FOSSEE Summer Fellowship

Report on

Any Time Network (ATN)

Submitted by

Prasaanth K Y
II Year ECE

Dharun A P
II Year CSE

Griffin Annshual S
II Year, CSE

Guhan K
II Year, CSE

Deepak Prakash S
II Year, CSE

Sri Eshwar College of Engineering, Coimbatore

under the guidance of

Rajesh Kushalkar
Sr. Manager Open Source Hardware,
IIT BOMBAY

Prof. Kannan M.Moudgalya
Chemical Engineering Department,
IIT BOMBAY

July 2024

Acknowledgment

We would like to express our deepest gratitude to Prof. Kannan M. Moudgalya from the Department of Chemical Engineering at IIT Bombay for his visionary creation of the FOSSEE Fellowship program. This initiative has provided students from all over India with a remarkable opportunity to engage in meaningful research and development, and we are truly honoured to have been a part of it.

We are equally indebted to our ATN mentor, Mr. Rajesh Kushalkar for his unwavering support and profound knowledge. His mentorship throughout this research project has been nothing short of transformative, and his patience and clarity in explaining complex concepts have greatly enriched our understanding.His insights greatly helped in the development of our project.

We also want to extend our heartfelt thanks to our mentor, Mr. Pratik Bhosale. His guidance and insightful feedback have been invaluable throughout this fellowship. We deeply appreciate the time and effort he invested in helping us grow both academically and professionally.

We are incredibly fortunate to have had the opportunity to work on such an exciting and challenging project, and we are profoundly grateful for the support and encouragement we received from everyone involved.

INDEX

1.Introduction

ATN - Any Time Network is a team of five that aims in solving networkless situations during flood through customised self made network that can communicate in flood areas. This project involves hardware , software and networking technologies to provide a complete product that can receive requests and SOS from people affected by flood from ATN mobile application and carries the messages through ATN network to rescue and helping crews through ATN website.

1.1 Problem Statement:

- Cellular networks often fail during floods due to damage and power outages.
- These networks are centralised, relying on a hub to receive and transmit data.
- A single point of failure can disrupt communication across an entire area.
- Floods are critical situations where lives are at risk.
- The loss of communication in such scenarios can have severe consequences, leaving people isolated and unable to seek help.
- During floods, there is often no platform to track and manage the specific needs of affected people.
- Donors and rescue teams lack information about how many people need medical aid, food, clothing, and other essentials.
- Without proper coordination, donations may focus on one type of commodity, leading to a surplus of some items and a shortage of others.
- ATN (Any Time Network) aims to solve these problems by using advanced hardware, software, and networking technologies.

1.2 Solution:

- The main aim of ATN is to collect the requirements from the affected people and take it to the people who are willing to help them(rescue team) through a network made by LoRa technology in a mesh network structure.
- ATN Network connects to the people via mobile app by ATN mobile application that allows users to choose the type of requirements they need.

- Mobile applications communicate to the node (Client Station) , a hardware device that can communicate with the ATN network which transmits to Base Station , a mesh node that is implanted in the area.
- Base station communicate with the other mesh nodes and reaches main station that uploads the information to the ATN website
- ATN website sorts the requirements which can be seen by rescue crew and common people.
- Allowing common people to see the requirement needs , allows donated products to be more classified.

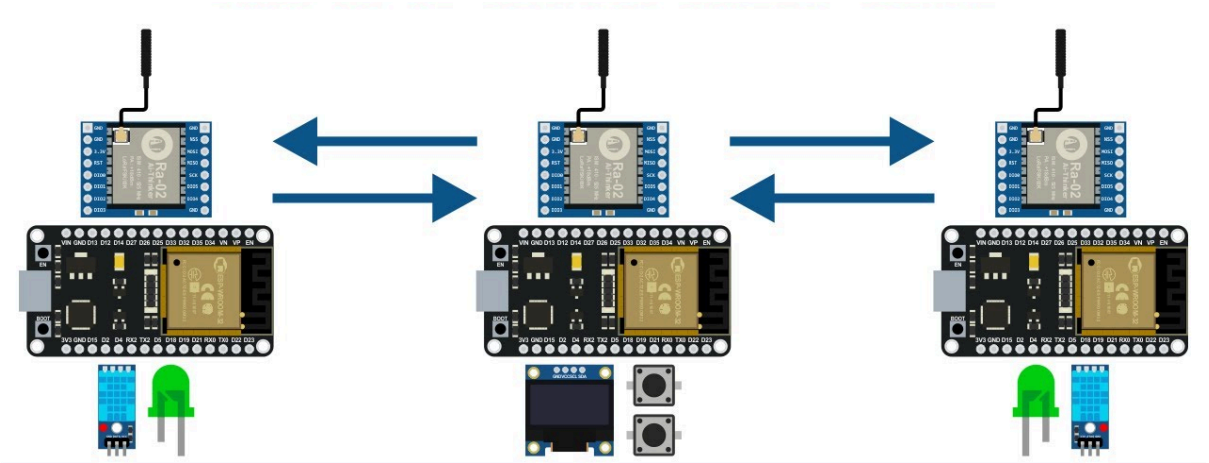
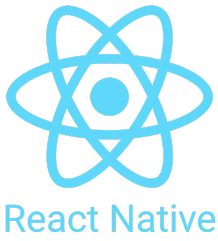


Fig 1.1 : Flow of data from one LoRa to Another

2.Mobile Application



2.1.Installation:

Dependencies:

- Node (Installing via Chocolatey).
- JDK.
- Android development environment.

Android studio setup:

- Download and install android studio.
  While installation make sure the boxes next to all of the following items are checked:
  I. Android SDK
  II. Android SDK Platform
  III. Android Virtual Device
- Install the Android SDK.
  React native requires `Android 14 (UpsideDownCake)` version in particular.
- Configure ANDROID_HOME environment variable that points to the path to your Android SDK.
- Add platform-tools to the path variable (default path- %LOCALAPPDATA%\Android\Sdk\platform-tools).

Code Setup:

- Download the code from git repository.
- Install the dependencies using command - npm install
- Connect the android device with the computer using an USBcable
- Turn on android debugging in the mobile device.
- Run the mobile app using the command - npm run android (or) npm run ios

2.2.Integration with NodeMCU (IP method):

Prerequisites:

- A mobile device with the app installed.
- A NodeMCU module connected to a Wi-Fi network(Mostly your mobile device).
- The IP address of the NodeMCU(will be displayed in the LCD screen).
- A Wi-Fi network for both the mobile device and NodeMCU to connect to.

Setting Up the NodeMCU:

1. **Flash the NodeMCU:** Ensure your NodeMCU is flashed with firmware that supports the HTTP server. You can use the Arduino IDE or another platform to upload your code.

2. **Configure the NodeMCU:**
    a. Connect the NodeMCU to your Wi-Fi network.
    b. Set up the HTTP server to listen for requests.

c. Program the NodeMCU to handle requests and control hardware (e.g., dispensing food or water).

3. **Find the IP Address:**

    a. Once connected, find the IP address of the NodeMCU. This is typically shown in figure 2.1 of the LCD screen in the component

Fig 2.1: IP address in LCD

Using the Mobile App:.

1. **Launch the App**: Open the app on your mobile device
2. **Enter IP Address**:
    - In the field labelled "NodeMCU IP Address," enter the IP address of your NodeMCU as shown in figure 2.2.
3. **Specify Items and Quantities**:
    - Enter the required amounts for items like food and water in the respective fields as shown in figure 2.3.
    - Double-check the quantities to ensure they are correct.
4. **Send Request**:
    - Press the "Submit" or "Send" button to send your request to the NodeMCU.
    - The app will make an HTTP request to the NodeMCU, which will then process the request and dispense the specified amounts.

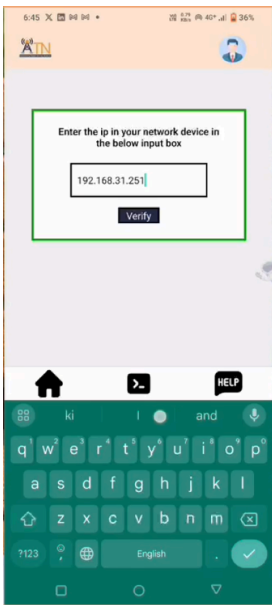


Fig 2.2: Filling the ip address

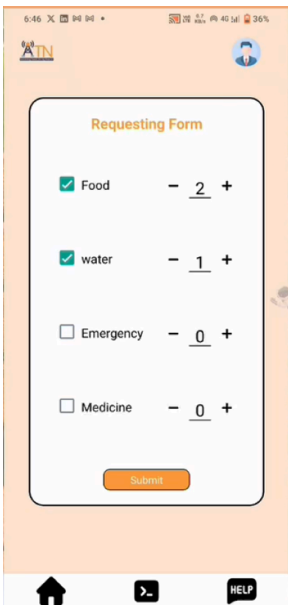


Fig 2.3: Specifying the required products

Example video:

IP method mobile app demo.mp4

2.3.Integration with NodeMCU (QR code method):

Prerequisites:

- Only prerequisite is to enable the WIFI in the user's mobile device. Then when we open our app a QR code scanner opens. User has to scan the QR code with the device that contains the WIFI credentials.
- Mobile connects to that WIFI-hotspot and a connection is set. With this connection we can send data from the mobile application

Sending Data:

Everything else is the same as the previous version. There will be a page to enter the user's requirements and when they click submit the data is sent to the web server in the NodeMCU through the WIFI connection. Then the LoRa network takes care of the data and sends it to the central station.

.OTA Updating Part

- Whenever firmware needs to be updated it is put in the server manually by the developers. It is found inside a folder called OTAcode as shown in the figure 2.4.
- On the request page whenever requests are fetched it sends the newest version number in the server.
- If the code in the mobile app is not up to date it requests for the code and gets it.
- So whenever a QR code is scanned it hits an endpoint in the ESPserver and gets the current version number in the ESP32
- After checking with the version returned, the app asks whether we should update ESP and once an update is clicked it updates the code in the ESP with the newer version. A user can also skip the updating part if required.

