: DHT-11 Sensor

- Temperature and Humidity Sensing: The DHT11 sensor is capable of measuring both temperature and humidity levels in the surrounding environment, making it suitable for weather monitoring, home automation, and greenhouse applications.

- Low Cost: DHT11 sensors are affordable and readily available, making them accessible to hobbyists, students, and professionals working on budget-constrained projects.

- Digital Output: The sensor provides digital output, simplifying interfacing with microcontrollers like Arduino, Raspberry Pi, and NodeMCU. It communicates data using a single-wire digital signal protocol.

- Wide Operating Voltage Range: The sensor operates within a wide voltage range (typically 3.3V to 5V), making it compatible with various microcontroller platforms without requiring additional level-shifting circuitry.

- Decent Accuracy: While not as precise as some higher-end sensors, the DHT11 offers decent accuracy for many applications. It typically has an accuracy of ±2°C for temperature measurements and ±5% for humidity measurements.

- Low Power Consumption: DHT11 sensors have low power consumption, making them suitable for battery-powered applications where energy efficiency is crucial.

2.5.3 BMP-180

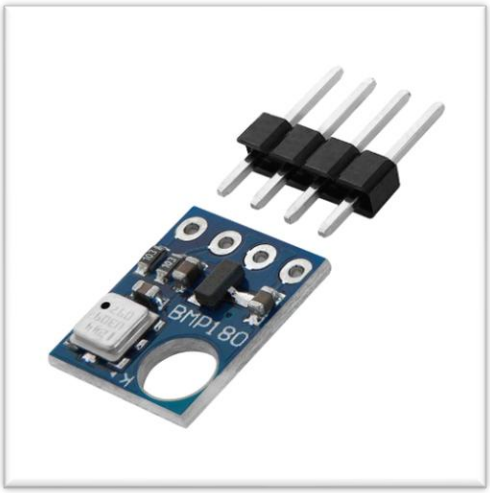

Fig 2.12: BMP-180 Sensor

- Pressure and Temperature Sensing: The BMP180 sensor is designed to measure both atmospheric pressure and temperature, making it suitable for weather forecasting, altitude sensing, and indoor climate control applications.

- High Precision: It offers high precision in pressure measurements, with an accuracy of ±1 mb (millibar) and a resolution of 0.01 mb. Temperature measurements have an accuracy of ±1°C.

- Low Power Consumption: The BMP180 sensor has low power consumption, making it suitable for battery-powered applications where energy efficiency is essential.

- I2C Interface: It communicates with microcontrollers via the I2C (Inter-Integrated Circuit) protocol, simplifying interfacing with popular platforms like Arduino, Raspberry Pi, and ESP8266.

- Compact Design: The sensor comes in a compact form factor, enabling easy integration into small-scale projects and devices with limited space.

2.5.4 LM-393 LDR

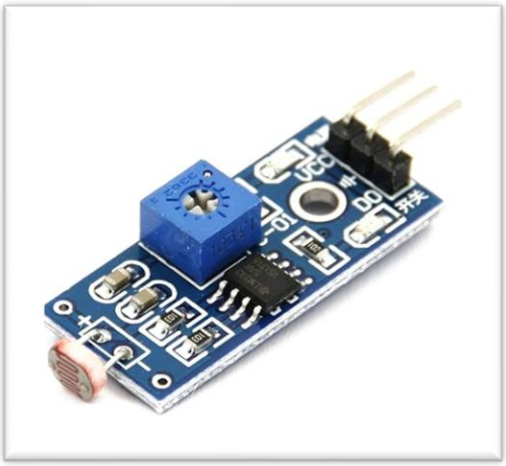

Fig 2.13: LM393 LDR Sensor Module

- Light Sensitivity: The LM393 light-dependent control sensor utilizes a photodiode or a phototransistor to detect changes in ambient light levels.

- Comparator Circuit: Integrated with LM393 comparator IC, it compares the output voltage of the light-sensitive element with a reference voltage, triggering a response when light levels surpass a predetermined threshold.

- Adjustable Sensitivity: Users can adjust the sensitivity of the sensor by varying the reference voltage or through external components like resistors, allowing customization for specific light conditions.

- Applications: Commonly employed in light-activated switches, streetlights, dusk-to-dawn lamps, and other automated systems requiring light-dependent control.

- Reliability and Cost-effectiveness: LM393-based light-dependent sensors offer a balance of reliability, simplicity, and affordability, making them popular choices in various consumer and industrial applications.

2.5.5 FC-37 Sensor Module

Fig 2.14: FC-37 Sensor Module

- Rain Sensing: The FC-37 rain sensor is designed to detect the presence of raindrops or moisture, making it suitable for weather monitoring systems, irrigation control, and rainwater harvesting.

- Analog Output: It provides analog output proportional to the intensity of rainfall, allowing for precise measurement and analysis of rainfall levels.

- Adjustable Sensitivity: The sensitivity of the sensor can be adjusted to detect various levels of rainfall, making it adaptable to different environmental conditions and applications.

- Simple Interface: With a straightforward interface, the FC-37 rain sensor can be easily connected to microcontrollers like Arduino or Raspberry Pi for data collection and processing.

- Durable Construction: The sensor is typically constructed with corrosion-resistant materials, ensuring durability and reliability in outdoor environments.

2.5.6 BreadBoard & Jumper Wires

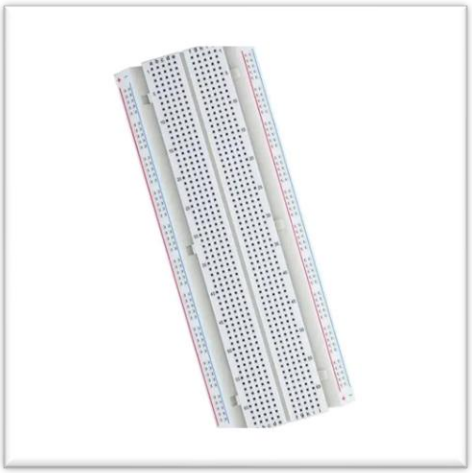

Fig 2.15: Breadboard

- A breadboard is a prototyping tool used to create temporary circuits for testing and experimenting with electronic components. It consists of a grid of holes connected by metal strips, allowing components to be easily inserted and connected without soldering. It's commonly used in electronics projects for its versatility and convenience.

Fig 2.16: Jumper Wires

- Jumper wires are flexible conductive wires used to create electrical connections between components on a breadboard or other electronic circuit platforms. They come in various lengths and colors, making it easy to organize and troubleshoot circuits. Jumper wires facilitate rapid prototyping and experimentation in electronics projects.

2.6 IOT TECHNOLOGIES (SOFTWARE)

2.6.1 C++

Fig 2.17: C++

- Efficiency: C++ is known for its performance and efficiency, which is crucial in IoT projects where resources like memory and processing power are often limited. NodeMCU boards typically have constrained resources, and C++ allows developers to write code that maximizes efficiency.

- Platform Compatibility: C++ is widely supported across different platforms, including NodeMCU boards. This ensures that code written in C++ can be easily ported and run on these IoT devices without much modification.

