Project Title:

Smart Vaccine Box Tracker and Alert System using NodeMCU (ESP8266)

Course:

Embedded Systems and IoT

Team Members:

1. Md. Sabbir Hossain (221-15-5199)
2. Md. Abu Nayem (221-15-5198)
3. Nadia Islam Nupur (221-15-5021)
4. Md. Didarul Islam Didar (221-15-5352)

Abstract:

To preserve their potency and efficacy, vaccines must be transported and stored according to precise temperature ranges. Consistent cold-chain conditions are challenging to maintain in many remote and developing regions due to limited infrastructure. Manual monitoring can result in vaccine spoilage and waste because it is frequently error-prone and lacks real-time responsiveness. In many cases, the vaccine got illegally traficked in different regions and sold in lesser prices than usual because there is no digital or even semi-digital system to monitor the supply-chain.

1. Introduction

The Smart Vaccine Box Tracker resolves a range of issues into a single tiny and affordable device. Based on NodeMCU ESP8266 microcontroller, it locates through a unit NEO-6M GPS and at the same time measures temperature through DHT11 sensor. Data is sent to the Arduino IoT Cloud every so often for real-time monitoring. This solution is designed to be easily modified for other types of cold-chain applications such as transportation for perishable food stuffs and drugs and is thus scalable and portable.

2. Objectives

- Temperature monitoring: The DHT11 sensor can used to sense and continuously monitor temperature inside the vaccine box.
- GPS tracking: The NEO-6M GPS is used to receive and send off co-ordinate data.
- Cloud Integration: The information that is collected will be uploaded to the Arduino IoT Cloud for logging, alerts and visualisation of the data.

- Cost Affectivity: Using readily available low-cost components allows the expansion.
- Alerting users: Our system will send alerts to user/manufacturers when the temperature crosses certain threshold values.
- Smart Dashboard for QR scanning: To ensure vaccine's safe delivery and route, a scanning verification mechanism.

3. System Architecture

3.1 Components Used

• **Microcontroller**: NodeMCU ESP8266 for Wi-Fi connectivity and processing.
• **Sensors**:
  • DHT11 for temperature and humidity sensing.
  • NEO-6M gps module for gps data.
• **Actuators**: Relays for controlling lights and appliances.
• **Cloud Platform**: Arduino IoT Cloud for data storing and communication.
• **Web Application-Dashboard**: Built using Arduino Iot cloud for live tracking and monitoring.
• **QR based verification system**: A QR-code based security portal for ensuring integrity of transported products.

3.2 Block Diagram

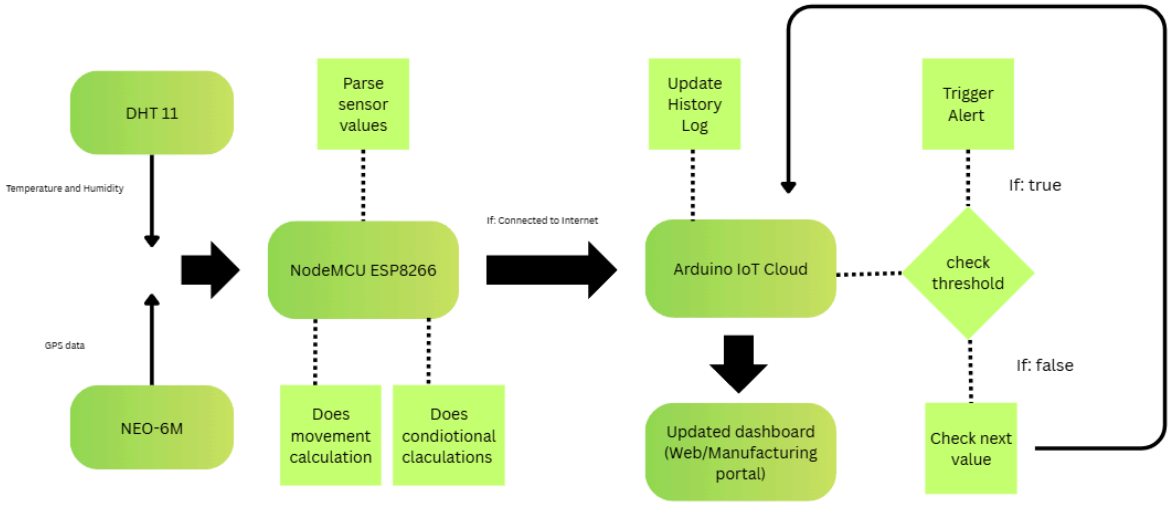

Figure 3.2.1: Overview of the Study's Methodological Workflow.