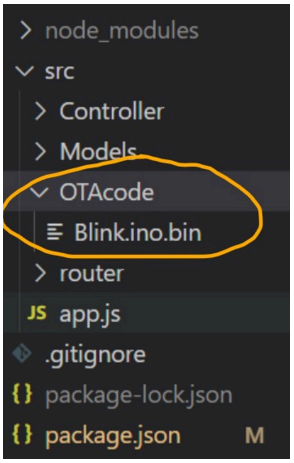


Fig 2.4: Updated code in the server for OTA

2.4 Flow of app for connecting with ESP32

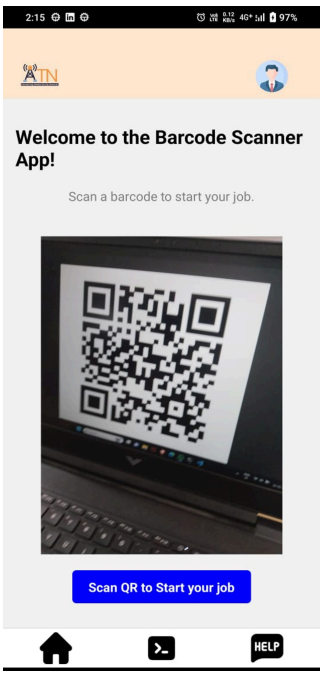


Fig 2.5: Scanning of the QR code

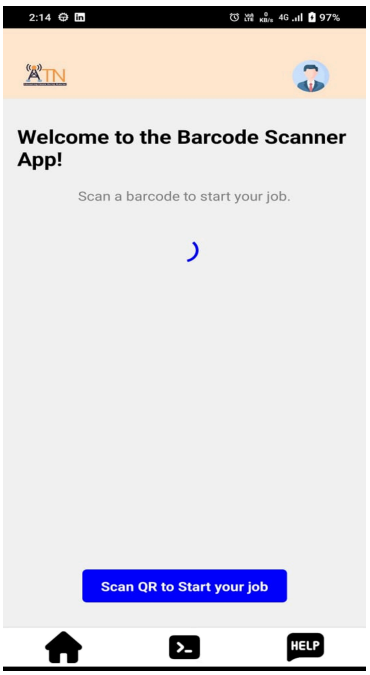


Fig 2.6: Connecting to WiFi

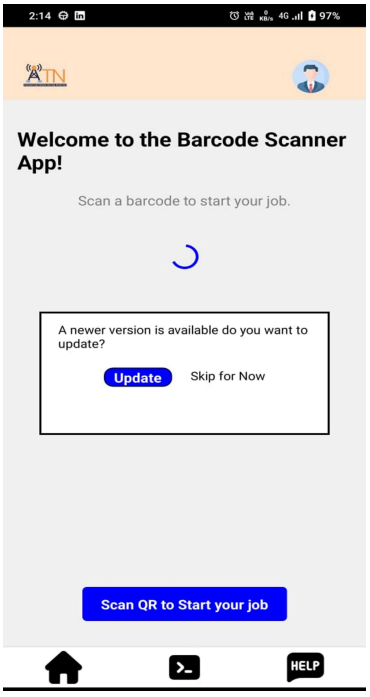


Fig 2.7: Entering Barcode scanner

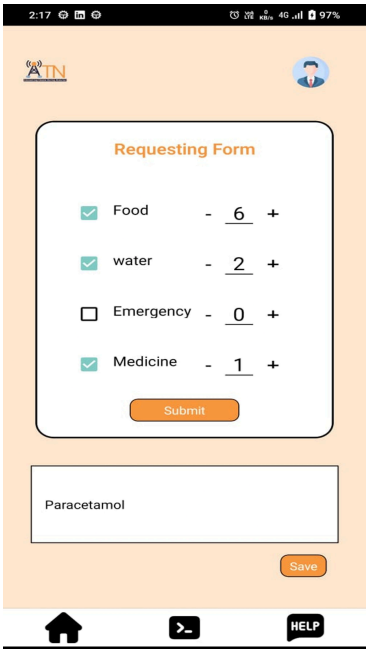


Fig 2.8: Requesting Needs

3.Nodes and Network:

3.1 ESP32

Fig 3.1.2 NodeMCU

3.1.1 ESP32 Specifications:

Model :

- ESP32 WROOM 32 by espressif

Microcontroller:

- Dual-core Tensilica LX6 microprocessor
- Operating Frequency: Up to 240 MHz
- Ultra-Low Power Co-processor

Wireless Connectivity:

- Integrated Wi-Fi 802.11 b/g/n (2.4 GHz)
- Bluetooth v4.2 BR/EDR and BLE (Bluetooth Low Energy)

Memory:

- 520 KB SRAM
- 448 KB ROM (for boot loader and core functions)
- External SPIRAM support up to 16 MB

Storage:

- Built-in 448 KB ROM
- External storage options:
    - SPI Flash up to 16 MB
    - MicroSD Card

I/O Pins:

- 38 GPIO pins (General Purpose Input/Output)
- Interfaces for UART, SPI, I2C, I2S, PWM, SDIO, and CAN

Analog Input:

- 18 channels of 12-bit SAR ADCs (Analog-to-Digital Converter)

Operating Voltage:

- 3.3V (Not 5V tolerant, level shifting required for interfacing with 5V devices)

Operating Temperature:

- 40°C to 125°C

Power Consumption:

- Optimised for low power consumption
- Multiple sleep modes for power saving:
    - Deep Sleep: Shut down most of the ESP32's internal circuits
    - Light Sleep: Maintain enough state to wake up quickly
    - Modem Sleep: Turn off the Wi-Fi modem while keeping the CPU running

Security:

- Hardware-based security features:
    - Secure Boot
    - Flash Encryption
- Cryptographic support for AES, SHA-2, RSA, ECC, etc.

Size:

- Compact form factor suitable for various embedded applications

Other Features:

- Real-Time Clock (RTC) with backup battery support
- Timers for PWM, watchdog and RTC
- Touch Sensor Inputs
- Temperature Sensor
- Hall Effect Sensor

3.1.2 ESP32 Pin Diagram:

Fig 3.1.2 pin diagram of NodeMCU

3.1.3 ESP32 Arduino IDE Setup:

Preferences:

- Go to “Files”
- Select “Preferences”
- Paste https://dl.espressif.com/dl/package_esp32_index.json this link in the “Additional Board Manager URLs”.
- Save the changes by clicking on the OK Button.

Installing the board:

- Go to “Tools”
- Select “Board manager”
- Search “ESP32” in the search bar
- Select the esp32 by espressif
- Click the install button on the right bottom.

Driver Installation:

- https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers click this text.
- Go to downloads and download the version for your specifications
- Extract the files once downloaded
- Click “SETUP” file
- Install the driver by accepting the policies and permissions

Partition Scheme (Only for Client and Base Stations):

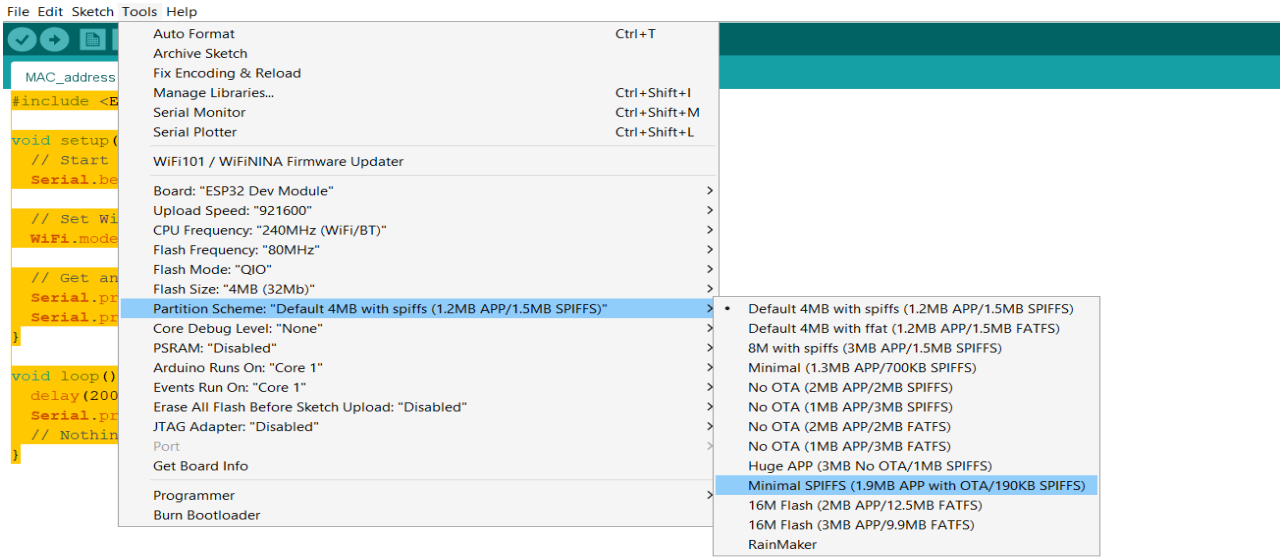


Fig 3.1.3 Arduino IDE Preferences Setting

- Go to “Tools”
- Select “Partition Schemes”
- Select “Minimal Spiffs (1.9MB APP with OTA / 190kb with SPIFFS)”

Libraries :

1. Go to tools
2. Select Manage libraries
3. Search for these libraries and install it

    a. Radio Head
    b. WiFi HTTPClient
    c. WebServer Adafruit
    d. Unified Sensor
    e. Adafruit_Sensor
    f. Adafruit_BMP280
    g. Adafruit_BME280
    h. SPI (if not installed)
    i. SoftwareSerial (if not installed)
    j. TinyGPS++ (if not installed)

3.2 ESP32 - LoRa :

3.2.0 LoRa (Long Range):

Fig 3.2.1 LoRa

3.2.1 LoRa Specifications:

Model :

- LoRa sx1278 (RF - 95) by SEMTECH (Antenna Mandatory)

Electrical Specifications:

- Operating Voltage: Typically 1.8V to 3.7V (varies depending on specific module)
- TX Power Output: Up to 20 dBm (adjustable)
- Receiver Sensitivity: Down to -148 dBm
- Low Current Consumption in sleep mode: Typically less than 1 µA

Frequency Range:

- 433 MHz, 470 MHz, 868 MHz, or 915 MHz bands (depending on the region and module variant)
- Programmable Frequency Synthesiser with resolution of 61 Hz

Modulation:

- LoRa modulation with spread spectrum technology
- FSK modulation available for compatibility with other systems

Data Rate:

- LoRa:
    - Up to 37.5 kbps (LoRa BW = 125 kHz, SF = 12)
    - Lower data rates possible with wider bandwidths and higher spreading factors
- FSK:
    - Programmable data rates up to 300 kbps

Interface:

- SPI (Serial Peripheral Interface) for communication with microcontrollers or other devices
- Other control signals for configuration and control

Operating Temperature Range:

- Typically from -40°C to 85°C, but specific module variants may have different ranges

Features:

- Built-in packet engine with CRC (Cyclic Redundancy Check) for data integrity
- Support for frequency hopping to improve robustness in noisy environments
- Flexible power management options to optimise power consumption for different applications
- Support for various modulation schemes and spreading factors to trade off between range and data rate

Package:

- Available in various packages, including small surface-mount modules for easy integration into designs

Reason for Choosing LoRa:

1. **Long Range Communication:** LoRa supports extensive coverage, transmitting data up to **10-15 km** in rural areas and **3-5 km** in urban settings.