- Arduino Framework: Many NodeMCU boards are programmed using the Arduino framework, which is based on C++. This framework provides a convenient and beginner-friendly environment for developing IoT applications, making it easier for developers to get started with NodeMCU projects.

- Access to Low-Level Hardware: C++ allows direct access to low-level hardware features, which is essential for IoT projects where interacting with sensors, actuators, and other peripherals is common. This level of control enables developers to optimize code for specific hardware configurations and requirements.

2.6.2 C++ Libraries

Fig 2.18: C++ Libraries

- NodeMCU ESP8266 Board Manager: This package integrates ESP8266-based NodeMCU support into the Arduino IDE, allowing developers to compile and upload C++ code tailored for Wi-Fi-enabled microcontroller boards, essential for building IoT applications requiring wireless communication and remote data transmission.

- DHT-11 Library: The DHT-11 library facilitates accurate reading of temperature and humidity values from the DHT11 sensor using a timing-based digital signal protocol. It simplifies implementation by handling signal timing, making it ideal for basic environmental monitoring applications in embedded systems.

- BMP-180 Library: This library provides functions to interface with the BMP180 sensor over I2C, enabling the reading of atmospheric pressure and temperature data. It handles calibration and conversions, supporting applications like altitude estimation and weather-based automation in Arduino projects.

- Adafruit Sensor Library: The Adafruit Sensor Library acts as a unified sensor interface, standardizing how data is accessed from various Adafruit sensors. It enhances code compatibility, modularity, and simplifies integration by providing consistent methods for obtaining sensor readings across different hardware types.

2.7 IOT TECHNOLOGIES (CLOUD SERVICE), ThingSpeak

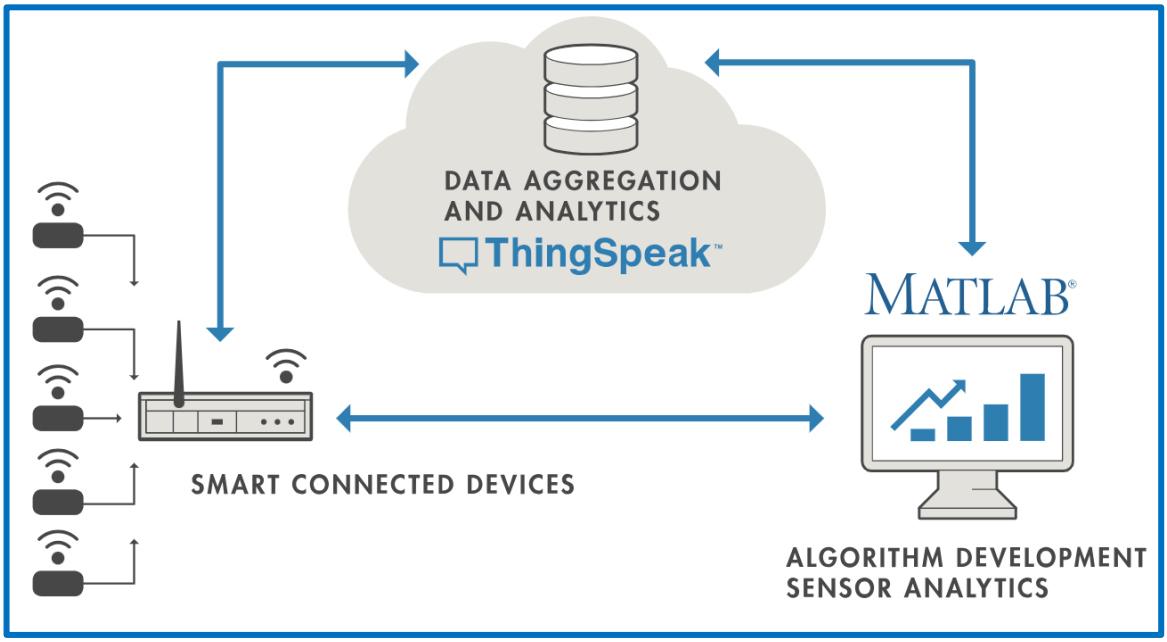

Fig 2.19: ThingSpeak

- Data Collection: ThingSpeak allows IoT devices, like weather sensors, to upload data to the cloud in real-time, facilitating continuous monitoring of weather conditions such as temperature, humidity, and atmospheric pressure.

- Visualization: It provides built-in tools for visualizing data, including graphs, charts, and gauges, enabling users to easily analyze and interpret weather data trends over time.

- Data Logging: ThingSpeak offers data logging capabilities, storing historical weather data securely in the cloud. This allows users to access past weather records for analysis, forecasting, and trend prediction.
- Customizable Dashboards: Users can create custom dashboards to display weather data in a format that suits their preferences and requirements. This customization

allows for a personalized and intuitive user experience.

- Alerting and Notifications: ThingSpeak supports alerting and notification features based on predefined thresholds or conditions. Users can receive alerts via email, SMS, or other communication channels when weather conditions exceed specified limits.

- Integration with IoT Devices: ThingSpeak integrates seamlessly with a wide range of IoT devices and platforms, including Arduino, Raspberry Pi, and NodeMCU. This compatibility simplifies the process of connecting weather sensors to the cloud and enables rapid deployment of weather monitoring systems.

- API Access: ThingSpeak offers an API (Application Programming Interface) for developers to access and manipulate weather data programmatically. This allows for seamless integration with third-party applications, services, and analytics platforms.

- Community Support and Collaboration: ThingSpeak benefits from a vibrant community of users and developers who contribute resources, tutorials, and support. This collaborative ecosystem fosters innovation and knowledge sharing in the development of weather monitoring systems and other IoT applications.

- Scalability: ThingSpeak is designed to scale according to the needs of users, accommodating large-scale deployments of weather monitoring systems with ease. Whether monitoring a single location or multiple sites, ThingSpeak can handle the data storage and processing requirements efficiently.