4. Methodology

4.1 Hardware Design

• **Microcontroller:** NodeMCU **ESP8266** with on-board Wi-Fi (3.3V logic).
• Sensors/Modules: **DHT11** for box temperature and humidity; **NEO-6M GPS** for latitude and longitude data.

Pin Map & Wiring:

- DHT11 **data to D4 (GPIO2)**, VCC to 3.3–5V (per module), GND to GND.
- GPS **TX to D5 (GPIO14)**, GPS **RX to D6 (GPIO12)** via **SoftwareSerial**; VCC to 5V, GND to GND.
- Common ground shared across all modules and short leads to reduce noise.

Power: A 5V adapter powers the GPS and a regulated 3.3V rail for ESP8266; disconnecting capacitors close to modules for achieving better and more stability if necessary.

Prototype Platform: A breadboard is used for setting up all the components for prototype phase building and testing purposes.

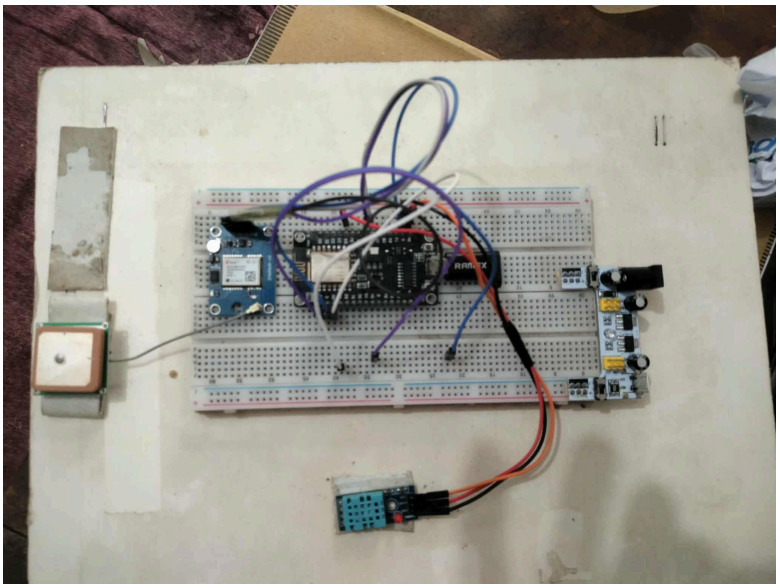

Figure 4.1.1: Prototype built on breadboard along with other components.

4.2 Software Design

- **IDE & Libraries:** Arduino IDE with TinyGPSPlus, SoftwareSerial, DHT, ArduinoIoTCloud, Arduino_ConnectionHandler, ESP8266WiFi.

- **Sampling and logic:** Main loop publishes every 5 seconds; ignores NaN sensor reads; parses GPS NMEA until a valid fix is available.

- **Alert Mechanism:** An alert mechanism is triggered if the parameters shows any sign of abnormality. The users will get an email for their products wellbeing and will get recommended on potential action to take.

- **Cloud Model (Arduino IoT Cloud):**
    1. **Properties:** temperature (float), location (lat, lon), optional alertFlag (bool), lastUpdate (String).

    2. Dashboard widgets: **Map** (location), **Gauge** (temperature), **Time-series graph** (history).

- **Mediverify (QR portal):**
    1. **QR code generation:** Generate QR code using python libraries.
    2. **Front-end:** HTML, CSS, JS
    3. **Backend:** NodeJs
    4. **Database:** MongoDB