2.**Low Power Consumption:** LoRa devices are **highly energy-efficient**, enabling years of operation on a single battery, ideal for remote deployments.

3. **High Capacity:** LoRa networks can handle millions of devices simultaneously, making them suitable for **large-scale IoT** deployments.

4. **Cost-Effectiveness:** Operating in unlicensed ISM bands and requiring less infrastructure, LoRa offers a **budget-friendly** solution for wide-area networks.

5. **Robustness and Security:** LoRa's spread spectrum technology provides interference resistance and supports **end-to-end encryption** for secure data transmission.

3.2.2 LoRa Pin Configurations:

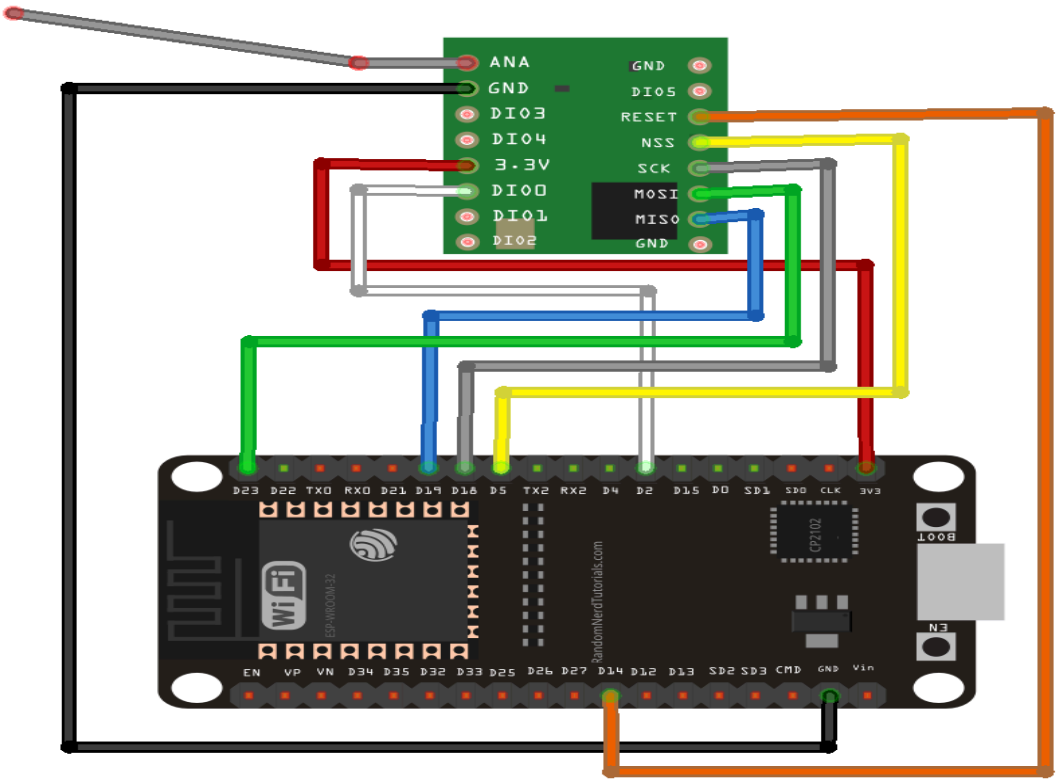


Fig 3.2.2 pin diagram of LoRa to ESP32

LoRa to ESP32:

- ANA : Antenna
- GND : GND
- DIO3 : don't connect
- DIO4 : don't connect
- 3.3V : 3.3V
- DIO0 : GPIO 2
- DIO1 : don't connect
- DIO2 : don't connect
- GND : don't connect
- DIO5 : don't connect
- RESET: GPIO 14
- NSS : GPIO 5
- SCK : GPIO 18
- MOSI : GPIO 23
- MISO : GPIO 19
- GND : don't connect

3.3 ESP32 - BMP280:

3.3.0 BMP280:

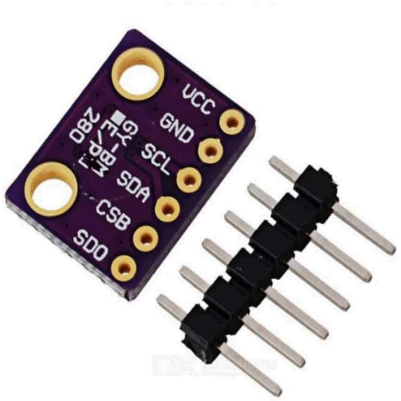


Fig 3.3.1 BMP280

3.3.1 BMP280 Specifications :

- The BMP280 is a barometric pressure sensor developed by Bosch Sensortec.

Environmental Specifications:

- Pressure Measurement Range: 300 hPa to 1100 hPa (altitude range -500 m to 9000 m)
- Temperature Measurement Range: -40°C to 85°C

Accuracy:

- Pressure: ±1 hPa (or ±1.7 metres)
- Temperature: ±1°C

Resolution:

- Pressure: 0.18 Pa (or 1.52 cm)
- Temperature: 0.01°C

Interfaces:

- Communication: I2C (up to 3.4 MHz) or SPI (up to 10 MHz)

Power Consumption:

- Current Consumption: 2.7 μA at 1 Hz (ultra-low power consumption)

Features:

- Built-in Temperature Sensor: Allows compensation for temperature variations during pressure measurements.
- Digital Filtering: Helps to reduce short-term pressure fluctuations caused by environmental conditions.
- Low Power Consumption: Designed for battery-powered applications and energy-efficient systems.
- Calibration Coefficients: Stored on-chip, eliminating the need for external circuitry or calibration.

Applications:

- Weather Forecasting
- Altitude Measurement (e.g., drones, altimeters)
- Indoor Navigation (e.g., floor detection in smartphones)
- Health and Sports Monitoring (e.g., tracking elevation changes during exercise)
- Industrial Applications (e.g., HVAC systems, environmental monitoring)

3.3.2 BMP280 Pin Configurations :

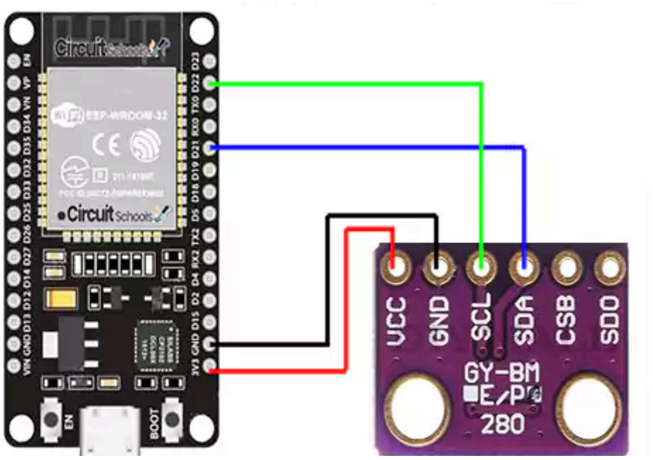


Fig 3.3.2 pin diagram of BMP280 to ESP32

BMP280 to ESP32 :

- VCC - 3.3v
- GND - GND
- SDA - D21
- SCK - D22
- CSB - No Connection
- SDD - No Connection

3.4 ESP32 - BME280

3.4.0 BME 280:

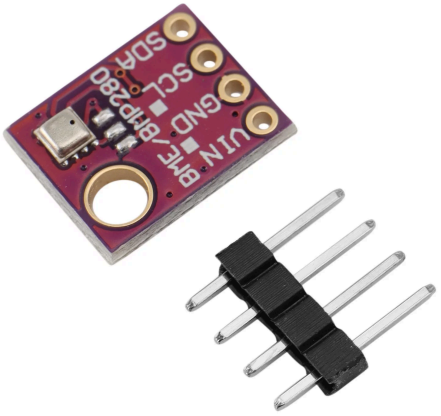


Fig 3.4.1 BME280

3.4.1 BMP280 Specifications:

Model:

- The BME280 is a combined sensor developed by Bosch Sensotec.

Environmental Specifications:

- Pressure Measurement Range: 300 hPa to 1100 hPa (altitude range -500 m to 9000 m)
- Temperature Measurement Range: -40°C to 85°C
- Humidity Measurement Range: 0% to 100% RH (Relative Humidity)

Accuracy:

- Pressure: ±1 hPa (or ±1.7 metres)
- Temperature: ±1°C
- Humidity: ±3% RH

Resolution:

- Pressure: 0.18 Pa (or 1.52 cm)
- Temperature: 0.01°C
- Humidity: 0.008% RH

Interfaces:

- Communication: I2C (up to 3.4 MHz) or SPI (up to 10 MHz)

Power Consumption:

- Current Consumption: Varies depending on the operating mode, typically ranges from 0.1 µA to 0.5 mA.