CHAPTER 3

IOT METHODOLOGY

3.1 IOT CIRCUIT DIAGRAM AND HARDWARE SETUP

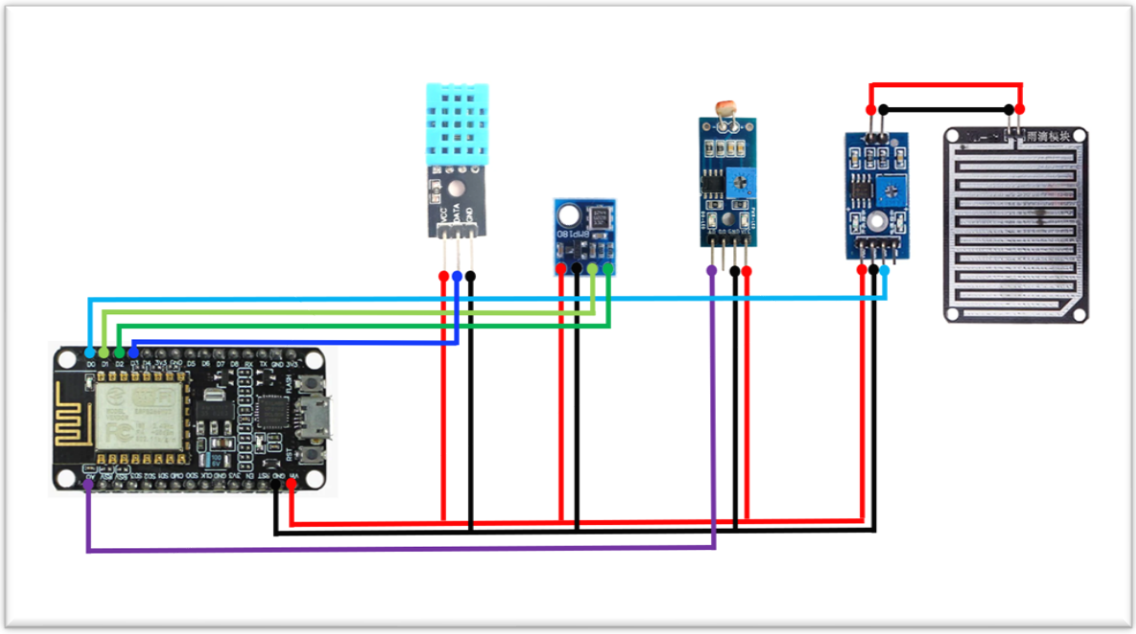

Fig 3.1: Circuit Diagram of IOT System

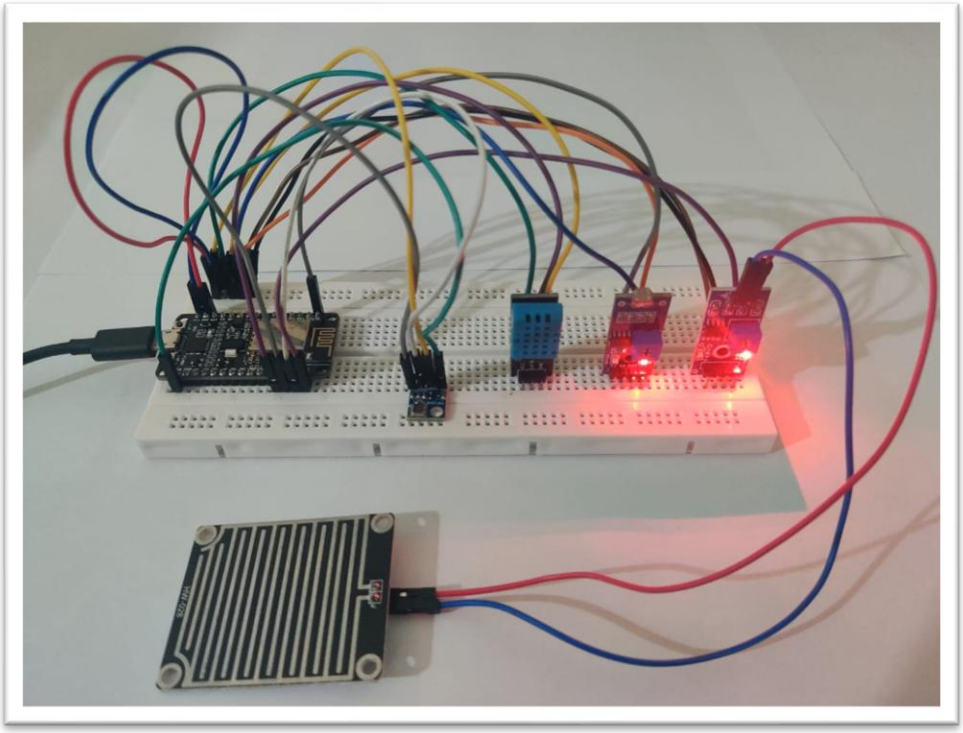

Fig 3.2: Hardware setup of IOT System

Step 1: These components

- NodeMCU
- DHT11
- BMP180
- LM393 LDR Module
- FC-37

are fitted into the breadboard and wiring is done in the following ways

1. One end of male to male jumper wire is connected to Vin of NodeMCU and other end to (+) of breadboard.
2. One end of male to male jumper wire is connected to GND of NodeMCU and other end to (-) of breadboard.
3. One end of male to male jumper wire is connected to VCC of DHT11 and other end to (+) of breadboard.
4. One end of male to male jumper wire is connected to GND of DHT11 and other end to (-) of breadboard.
5. One end of male to male jumper wire is connected to DATA of DHT11 and other end to D3 of NodeMCU.
6. One end of male to male jumper wire is connected to VCC of BMP180 and other end to (+) of breadboard.
7. One end of male to male jumper wire is connected to GND of BMP180 and other end to (-) of breadboard.
8. One end of male to male jumper wire is connected to SCL of BMP180 and other end to D1 of NodeMCU.
9. One end of male to male jumper wire is connected to SDA of BMP180 and other end to D2 of NodeMCU.
10. One end of male to male jumper wire is connected to AO of LM393 LDR Module and other end to A0 of NodeMCU.
11. One end of male to male jumper wire is connected to VCC of LM393 LDR Module and other end to (+) of breadboard.
12. One end of male to male jumper wire is connected to GND of LM393 LDR Module and other end to (-) of breadboard .

13. One end of male to male jumper wire is connected to VCC of FC-37 and other end to (+) of breadboard.
14. One end of male to male jumper wire is connected to GND of FC-37 and other end to (-) of breadboard.
15. One end of male to male jumper wire is connected to A0 of FC-37 and other end to A0 of NodeMCU.

PIN DESCRIPTIONS

- Vin: Voltage In, typically refers to the input voltage.
- A0 of NodeMCU: Analog Pin 0 of NodeMCU, used for analog input or output.
- D0 of NodeMCU: Digital Pin 0 of NodeMCU, used for digital input or output.
- D1 of NodeMCU: Digital Pin 1 of NodeMCU, used for digital input or output.
- D2 of NodeMCU: Digital Pin 2 of NodeMCU, used for digital input or output.
- D3 of NodeMCU: Digital Pin 3 of NodeMCU, used for digital input or output.
- GND: Ground, the reference point for electrical circuits.
- VCC: Voltage (Positive) Common Collector, provides the supply voltage for the component.
- DATA of DHT11: Data pin of the DHT11, used to send temperature and humidity data.
- SCL of BMP180: Serial Clock, used for synchronized communication in I2C protocol with BMP180.
- SDA of BMP180: Serial Data, used for bidirectional data transfer in I2C protocol with BMP180.
- A0 of LM393 LDR Module: Analog pin of the LM393 LDR Module, used to send analog data.
- D0 of FC-37: Data pin of the FC-37, used to send rainfall data.

Step 2: Micro-usb cable is connected between the PC and the NodeMCU

3.2 SOFTWARE SETUP

Step 1: A account on is created on ThingSpeak website.

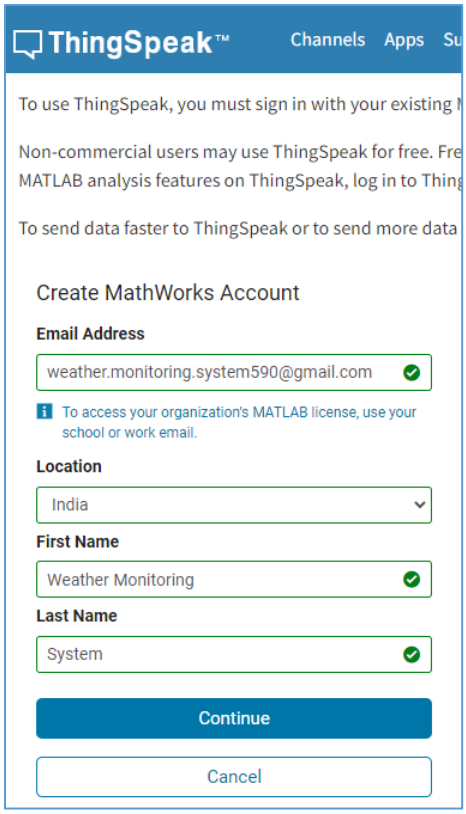

Fig 3.3: ThingSpeak account creation

Step 2: A new channel is created with the following field details and then title of each graph is changed accordingly.

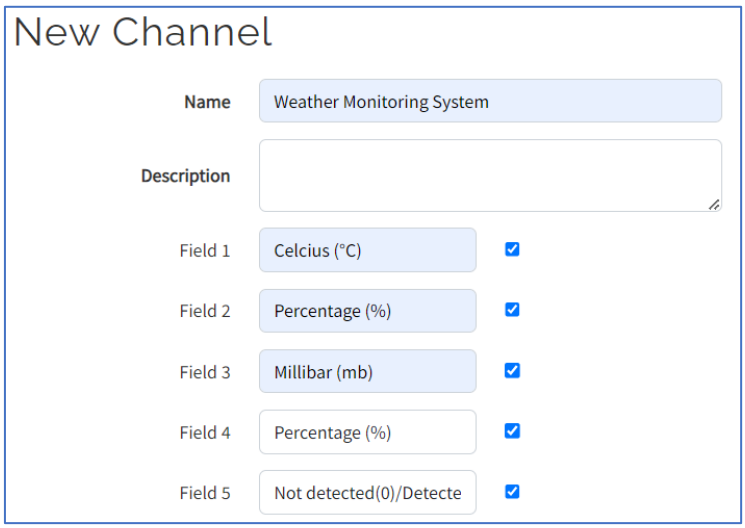

Fig 3.4: ThingSpeak channel creatio

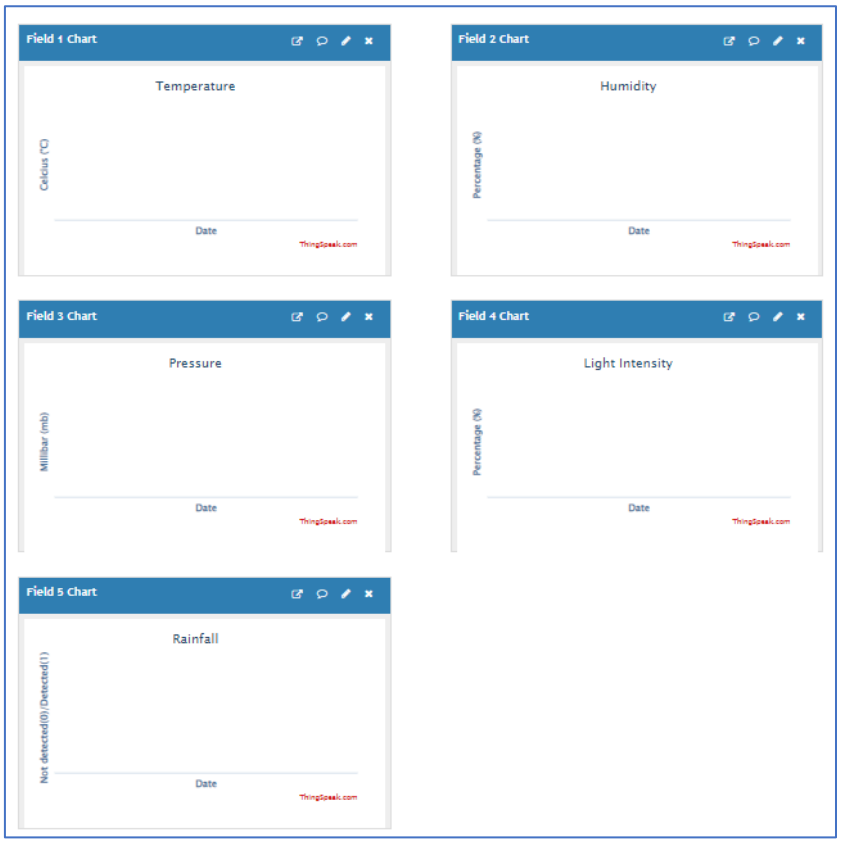

Fig 3.5: ThingSpeak private view

Step 3: Under API keys tab, the key is recorded.

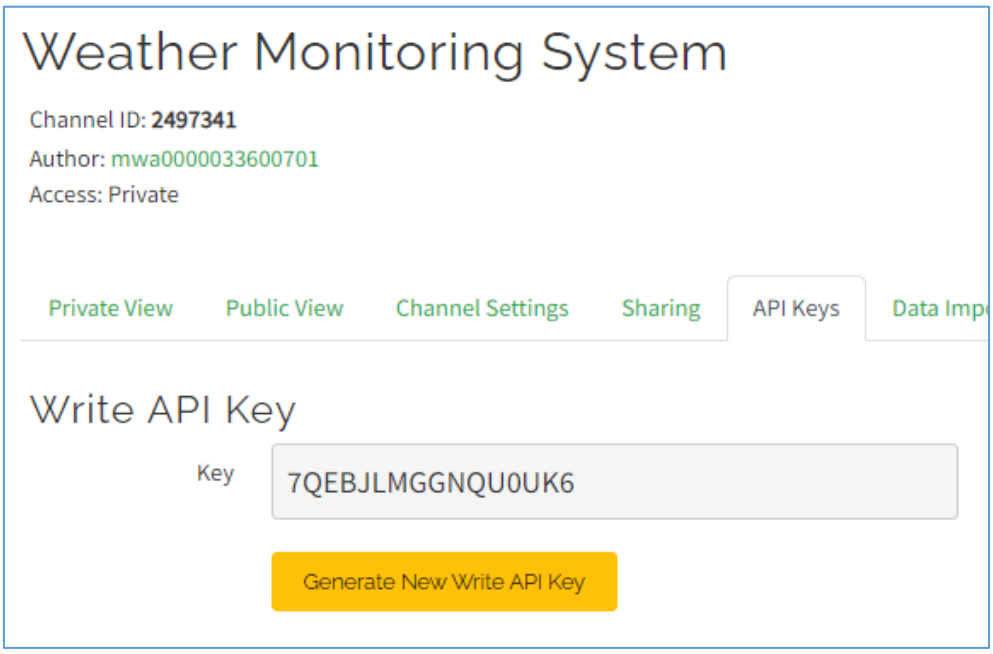

Fig 3.6: API key of ThingSpeak

Step 3: A new sketch is created, the required code is written and the file is saved as "weather_monitoring_system.ino".