Features:

- Integrated Temperature Sensor: Allows compensation for temperature variations during pressure and humidity measurements.
- Digital Filtering: Helps reduce short-term pressure fluctuations caused by environmental conditions.
- Low Power Consumption: Designed for battery-powered applications and energy-efficient systems.
- Calibration Coefficients: Stored on-chip, eliminating the need for external circuitry or calibration.
- Optional IIR Filter: Available for additional filtering of humidity data.
- Humidity Sensing Accuracy: Enhanced by factory calibration, compensating for sensor-to-sensor variations.

Applications:

- Weather Monitoring: Provides accurate data for weather forecasting, including temperature, humidity, and pressure measurements.
- Indoor Environmental Monitoring: Tracks indoor climate conditions in smart home systems, HVAC (Heating, Ventilation, and Air Conditioning) systems, and other indoor applications.
- Health and Wellness: Monitors environmental conditions in wearable devices and health monitoring systems.
- Industrial Applications: Utilized for environmental monitoring in industrial automation, HVAC systems, and process control

3.4.2 BMP280 Pin Configuration:

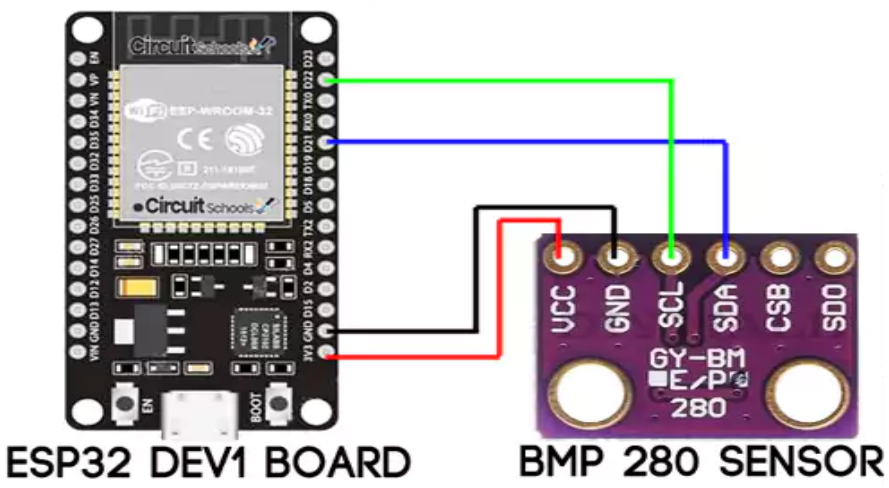


Fig 3.4.2 pin diagram of BME280 to ESP32

BMP280 to ESP32:

- VCC - 3.3v
- GND - GND
- SDA - D21
- SCK - D22
- CSB - No Connection
- SDD - No Connection

3.5 ESP32 - BH1750:

3.5.0 BH1750

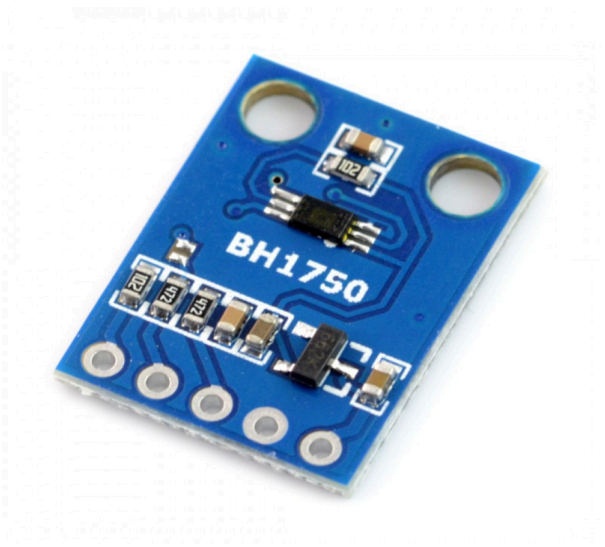


Fig 3.5.1 BH1750

3.5.1 BH1750 Specifications:

Model:

- The BH1750 is a light intensity sensor developed by Rohm.

Electrical Characteristics:

- *Supply Voltage (Vcc)*: 2.4V to 3.6V
- *Current Consumption*:
  - *Typical*: 0.12 mA
  - *Maximum*: 0.20 mA (at H-resolution mode)
- *Standby Current*: 0.01 µA
- *I2C Bus Voltage*: 1.8V to 3.6V
- *Output*: Digital, via I2C interface

Light Measurement:

- *Measurement Range*: 1 - 65535 lux
- *Resolution*:
  - *High Resolution Mode*: 1 lux

    - *Low Resolution Mode*: 4 lux
- *Accuracy*: ±20% (under defined conditions)

Operating Conditions:

- *Operating Temperature Range*: -40°C to +85°C
- *Storage Temperature Range*: -40°C to +85°C
- *Humidity Range*: 0% to 85% RH (non-condensing)

Features:

- *Modes*:
    - High resolution mode: 1 lux resolution, 120 ms measurement time
    - High resolution mode 2: 0.5 lux resolution, 120 ms measurement time
    - Low resolution mode: 4 lux resolution, 16 ms measurement time
- *Automatic data register reset (initiate new measurement cycle automatically)*
- *Low current by power down function*
- *Small measurement variation (± 20%)*
- *Digital output I2C bus interface (fast-mode and high-speed mode are available)*

Physical Characteristics:

- *Package Type*:
    - BH1750FVI: Surface-mount package (8-pin SOP)
    - BH1750FVC: Chip-scale package
- *Package Dimensions*:
    - BH1750FVI: 4.5mm x 2.0mm x 1.1mm
    - BH1750FVC: 1.6mm x 1.6mm x 0.75mm

Pin Configuration:

- *SDA*: Serial Data (I2C data)
- *SCL*: Serial Clock (I2C clock)
- *ADDR*: I2C address selection (low for address 0x23, high for address 0x5C)
- *GND*: Ground
- *VCC*: Power Supply
- *NC*: No Connect (unused pins)

Typical Applications:

- *Mobile phones*
- *Digital cameras*
- *LCD TVs*
- *Backlight control systems*

Reference Circuit:

- Pull-up resistors (typically 10 kΩ) are required for the SDA and SCL lines.
- Capacitor (0.1 µF) between VCC and GND for power stabilisation.

Usage Notes:

- The BH1750 can be directly connected to a microcontroller's I2C bus.
- It has two I2C addresses selectable by the ADDR pin, allowing for easy integration with other I2C devices.

These specifications make the BH1750 a versatile and efficient choice for ambient light sensing in a wide range of consumer electronics and industrial applications.

3.5.2 BH1750 Pin Configurations:

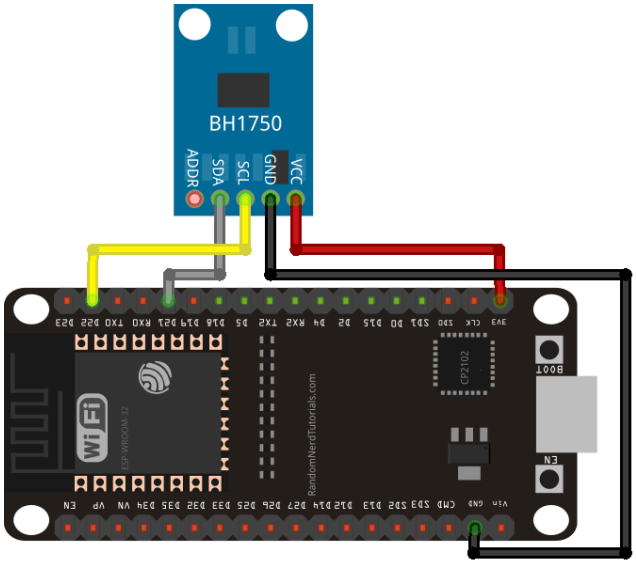


Fig 3.5.2 pin diagram of BH1750 to ESP32

BH1750 to ESP32:

- VCC - 3.3v
- GND - GND
- SDA - D21
- SCl - D22
- ADDR - No Connection

3.6 ESP32 - L89 :

3.6.0 L89:

Fig 3.6.1 L89

3.6.1 L89 Specifications:

Model:

- L89 GNSS module by Quectel

GNSS Support:

- Constellations: GPS, IRNSS, GLONASS, BeiDou, Galileo, and QZSS.
- Frequency Bands: L1 (1575.42 MHz), L5 (1176.45 MHz), B1 (1561.098 MHz), and E1.

Antennas and Sensitivity:

- Dual embedded antennas (patch and chip antenna).
- Integrated LNAs (Low Noise Amplifiers) for improved sensitivity.
- Tracking Sensitivity: -147 dBm.
- Acquisition Sensitivity: -162 dBm.

Performance:

- TTFF (Time to First Fix): Reduced due to multi-band operation.
- Enhanced positioning accuracy, especially in urban environments.

Interface and Power:

- Interfaces: I2C, UART.
- Supply Voltage: 3.1 to 4.3 V.
- Acquisition Current: 90 mA.

Physical and Environmental:

- Dimensions: 25.0 mm × 16.0 mm × 6.8 mm.
- Weight: 8.2 g.
- Operating Temperature: -40 to 85°C.
- Compliance: RoHS compliant.

Additional Features:

- Support for DGPS and SBAS (WAAS, EGNOS, MSAS, GAGAN).
- Great anti-jamming performance due to multi-frequency operation.