Step 4: The link "https://arduino.esp8266.com/stable/package_esp8266com_index.json" is pasted in additional board manager URLs under preferences settings.

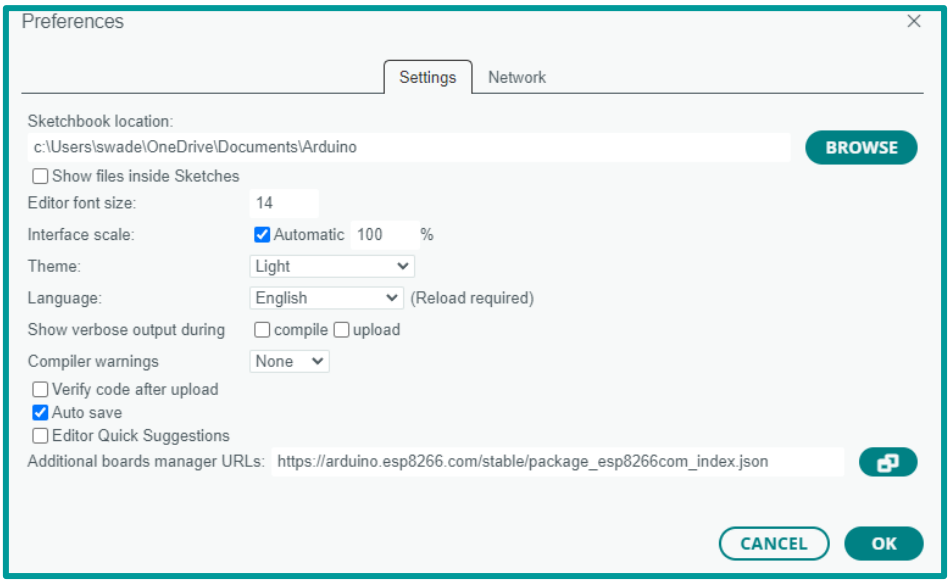

Fig 3.7: Preferences settings of Arduino IDE

Step 5: esp8266 is installed under board manager tab.

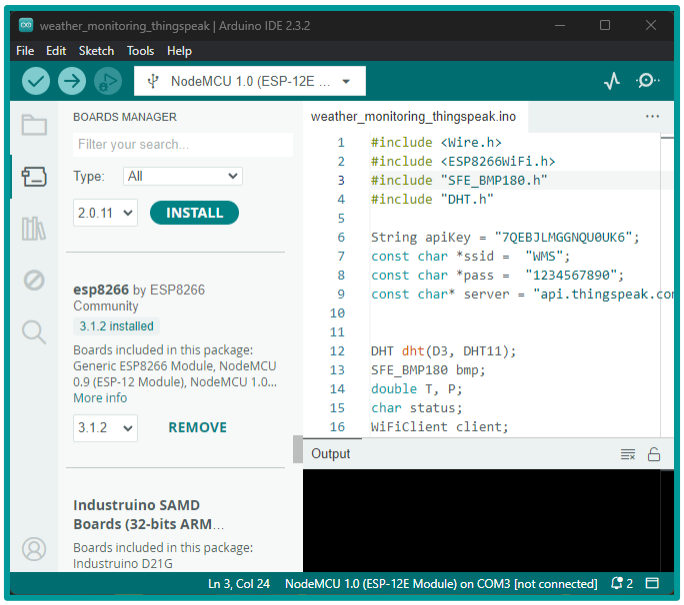

Fig 3.8: Board manager section of Arduino IDE

Step 6: After writing the required code it is compiled and the following result in seen in the output section.

```
. Variables and constants in RAM (global, static), used 29280 / 80192 bytes (36%)
╠═ SEGMENT  BYTES    DESCRIPTION
╠═══ DATA     1516     initialized variables
╠═══ RODATA   1260     constants
╚═══ BSS      26504    zeroed variables
. Instruction RAM (IRAM_ATTR, ICACHE_RAM_ATTR), used 60995 / 65536 bytes (93%)
╠═ SEGMENT  BYTES    DESCRIPTION
╠═══ ICACHE   32768    reserved space for flash instruction cache
╚═══ IRAM     28227    code in IRAM
. Code in flash (default, ICACHE_FLASH_ATTR), used 253860 / 1048576 bytes (24%)
╠═ SEGMENT  BYTES    DESCRIPTION
╚═══ IROM     253860   code in flash
```

Step 7: The code is uploaded to NodeMCU through a micro USB cable and the following result is seen in the output section.

```
esptool.py v3.0
Serial port COM3
Connecting....
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 08:f9:e0:71:04:16
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 289024 bytes to 211562...
Writing at 0x00000000... (7 %)
Writing at 0x00004000... (15 %)
Writing at 0x00008000... (23 %)
Writing at 0x0000c000... (30 %)
```

Writing at 0x00010000... (38 %)

Writing at 0x00014000... (46 %)

Writing at 0x00018000... (53 %)

Writing at 0x0001c000... (61 %)

Writing at 0x00020000... (69 %)

Writing at 0x00024000... (76 %)

Writing at 0x00028000... (84 %)

Writing at 0x0002c000... (92 %)

Writing at 0x00030000... (100 %)

Wrote 289024 bytes (211562 compressed) at 0x00000000 in 18.4 seconds (effective 125.5 kbit/s)...

Hash of data verified.

Leaving...

Hard resetting via RTS pin...

Then in serial monitor section the following output is seen.

```
....
WiFi connected
Temperature: 38.00
Humidity: 29.00
Pressure: 1002.24
Light: 1
Rain: Not Detected
Temperature: 38.00
Humidity: 29.00
Pressure: 1002.23
Light: 1
Rain: Not Detected
Temperature: 37.00
Humidity: 30.00
Pressure: 1002.26
Light: 1
Rain: Not Detected
```

Finally, under the private view of ThingSpeak website the real-time results can be seen.