3.6.2 L89 Pin Configurations:

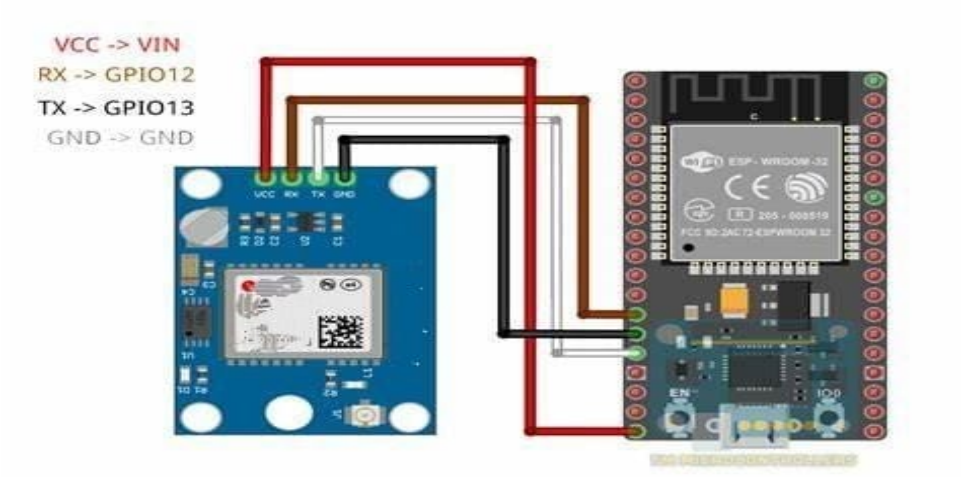


Fig 3.6.2 pin diagram of L89 to ESP32

L89 to ESP32

- VCC - 3.3v
- GND - GND
- RX - D12 / D16
- TX - D13 / D17

3.7 Mesh Network - Rhmesh

https://github.com/PaulStoffregen/RadioHead/blob/master/RHMesh.h

The RHMesh library is part of the RadioHead suite developed by Mike McCauley. It extends the RHRouter class to support mesh networking, enabling multi-hop routing and automatic route discovery in dynamic network topologies. This makes RHMesh particularly suitable for applications where nodes can move or change status, ensuring reliable communication even in fluid environments.

3.7.1 RHMesh Features

1. Mesh Networking: Supports multi-hop communication across a network of nodes.

2. Automatic Route Discovery: Dynamically discovers routes to destination nodes when needed.

3. Route Failure Handling: Detects and responds to route failures, ensuring messages can be rerouted as necessary.

4. Reliable Hop-to-Hop Delivery: Uses acknowledgments at each hop to ensure reliable delivery.

5. Optimised Memory Usage: Designed to operate within the limited memory constraints of typical Arduino and similar microcontroller environments.

6. Message Optimisation: Intermediate nodes cannot decode messages; only destination nodes can unwrap them, ensuring end-to-end encryption and security.

7. Auto Node-Addition: New node is automatically added to the RouteTable once it's initiated.

8. BroadCast message: Each node can send a single message to all other node in the route table by using NODE_ID : 255 (or)RH_BROADCAST_ADDRESS.

3.7.2 RHMesh Protocol Overview

Route Status Code:

- 0 ⇒ RH_ROUTER_ERROR_NONE
- 1 ⇒RH_ROUTER_ERROR_NO_ROUTE
- 2 ⇒RH_ROUTER_ERROR_TIMEOUT
- Not Recognised ⇒ General failure

3.7.3 RHMesh Detailed Functionality:

Initialization and Basic Operation

When an RHMesh node starts, it doesn't have knowledge of any routes. It relies on automatic route discovery to establish paths to other nodes.

Constructor:

RHMesh(RHGenericDriver& driver, uint8_t thisAddress = 0);

- Initialises the mesh node with a given driver and node address.

Route Discovery

1. Sending a Message: When a node sends a message using `sendtoWait`, it first checks if a route to the destination exists in its routing table.

2. Route Request: If no route is known, it broadcasts a `MeshRouteDiscoveryMessage` with type `RH_MESH_MESSAGE_TYPE_ROUTE_DISCOVERY_REQUEST`.
3. Intermediate Nodes: Nodes receiving the request check if the destination is themselves. If not, they rebroadcast the request, adding themselves to the list of visited nodes.
4. Destination Node: When the destination node receives the request, it sends a unicast `MeshRouteDiscoveryMessage` with type `RH_MESH_MESSAGE_TYPE_ROUTE_DISCOVERY_RESPONSE` back to the origin.
5. Route Formation: Intermediate nodes use the response to update their routing tables with the route back to the origin and other nodes on the path.

Route Failure Handling

When a node cannot deliver a message to the next hop:

1. Route Failure Message:
    - It sends a `MeshRouteFailureMessage` to the originator indicating the failure.
2. Route Deletion:
    - Intermediate nodes and the originator delete the failed route from their routing tables.
3. Reattempt:
    - If a message needs to be sent again, a new route discovery process is initiated.

3.7.4 RHMesh Coding Description:

Public Methods - Sending message

- sendtoWait:
    - Sends a message to the destination, initiating route discovery if necessary.

      uint8_t sendtoWait(uint8_t* buf, uint8_t len, uint8_t dest, uint8_t flags = 0);

- Parameters:
  - `buf`: Pointer to the message data.
  - `len`: Length of the message.
  - `dest`: Destination node address.
  - `flags`: Optional flags for use by subclasses or application layer.

- recvfromAck:

  - Receives a message addressed to this node, sends an acknowledgment, and processes the message.

    bool recvfromAck(uint8_t* buf, uint8_t* len, uint8_t* source = NULL, uint8_t* dest = NULL, uint8_t* id = NULL, uint8_t* flags = NULL, uint8_t* hops = NULL);

  - Parameters:
    - `buf`: Location to copy the received message.
    - `len`: Available space in `buf`, set to the actual number of bytes copied.
    - `source`, `dest`, `id`, `flags`, `hops`: Optional pointers to retrieve additional message details.

- recvfromAckTimeout:

  - Similar to `recvfromAck`, but with a timeout parameter.

    bool recvfromAckTimeout(uint8_t* buf, uint8_t* len, uint16_t timeout, uint8_t* source = NULL, uint8_t* dest = NULL, uint8_t* id = NULL, uint8_t* flags = NULL, uint8_t* hops = NULL);

  - Parameters:
    - `timeout`: Maximum time to wait in milliseconds.

The RHMesh library provides a robust framework for implementing mesh networks with dynamic routing and reliable communication. Its design accommodates the constraints of microcontroller environments while offering essential features for complex networking scenarios. By leveraging automatic route discovery and handling route failures, RHMesh ensures resilient and flexible wireless communication.

3.7.4 RHMesh - Esp32 Code:

Sender -
https://github.com/DharunAP/LoRa-Networking/blob/SenderReceiver/Navic_Client_Station.ino

Receiver and Sender-
https://github.com/DharunAP/LoRa-Networking/blob/SenderReceiver/BMP280_Base_Station.ino

Receiver -
https://github.com/DharunAP/LoRa-Networking/blob/SenderReceiver/Central_Station.ino

Source : https://github.com/PaulStoffregen/RadioHead/blob/master/RHMesh.h

---

4 . Web Application

This web application utilises React for the frontend and Node.js for the backend, ensuring a fast and responsive user experience. MongoDB Atlas is used for cloud storage, allowing data to be accessed from anywhere. The application integrates Grafana and QGIS for comprehensive location tracking and detailed dashboard analytics. Users can view completed and upcoming requests, with advanced filtering options based on districts to streamline data management and accessibility. The combination of these technologies ensures a scalable, efficient, and user-friendly platform suitable for diverse application needs.

Dependencies

1) Node
2) Web Development Environment
3) MongoDb Atlas

Step-by-Step Guide

4.0 Install the Node

- Install the latest version of the Node in your system
- Also if installed verify the version by ” node -v “

4.1 Create the React application

- Create the new React application with the Vite using “ npm create vite@latest “Your Project Name”
- After this change the directory to the file
- Change the directory to the file and install the dependencies using " npm install
- Hence the file will be created with the project name that you has mentioned

4.2 Backend Setup

- Create separate folder for the backend part of the applications
- Install the ExpressJs using " npm install express"
- Install the Body Parser using " npm install body-parser"
- Install the Cors using " npm install cors"
- Install the Mongoose using " npm install mongoose"

4.3 : MongDB Atlas Setup :

4.3.0 Sign Up:

- Go to the MongoDB Atlas website.
- Click on "Start Free" and sign up using your email or use an existing Google account.

4.3.1 Create a Cluster:

- After logging in, click on "Build a Cluster".
- Choose a cloud provider and region.
- Select the cluster tier (you can start with the free tier).
- Click "Create Cluster".

4.3.2 Configure Cluster

- Create Database User:
- Go to the "Database Access" tab.
- Click "Add New Database User".
- Choose "Password" as the authentication method.
- Enter a username and password, and save these credentials as you'll need them later.
- Set user privileges (read and write to any database is a common choice for development).

4.3.3 Network Access:

- Go to the "Network Access" tab.
- Click "Add IP Address".
- You can allow access from anywhere by clicking "Allow Access from Anywhere" or specify an IP address.
- Click "Confirm".

4.3.4 Get the Connection String

- Connect to Your Cluster:
- Go to the "Clusters" tab shown in Fig 4.3.1
- Click "Connect" for your cluster.
- Select "Connect Your Application".
- Copy the connection string provided (e.g., mongodb+srv://<username>:<password>@cluster0.mongodb.net/test?retryWrites=true&w=majority).

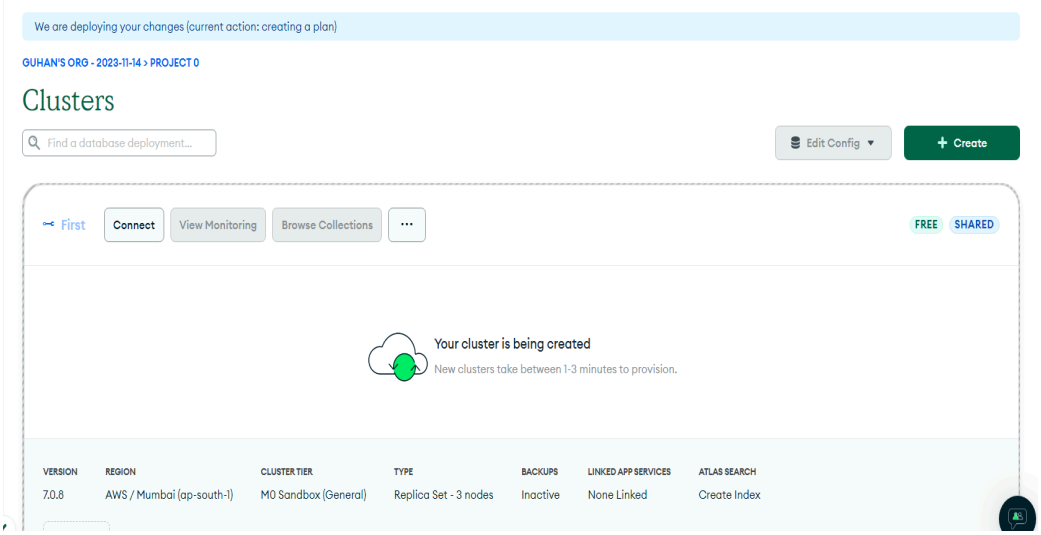


Fig : 4.3.1 - Cluster tab to get the connection string

4.4 Hosting the Backend

- Host the backend application to use it through whole internet
- Here the can use the Render to host the Backend server

4.4.0 Create a Render Account:

- Sign up for an account on Render if you don't already have one.