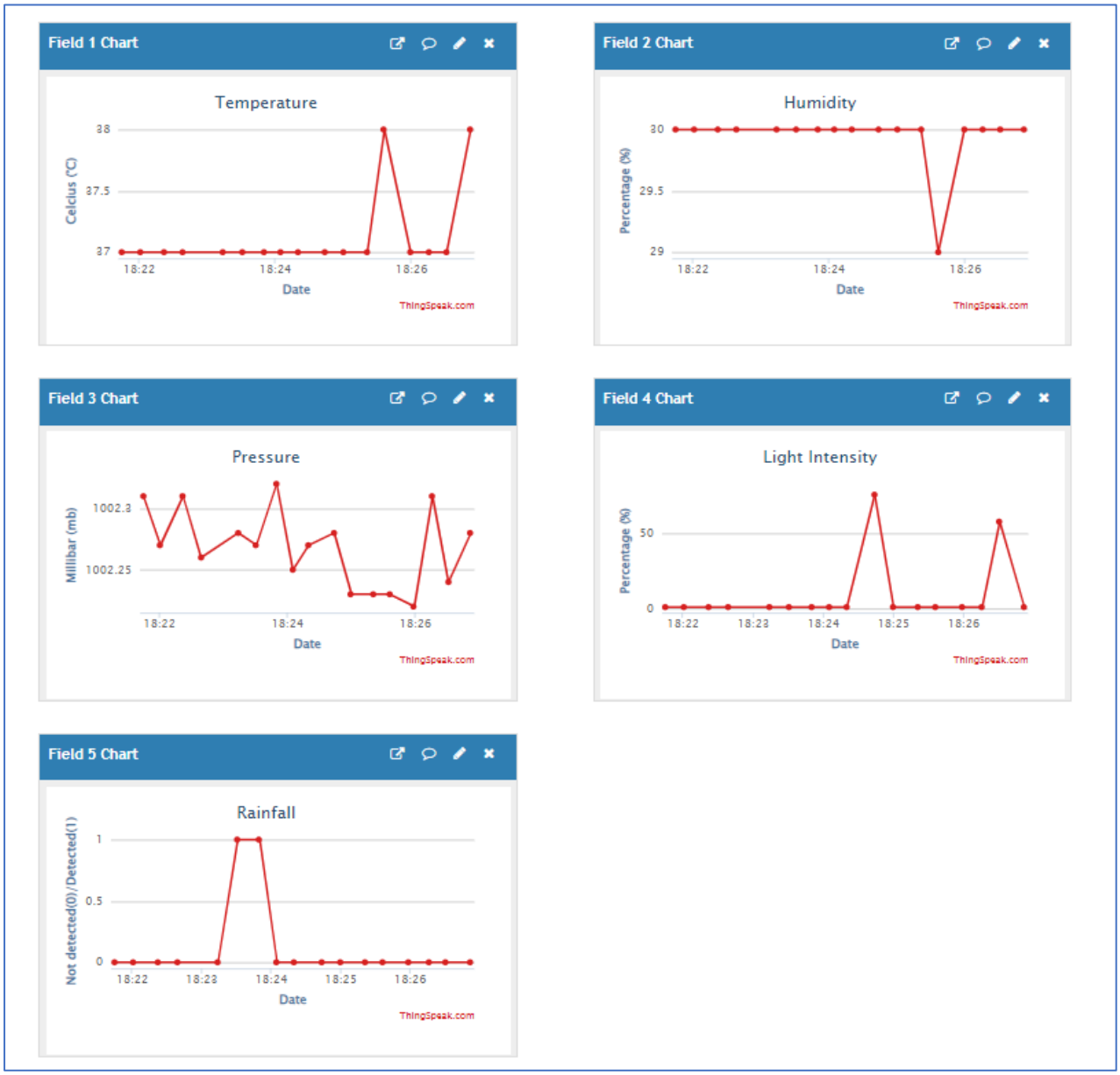

Fig 3.9: Final results on ThingSpeak

CHAPTER 4

WEBSITE METHODOLOGY

4.1 GETTING THE API KEY FROM WEATHERAPI.COM

After ending of trial version of 14 days, the api will work but it will still give complete current data with 3 days forecast data, no air quality forecast data and 3 days astro data instead of 14 days forecast data, 4/5 days air quality forecast data and 14 days astro data.

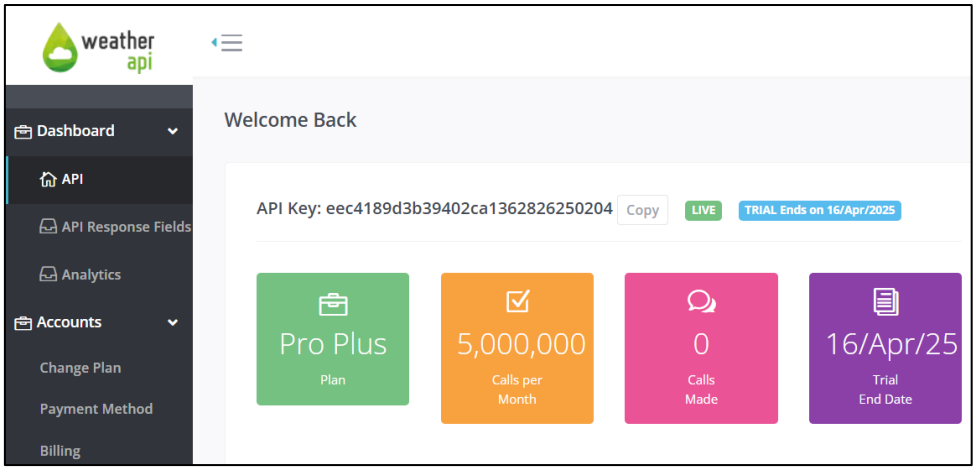

Fig: 4.1: weatherapi.com

4.2 SETTING THE BACKEND ENVIRONMENT

```
{
  "name": "lws-backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.5.0",
    "express": "^5.1.0",
    "https-localhost": "^4.7.1"
  },
  "type": "module"
}
```

4.3 SETTING THE FRONTEND ENVIRONMENT

```
{
  "name": "lws-frontend",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "next dev --turbopack",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "export": "next export"
  },
  "dependencies": {
    "@reduxjs/toolkit": "^2.6.1",
    "express": "^4.21.2",
    "next": "15.3.0",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-loader-spinner": "^6.1.6",
    "react-redux": "^9.2.0",
    "react-select": "^5.10.1",
    "react-toastify": "^11.0.5",
    "react-top-loading-bar": "^3.0.2",
    "world-countries": "^5.1.0"
  }
}
```

4.4 SETTING THE CONFIG FILE

Before building the frontend, updating the config file is must.

```
/** @type {import('next').NextConfig} */
const nextConfig = {
    reactStrictMode: false,
    // trailingSlash: true,
    output: "export",
    images: {
        unoptimized: true // <-- if you're using <Image> and need static export
compatibility
    },
};

export default nextConfig;
```

4.5 BUILDING THE FRONTEND CODE

After typing “npm run build” in console buidling process get started.

This is Next Js App router version.