4.4.1 Create a New Web Service:

- Log in to your Render account.
- Click on the "New" button and select "Web Service" from the dropdown menu shown in Fig 4.4.2.

4.4.2 Connect Your Repository

- Connect Render to your GitHub or GitLab account.
- Select the repository containing your Node.js application.

4.4.3 Configure the Service:

- Name: Choose a name for your service.
- Branch: Select the branch you want to deploy (e.g., main or master).
- Root Directory: Specify the directory if your server.js is not in the root of the repository.
- Environment: Select the environment (Node.js).
- Build Command: If your project uses a build tool (like Webpack), specify the build command (e.g., npm run build).
- Start Command: Specify the command to start your application (e.g., node server.js or npm start).

4.4.4 Set Environment Variables:

- In the "Advanced" section, add any environment variables your application needs (e.g., MONGODB_URI for your MongoDB connection string).

4.4.5 Deploy the Application:

- Click "Create Web Service" to start the deployment process.
- Render will build and deploy your application. You can view the build logs to monitor the progress.

4.4.6 Access Your Application:

- Once the deployment is complete, Render will provide a URL where your Node.js application is hosted.
- You can visit this URL to access your live backend application.

4.4.7 Monitor and Manage:

- Use the Render dashboard to monitor the status of your service, view logs, and manage settings.
- You can also set up automatic deployments to trigger whenever you push changes to the connected branch in your repository.

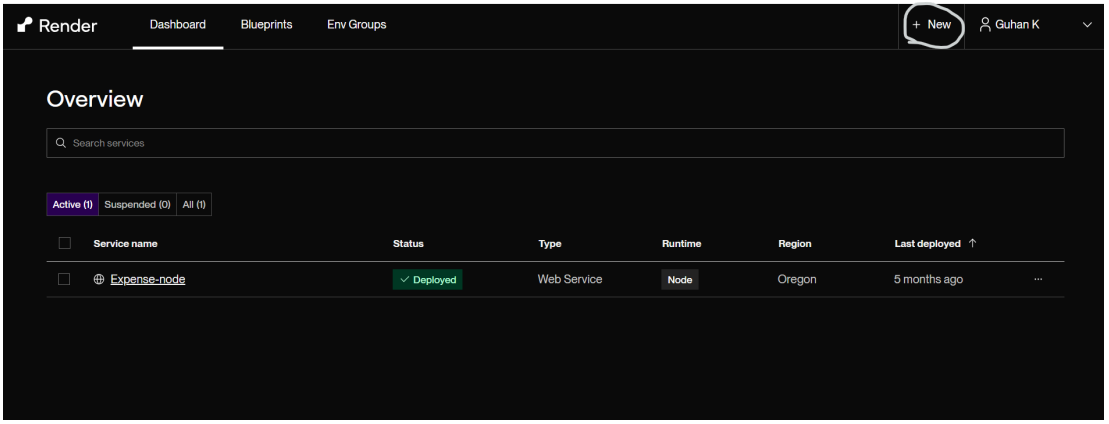


Fig : 4.4.2 - Render Dashboard for new deploying

4.5 Hosting the Frontend

- To use the React application over the internet we need to host
- Here the can use the Vercel to host the application

4.5.0 Create a New Project:

- Log in to your Vercel account.

4.5.1 Click on the "New Project" button on the dashboard.

- Log in to your Vercel account.
- Click on the "Add New" button on the dashboard shown on Fig 4.5.3

4.5.2 Import Your Git Repository:

- Connect Vercel to your GitHub, GitLab, or Bitbucket account.
- Select the repository containing your React application.

4.5.3 Configure Project Settings:

- Project Name: Choose a name for your project.
- Root Directory: Specify the directory if your package.json is not in the root of the repository.
- Framework Preset: Vercel will automatically detect "Create React App" if applicable. Otherwise, select "Create React App" or the appropriate framework from the dropdown.
- Build and Output Settings: The default settings should work for most Create React App setups:
- Build Command: npm run build
- Output Directory: build

4.5.4 Set Environment Variables:

- If your application requires environment variables, go to the "Environment Variables" section.
- Add the necessary environment variables (e.g., REACT_APP_API_URL).

4.5.5 Deploy the Application:

- Click the "Deploy" button to start the deployment process.
- Vercel will build and deploy your React application. You can monitor the deployment progress in the Vercel dashboard.

4.5.6 Access Your Application:

- Once the deployment is complete, Vercel will provide a URL where your React application is hosted.
- You can visit this URL to access your live frontend application.

4.5.7 Manage and Monitor:

- Use the Vercel dashboard to monitor the status of your deployment, view logs, and manage settings.
- Also the user can set up automatic deployments to trigger whenever you push changes to the connected branch in your repository.

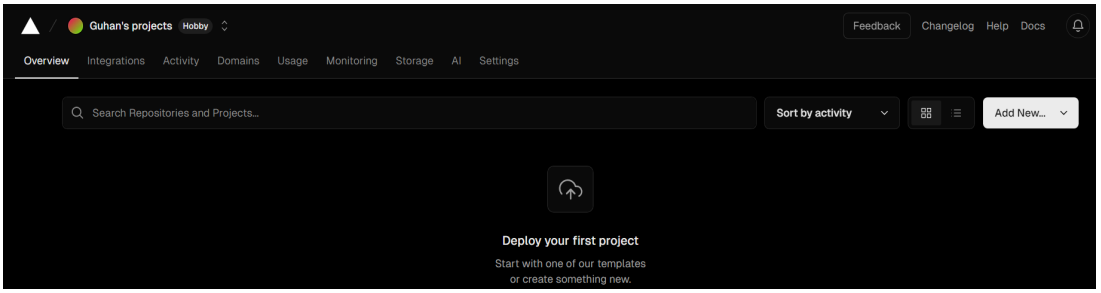


Fig : 4.5.3 - Dashboard for deploying the new project

4.6 Frontend and Backend Integration

4.6.0 Backend

- Use the cors to avoid the cors problem while working in the internet

Project Github Link : https://github.com/Guhansamy/ATN-backend.git

4.6.1 Frontend

- Use the fetch method of React to with the hosted link to access the database

Project Github Link : https://github.com/Guhansamy/ATN-frontend.git

Fig : 4.6.4 - Both Frontend and Backend Integrated

5. QGIS

QGIS (Quantum Geographic Information System) is a free, open-source software that allows users to create, edit, visualise, analyse, and publish geospatial information. In this guide, we will create an interactive web map that displays the locations of victims, LoRa devices, and sensor devices. Active LoRa devices will be marked in green, while failed ones will be marked in red. We will add layers using an XYZ layer, export the map using the QGIS2Web plugin, and then modify the exported code to include labels and buttons. When a user clicks on a node, a label will show a button that sends data to Grafana. Our main goal is to present this information visually for easy user understanding.

Step-by-Step Guide

Module 5: QGIS and QGIS2Web Plugin

5.0 Install QGIS and QGIS2Web Plugin

5.0.1 Download and Install QGIS:

- Go to the QGIS official website.
- Download the latest version of QGIS suitable for your operating system (Windows, macOS, Linux).
- Follow the installation instructions provided on the website for your OS.

5.0.2 Install QGIS2Web Plugin:

- Open QGIS after installation.
- Navigate to Plugins > Manage and Install Plugins.
- In the Plugins dialog, search for "qgis2web".
- Select the QGIS2Web plugin and click Install Plugin.

5.1 Prepare Your Map in QGIS

5.1.0 Add Layers:

- Open QGIS.
- Load the layers you want to include in your web map.
- Go to Layer > Add Layer.
- Choose the appropriate type of layer (vector, raster, or XYZ layers).
- Load your shapefiles or other geographic data formats.

5.1.0.1 Applications of Layers

Layers are used to enhance the functionality, usability, and analytical capabilities of maps. Here are some key advantages of using layers in mapping:

1. Data Organization and Management:

- **Separation of Data:** Different types of data can be stored in separate layers. For instance, roads, rivers, buildings, and vegetation can each be in their own layer. This makes it easier to manage and update data.
- **Hierarchical Structure:** Layers allow for a hierarchical structuring of data, making it easier to organise complex datasets.

2. Enhanced Visualisation:

- **Thematic Mapping:** Different themes or categories of data can be visualised distinctly. For example, land use types (residential, commercial, agricultural) can be shown in different colours.

3. Interactivity:

- **Dynamic Interaction:** Layers can include interactive elements like pop-ups, tooltips, and clickable areas that provide more information or perform actions.
- **Layer Control:** Users can turn layers on and off, allowing them to focus on specific aspects of the data without distraction from other data.

4. Customization and Flexibility:

- **Symbology:** Layers allow for customised symbols, colours, and styles for different types of data, enhancing the readability and aesthetics of the map.
- **Scalability:** Layers make it easier to scale projects by adding new data without affecting the existing structure.
- **Optimised Rendering:** By loading only the necessary layers, the performance of the mapping application can be optimised, especially when dealing with large datasets.

5. Data Integration:

- **Combining Data Sources:** Layers enable the integration of various data sources, such as satellite imagery, vector data, and real-time sensor data, into a cohesive map.
- **Multi-disciplinary Use:** Different disciplines (e.g., urban planning, environmental science, disaster management) can overlay their specific data onto a common map framework.

6. Analysis and Decision Making:

- **Informed Decisions:** Layers provide a clear and organised way to view multiple datasets, aiding in better analysis and more informed decision-making.
- **Scenario Analysis:** Different scenarios can be analysed by toggling various layers, such as the impact of new infrastructure on flood zones.