This frontend is not ssr (server side rendering). This is csr (client side rendering) and ssg (static side generation).

```
> lws-frontend@1.0.0 build
> next build

   ▲ Next.js 15.3.0
   - Environments: .env

   Creating an optimized production build ...
 ✓ Compiled successfully in 3.0s
 ✓ Linting and checking validity of types
 ✓ Collecting page data
 ✓ Generating static pages (9/9)
 ✓ Collecting build traces
 ✓ Exporting (3/3)
 ✓ Finalizing page optimization

Route (app)                                 Size  First Load JS
┌ ○ /                                      364 B         102 kB
├ ○ /_not-found                            977 B         103 kB
├ ○ /about                                 423 B         112 kB
├ ○ /blog                                3.33 kB         115 kB
├ ○ /forecast                            10.9 kB         122 kB
└ ○ /station                             6.32 kB         118 kB
+ First Load JS shared by all             102 kB
  ├ chunks/4bd1b696-da872c362ff0e0b5.js  53.2 kB
  ├ chunks/684-20fb8f525df4f8bf.js         46 kB
  └ other shared chunks (total)          2.67 kB


○  (Static)  prerendered as static content
```

4.6 SERVING FRONTEND VIA BACKEND

Backend not only acts as a middleman for calling api request to weatherapi.com but also used to serve the frontend code to the client

---

```
import express from "express"
import cors from "cors"
import path from "path";
import { fileURLToPath } from "url";
import dotenv from 'dotenv';
import { getDataController } from "./controllers/getDataController.js";

const app = express()
const port = 3001
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
// const host = "192.168.29.180";
const host = "localhost";


dotenv.config();
app.use(cors())
app.use(express.json())

const frontendPath = path.join(__dirname, "..", "frontend", "out");
console.log(frontendPath);
app.use(express.static(frontendPath));

app.get('/about', (req, res) => {
  res.sendFile(path.join(frontendPath, 'about.html'));
});

app.post('/getdata', getDataController)

app.get('/:any', (req, res) => {
  res.sendFile(path.join(frontendPath, 'index.html'));
});

app.listen(port,host,() => {
  console.log("\n===== LIVECAST WEATHER STATION =====");
  console.log("☑ Server is up and running!");
  console.log(`🌐 http://${host}:${port}`);
  console.log("\n");
});
```

4.7 WEBSITE RESULTS

Fig: 4.2: Website Frontend

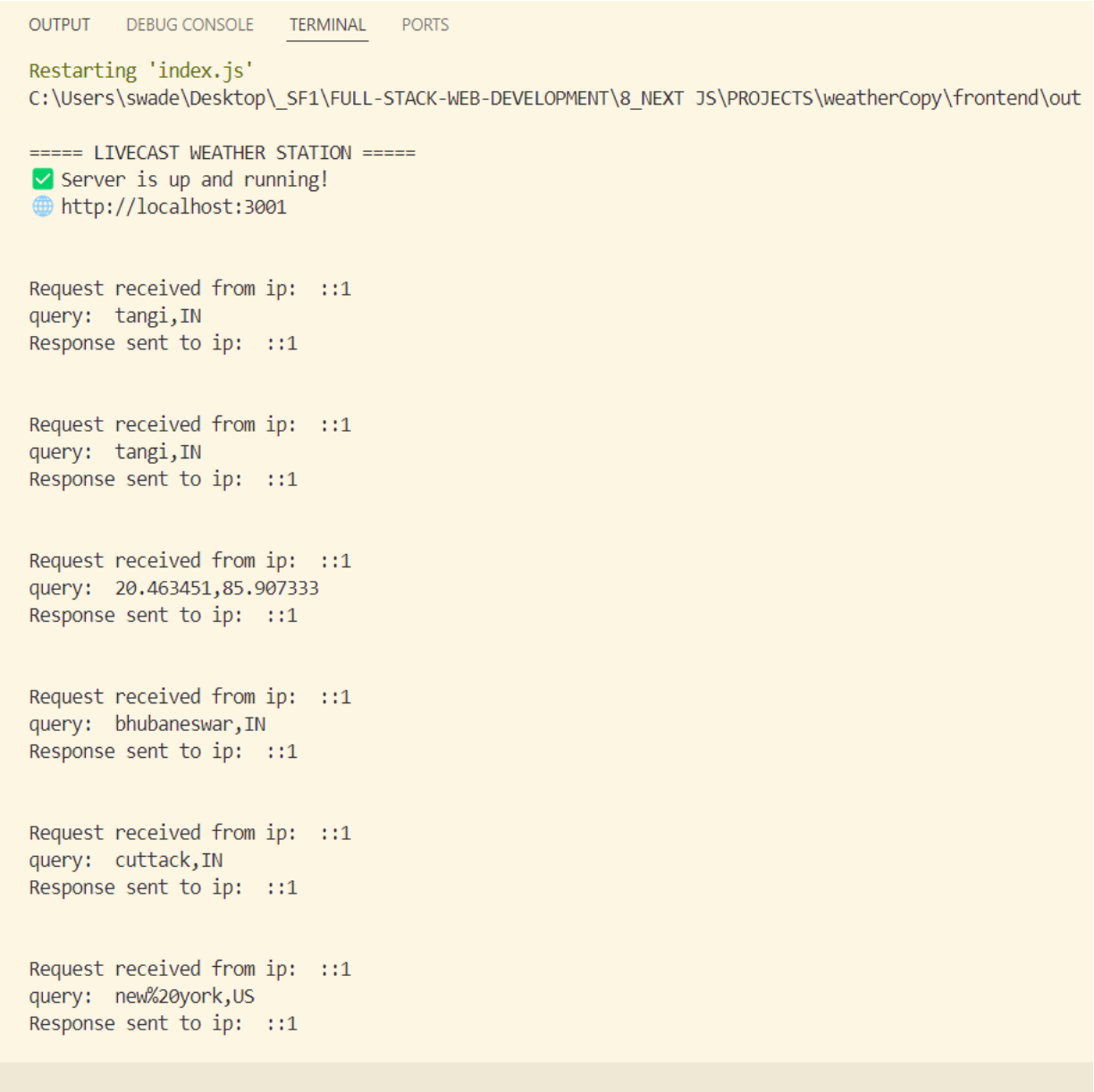

Fig: 4.3: Website Backend

4.8 SYSTEM ARCHITECTURE DIAGRAM

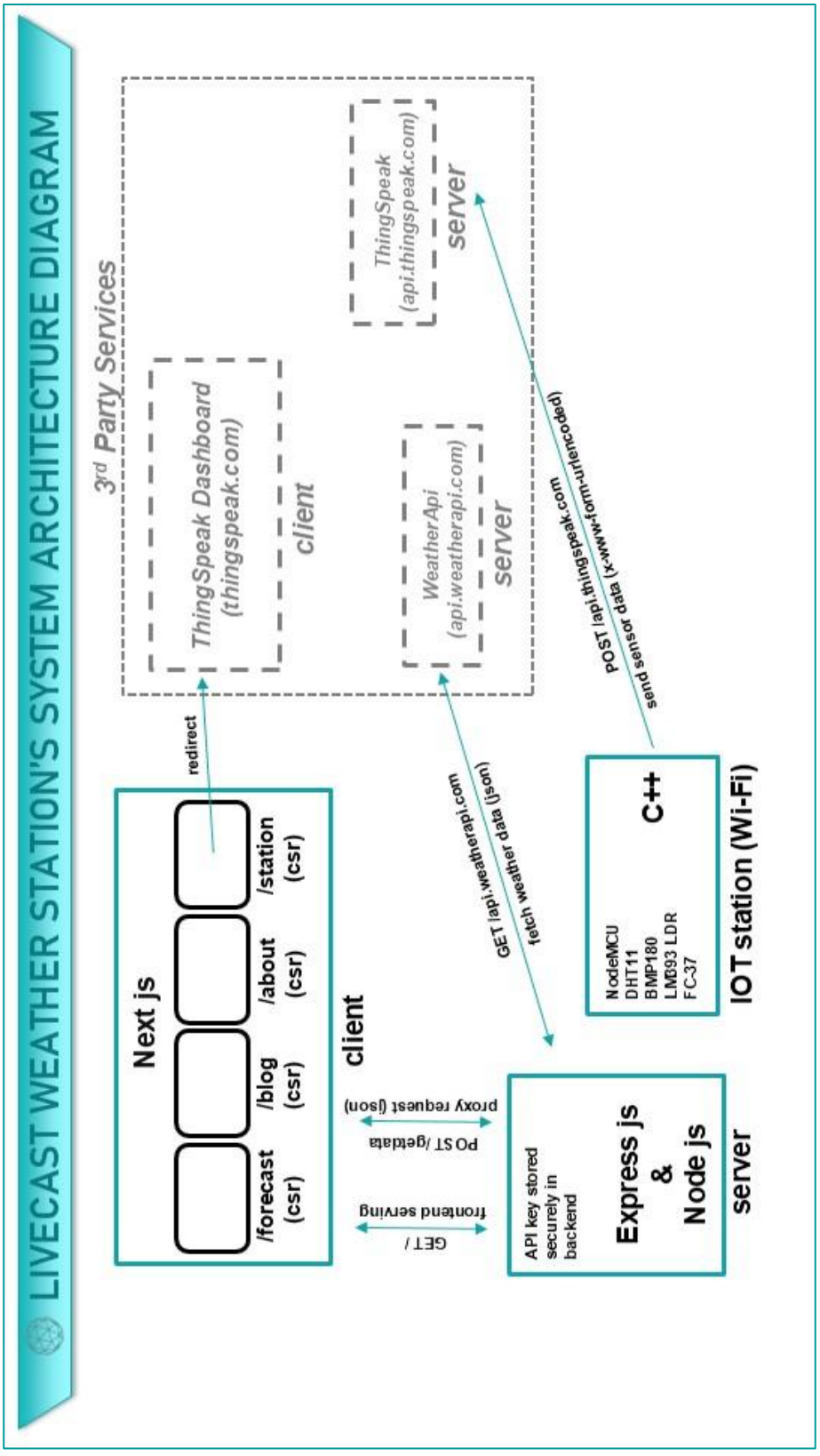

Fig: 4.4: System Architecture Diagram

CONCLUSION

The "LiveCast Weather Station" project effectively combines web technologies and IoT to deliver a reliable, real-time weather monitoring system tailored for agriculture and smart farming. By integrating WeatherAPI’s global data with localized sensor inputs, it provides accurate, location-specific information to support better decisions in irrigation, crop management, and farm planning.

Key sensors such as DHT11, BMP180, LDR, and FC-37 rain sensor—controlled by a NodeMCU—monitor vital parameters like temperature, humidity, pressure, light intensity, and rainfall. This data is continuously uploaded to ThingSpeak, enabling users to visualize trends and analyze environmental patterns for more informed planning.

Built using Next.js for the frontend and Node.js with Express.js for the backend, the platform ensures scalability, performance, and user accessibility. Its open-source, modular architecture supports future improvements like mobile alerts, AI predictions, and multi-language support.

Overall, the project demonstrates the practical application of IoT and web development in agriculture, promoting smart, data-driven, and sustainable farming practices.

FUTURE SCOPE

- We plan to replace ThingSpeak with our custom cloud dashboard, offering improved real-time data visualization, interactive graphs, and better control over sensor data storage and analytics.
- After deploying the custom dashboard, a dedicated mobile application will be developed to provide easy access to weather data, sensor readings, and notifications for farmers on the go.
- Additional sensors like soil moisture, soil pH, and UV index sensors will be integrated to enhance the system's capability in providing more comprehensive agricultural and environmental insights.
- SMS alert functionality will be added to notify farmers instantly about critical weather changes, sensor warnings, or environmental conditions, ensuring timely actions can be taken to protect crops.

REFERENCES

1. Deepanjan Biswas, Asis Mondal, Suvadip Ghosal, Bikramjit Prasad Ghosal. IoT-Based Real-Time Weather Monitoring and Reporting System. RCC INSTITUTE OF INFORMATION TECHNOLOGY.

2. R. Suresh Babu, Palaniappan Thillainathan. IoT Based Weather Monitoring System. ResearchGate, ISSN2394-3777 (Print), ISSN2394-3785 (Online).

3. Girija C, Andreanna Grace Shires. Internet of Things (IOT) based Weather Monitoring System. International Journal of Engineering Research & Technology (IJERT). ISSN: 2278-0181.

4. Dhannjay Verma, Ishan Choudhury, Manish Singh, Abhijeet Shukla, Dharendra Kumar. An IOT Based Weather Monitoring System. © 2019 JETIR April 2019, Volume 6, Issue 4. ISSN-2349-5162.

5. Puja Sharma and Shiva Prakash. Real Time Weather Monitoring System Using Iot. Madan Mohan Malaviya University of Technology, Gorakhpur.

APPENDIX

SOURCE CODE (IOT)

1. weather_monitoring_system.ino

```
#include <Wire.h>
#include <ESP8266WiFi.h>
#include "SFE_BMP180.h"
#include "DHT.h"


String apiKey = "7QEBJLMGGNQU0UK6";
const char *ssid =  "WMS";
const char *pass =  "1234567890";
const char* server = "api.thingspeak.com";


DHT dht(D3, DHT11);
SFE_BMP180 bmp;
double T, P;
char status;
WiFiClient client;


void setup() {
  Serial.begin(9600);
  delay(10);
  bmp.begin();
  Wire.begin();
  dht.begin();
  pinMode(D0, INPUT);
  WiFi.begin(ssid, pass);


  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
}
```

```
void loop() {
  //BMP180 sensor
  status =  bmp.startTemperature();
  if (status != 0) {
    delay(status);
    status = bmp.getTemperature(T);

    status = bmp.startPressure(3);// 0 to 3
    if (status != 0) {
      delay(status);
      status = bmp.getPressure(P, T);
      if (status != 0) {

      }
    }
  }

  //DHT11 sensor
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }


  //Rain sensor
  int r = 0;
  if(digitalRead(D0)==LOW){
    r=1;
  }


  //Light sensor
  int l = analogRead(A0);
  l = map(l, 0, 1024, 100, 0);


  if (client.connect(server, 80)) {
    String postStr = apiKey;
    postStr += "&field1=";
    postStr += String(t);
```

```
    postStr += "&field2=";
    postStr += String(h);
    postStr += "&field3=";
    postStr += String(P, 2);
    postStr += "&field4=";
    postStr += String(l);
    postStr += "&field5=";
    postStr += String(r);
    postStr += "\r\n\r\n\r\n\r\n";

    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n\n\n");
    client.print(postStr);

    Serial.print("Temperature: ");
    Serial.println(t);
    Serial.print("Humidity: ");
    Serial.println(h);
    Serial.print("Pressure: ");
    Serial.println(P, 2);
    Serial.print("Light: ");
    Serial.println(l);
    Serial.print("Rain: ");
    Serial.println(r ? "Detected" : "Not Detected");

  }

  client.stop();
  delay(1000);
}
```