5.1.1 Open the Data Source Manager:

- In QGIS, go to Layer in the top menu and select Data Source Manager.

5.1.2 Select XYZ Tiles:

- In the Data Source Manager window, choose the XYZ option from the left sidebar.

5.1.3 Manage XYZ Connections:

- In the XYZ Connections section, you can manage your tile connections. You will see options to New, Edit, and Remove connections.

5.1.4 Add a New XYZ Connection:

- Click on the New button to add a new XYZ connection.
- A new window will open where you can enter the connection details.

5.1.5 For Example:

- Name: Give your new XYZ layer a name. In this case, it might be something descriptive like "ESRI World Imagery".
- URL: Enter the URL template for the tile service. For example, the URL is https://services.arcgisonline.com/arcgis/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}. This URL will fetch the tiles from the ESRI World Imagery tile service.
- Min. Zoom Level: Set the minimum zoom level. The default is usually 0.
- Max. Zoom Level: Set the maximum zoom level. In your screenshot, it's set to 18, which is common for many tile services.
- Referrer: If needed, you can specify a referrer URL. This is usually optional.
- Tile Resolution: Set the tile resolution if known, otherwise leave it as "Unknown (not scaled)".
- Interpretation: Leave this as "Default" unless you need a specific interpretation for the tiles.

5.1.6 Save and Add the Layer:

- After entering all the necessary details, click the Add button.
- The XYZ tile layer will be added to your QGIS project and displayed in the Layers panel.

5.2 Configure QGIS2Web Plugin

5.2.0 Open QGIS2Web Plugin:

- Go to Web > qgis2web > Create web map.

5.2.1 Configure Export Options:

- **Layers and Appearance:**
  - Select the layers you want to include in the web map.
- **Export Format:**
  - Choose between Leaflet or OpenLayers (Leaflet is commonly used and user-friendly).
  - Leaflet allows you to customise the map and add functionalities in the map by using JavaScript.
- **Map Options:**
  - Extent: Set the extent of the map to cover your area of interest.
  - Controls: Configure controls like zoom, scale, search, and live location.
- **Preview:**
  - Use the preview pane to see how your web map will look.

5.3 Export the Web Map

5.3.0 Export Map:

- Click Export.
- Choose the directory where you want to save the web map files.
- The plugin will generate HTML, JavaScript, and CSS files required for your web map.

5.4 Access and Customise the Web Map

5.4.0 Local Preview:

- Open the exported HTML file in a web browser to preview your web map locally.

5.4.1 Edit HTML/JavaScript/CSS:

- For advanced customizations, you can manually edit the exported HTML, JavaScript, and CSS files.
- Add custom scripts or styles to enhance functionality and appearance.

Customizations

5.4.2 Custom Icon Function:

- Purpose: This function creates custom icons for the map markers. It specifies the image URL for the icon, its size, the point of the icon that should be anchored to the marker's location, and the point from which a popup should open.

5.4.3 Defining Colors and Places:

- **Colours:** A JavaScript object called colours is used to store the status of various LoRa nodes. Each key in the object corresponds to a node ID, and the value indicates whether the node is active (1) or failed (0).
- **Places:** An array called places is used to define the coordinates, names, and custom icon URLs for different locations. Each entry in the array represents a location with its ID, name, latitude, longitude, and icon URL.

5.4.4 Sending Data to Backend:

- **Function:** A function sendDataToBackend is defined to send data to the backend server when a marker is clicked. This function uses the Fetch API to send a POST request with the marker's ID, name, latitude, and longitude as a JSON object.
- **Usage:** When a user clicks on a marker, this function is triggered to send the marker's details to a specified backend URL.

5.4.5 Periodic Data Update:

- **Set Interval:** A setInterval function is used to periodically fetch updated data from the backend server. This is done every 30 seconds.
- **Fetching Data:** The function fetches data from the backend URL and updates the colours object based on the fetched data.
- **Updating Markers:** After updating the colours object, the function iterates over the places array and updates the markers on the map. It sets the

marker's icon and name based on the current status (active or failed) of each node.

5.4.6 Adding Markers to the Map:

- **Function to Add Markers:** A function addMarker is defined to add markers to the map. It takes the marker's ID, latitude, longitude, name, and icon URL as arguments.
- **Creating Popups:** For each marker, a popup is created with the location's name and a button. When the button is clicked, it triggers the sendDataToBackend function to send the marker's details to the backend.
- **Custom Icons:** The function uses the createCustomIcon function to create a custom icon for each marker based on the provided icon URL.

5.4.7 Module Export:

- **Custom Leaflet Control:** A custom Leaflet control is extended to include search functionality. This control allows users to search for locations on the map and provides various customization options for the search behavior.
- **URL Parsing:** A method findsearch is included to extract a location from the URL and trigger a geocode search. This helps in locating and highlighting specific places on the map based on the URL parameters.

5.4.8 Add Labels and Buttons:

- Modify the exported JavaScript code to add labels for victim locations and LoRa devices.
- When a user clicks on a node, a label should show a button.
- Configure the button to send data to Grafana when clicked.

5.5 Link:

- **Project Github Link** : https://github.com/dprakash22/LeafletmapQGIS
- **Map Hosted Link :** https://dprakash22.github.io/LeafletmapQGIS/?search=13.0843,80.2705#11/13.1339/80.2737

5.5.1 Map view:

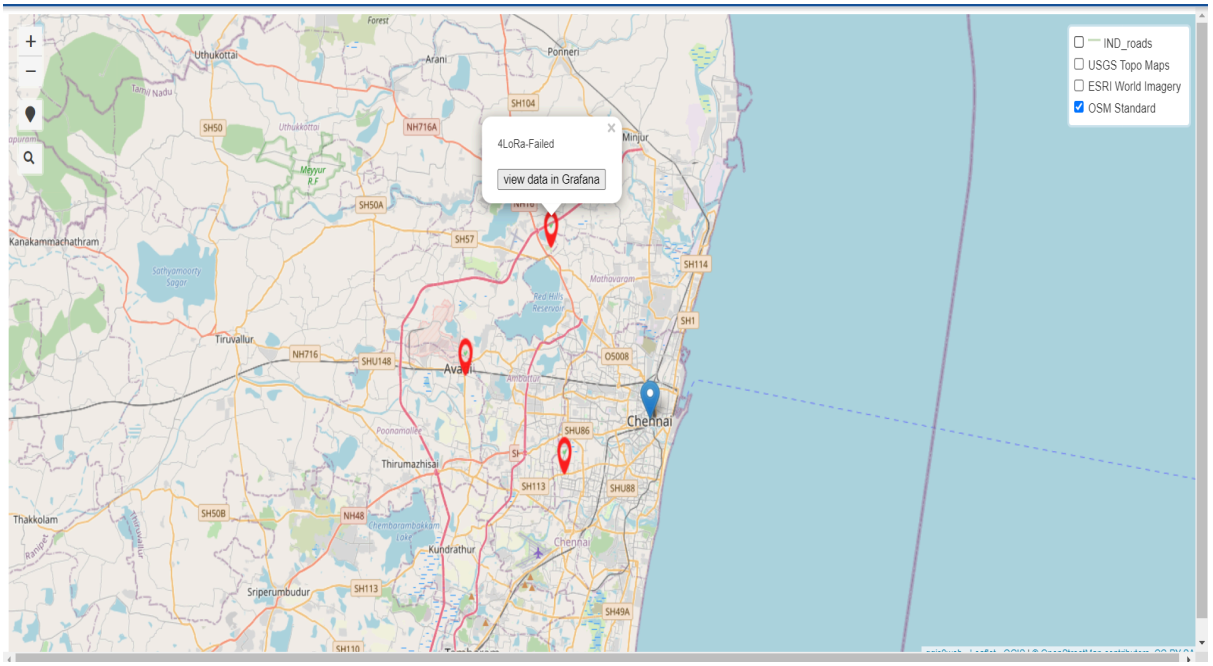


Figure 5.0

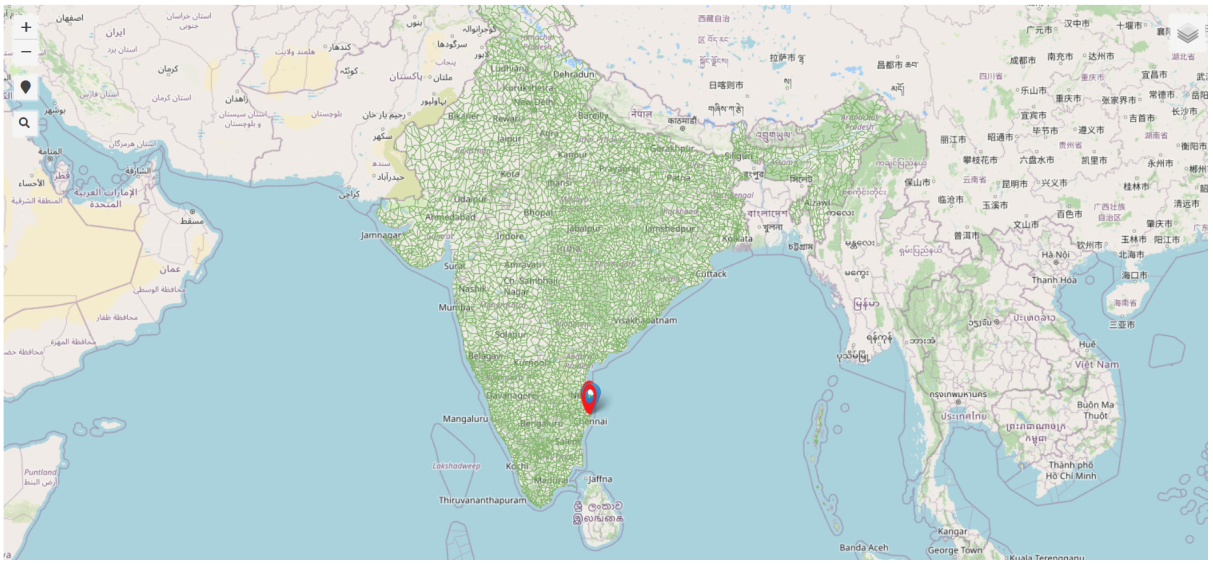


Figure 5.1

6. Grafana

Resources:

Sensor Data.csv - View

Influx DB Installation guide - View

Grafana Installation guide - View

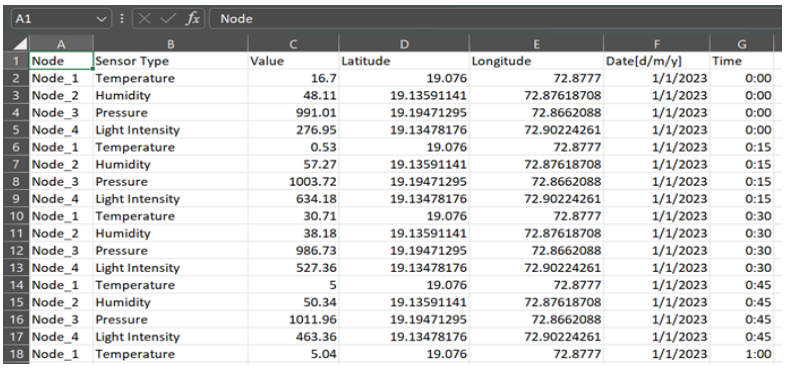
This documentation details the process of implementing a Grafana dashboard using timestamp information stored in a CSV file. The data is ingested into an InfluxDB database, which is then connected to Grafana for visualisation. Filters are also added to the Grafana dashboard to enhance data interaction. This guide provides a step-by-step approach to help new developers replicate the setup and understand the integration process.

6.01 Step-by-Step Guide

Step 1: **Prepare the CSV File**

1. **Create the CSV File:**

- Ensure the CSV file contains the necessary timestamp information and any other relevant data.



	A	B	C	D	E	F	G
1	Node	Sensor Type	Value	Latitude	Longitude	Date[d/m/y]	Time
2	Node_1	Temperature	16.7	19.076	72.8777	1/1/2023	0:00
3	Node_2	Humidity	48.11	19.13591141	72.87618708	1/1/2023	0:00
4	Node_3	Pressure	991.01	19.19471295	72.8662088	1/1/2023	0:00
5	Node_4	Light Intensity	276.95	19.13478176	72.90224261	1/1/2023	0:00
6	Node_1	Temperature	0.53	19.076	72.8777	1/1/2023	0:15
7	Node_2	Humidity	57.27	19.13591141	72.87618708	1/1/2023	0:15
8	Node_3	Pressure	1003.72	19.19471295	72.8662088	1/1/2023	0:15
9	Node_4	Light Intensity	634.18	19.13478176	72.90224261	1/1/2023	0:15
10	Node_1	Temperature	30.71	19.076	72.8777	1/1/2023	0:30
11	Node_2	Humidity	38.18	19.13591141	72.87618708	1/1/2023	0:30
12	Node_3	Pressure	986.73	19.19471295	72.8662088	1/1/2023	0:30
13	Node_4	Light Intensity	527.36	19.13478176	72.90224261	1/1/2023	0:30
14	Node_1	Temperature	5	19.076	72.8777	1/1/2023	0:45
15	Node_2	Humidity	50.34	19.13591141	72.87618708	1/1/2023	0:45
16	Node_3	Pressure	1011.96	19.19471295	72.8662088	1/1/2023	0:45
17	Node_4	Light Intensity	463.36	19.13478176	72.90224261	1/1/2023	0:45
18	Node_1	Temperature	5.04	19.076	72.8777	1/1/2023	1:00

Fig 1.0 Sensor Data.csv file

2. **Verify the Data:**

   - Ensure the CSV file is correctly formatted and contains no errors.

   - Use a text editor or a spreadsheet application to review the data.

Step 2: Set Up InfluxDB

1. **Install InfluxDB:**

   - Download and install InfluxDB from the [official website].

   - Follow the installation instructions for your operating system.

2. **Start InfluxDB:**

   - Start the InfluxDB service using the command:

     influxd

3. **Create a Database:**

   - Access the InfluxDB CLI by running:

     Influx

   - Create a new database:

     CREATE DATABASE mydatabase

4. Import the CSV Data:

   - Use the `influx` command to import data from the CSV file:

     influx -import -path=/path/to/yourfile.csv -precision=s

Step 3: Set Up Grafana

1. **Install Grafana:**

   - Download and install Grafana from the [official website].

   - Follow the installation instructions for your operating system.

2. **Start Grafana:**

   - Start the Grafana service using the command:

     sudo systemctl start grafana-server

3. **Access Grafana:**

   - Open a web browser and navigate to `http://localhost:3000`.

   - Log in with the default credentials (username: `admin`, password: `admin`).

4. **Add InfluxDB as a Data Source:**

   - Go to the Grafana homepage and click on "Add your first data source".

   - Select "InfluxDB" from the list of available data sources.

   - Fill in the required connection details:

     - URL: `http://localhost:8086`

     - Database: `mydatabase`

     - User: (if applicable)

     - Password: (if applicable)

   - Click "Save & Test" to verify the connection.

Step 4: Create a Grafana Dashboard

1. **Create a New Dashboard:**

   - Click on the "+" icon on the left sidebar and select "Dashboard".

   - Click "Add new panel" to create a new panel for visualising your data.

2. **Configure the Panel:**

   - Select "InfluxDB" as the data source.

   - Write a query to fetch the data from InfluxDB:

     SELECT "value" FROM "measurement_name" WHERE $timeFilter

   - Configure the visualisation settings (e.g., graph type, axes, legend).

3. **Add Filters:**

   - Use Grafana's built-in filter options to add time range filters or custom variable filters.

   - Click on "Dashboard settings" and navigate to the "Variables" section.

   - Add new variables and configure their settings to create interactive filters.

Step 5: Customise the Dashboard

1. **Add Additional Panels:**

   - Repeat the process of adding new panels to include different visualisations or metrics.

   - Customise each panel to match your specific needs.

2. **Organise the Layout:**

   - Arrange the panels on the dashboard to create a coherent layout.

   - Use drag-and-drop functionality to reposition panels.

3. **Save the Dashboard:**

- Click on the disk icon to save your dashboard.

- Provide a name and description for the dashboard.

Fig. 1.1 - ATN Grafana Dashboard

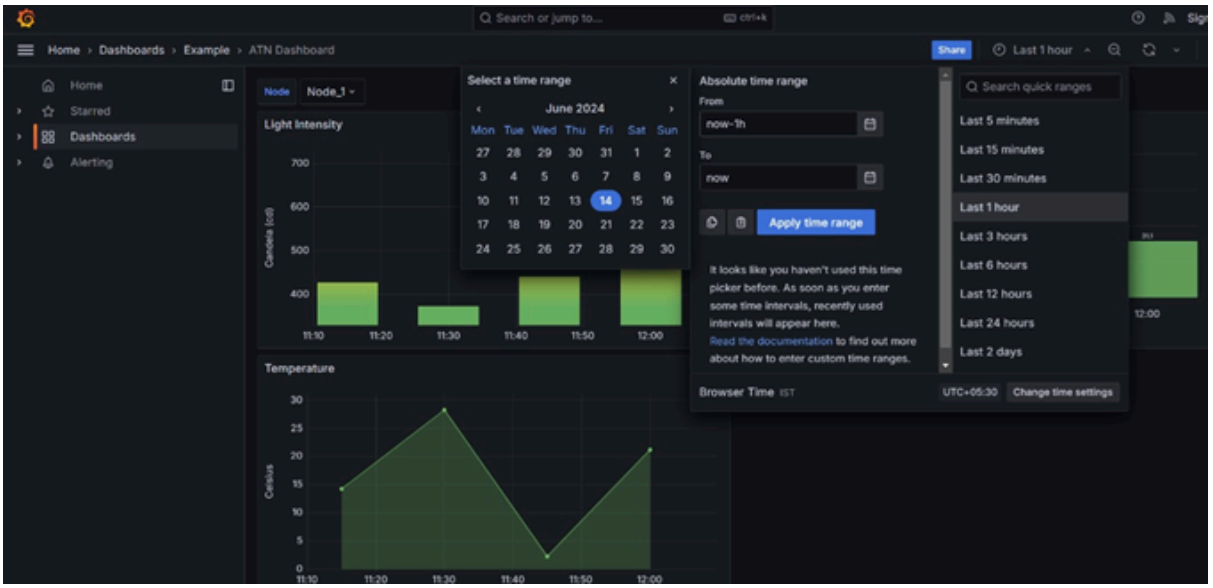


Fig 1.2 - Time Filter in ATN Grafana Dashboard

This documentation provides a comprehensive guide to implementing a Grafana dashboard using data from a CSV file stored in InfluxDB. By following these steps, a new developer should be able to replicate the setup and customise the dashboard to fit their specific needs. Ensure to maintain the system and keep the documentation updated for future reference.

7.REFERENCES:

- **Grafana:** https://grafana.com/docs/grafana/latest/
- **InfuxDB:** https://docs.influxdata.com/
- **RHMesh:** https://github.com/PaulStoffregen/RadioHead/blob/master/RHMesh.h
- **React :** https://react.dev/reference/react
- **Express** : https://expressjs.com/en/guide/routing.html
- **MongoDB Atlas** : https://www.mongodb.com/docs/atlas/create-connect-deployments/
- **Integration** : https://www.freecodecamp.org/news/create-a-react-frontend-a-node-express-backend-and-connect-them-together-c5798926047c/
- **QGIS:** https://docs.qgis.org/3.34/en/docs/user_manual/index.html

GitHub Links:

- **QGIS:** https://github.com/dprakash22/map.git
- **Mobile App:** https://github.com/DharunAP/ATN-mobile-APP
- **Network / Nodes:** https://github.com/DharunAP/LoRa-Networking
- **Website :** https://github.com/dprakash22/ATN-Web.git
- **React native setup** - https://reactnative.dev/docs/set-up-your-environment
- **QR code scanner** - https://medium.com/@varunkukade999/qr-code-scanner-in-react-native-527577aa74b1

- **React native wifi** - https://www.npmjs.com/package/react-native-wifi-reborn