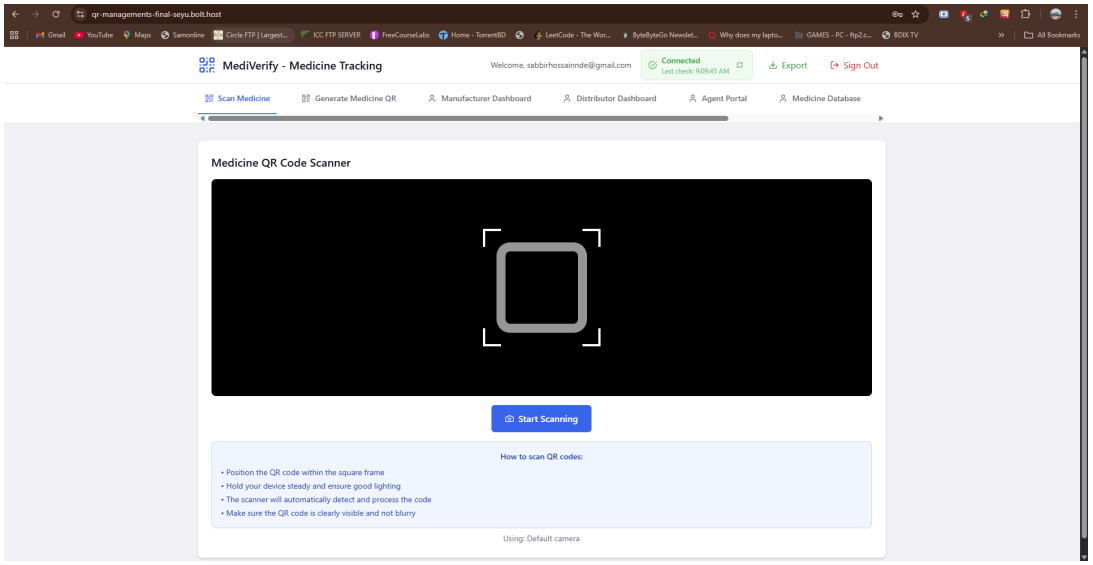

Figure 4.2.1: Scan option for general customers

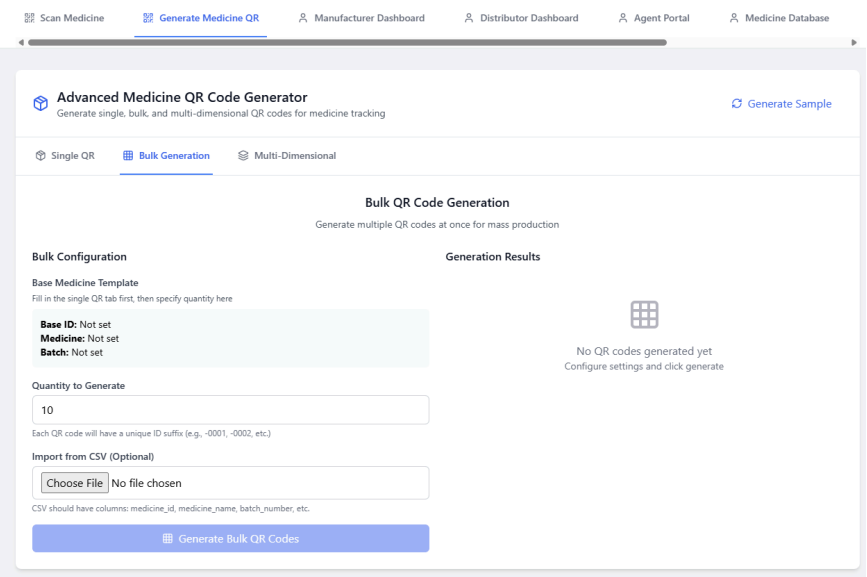

Figure 4.2.2: Bulk QR code generate option

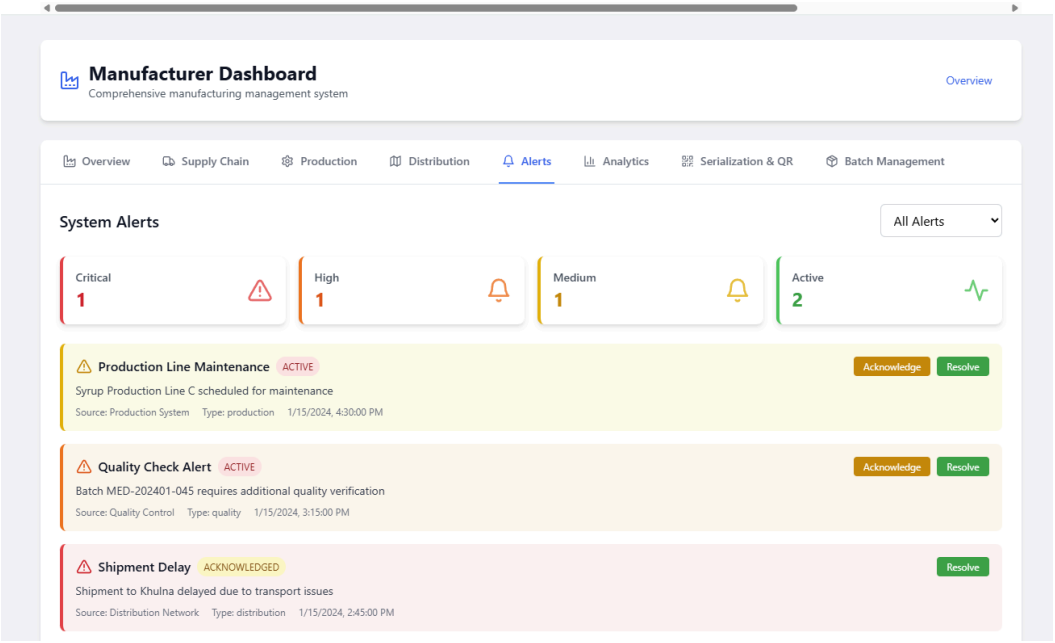

Figure 4.2.3: Manufacturer Dashboard (Alert section)

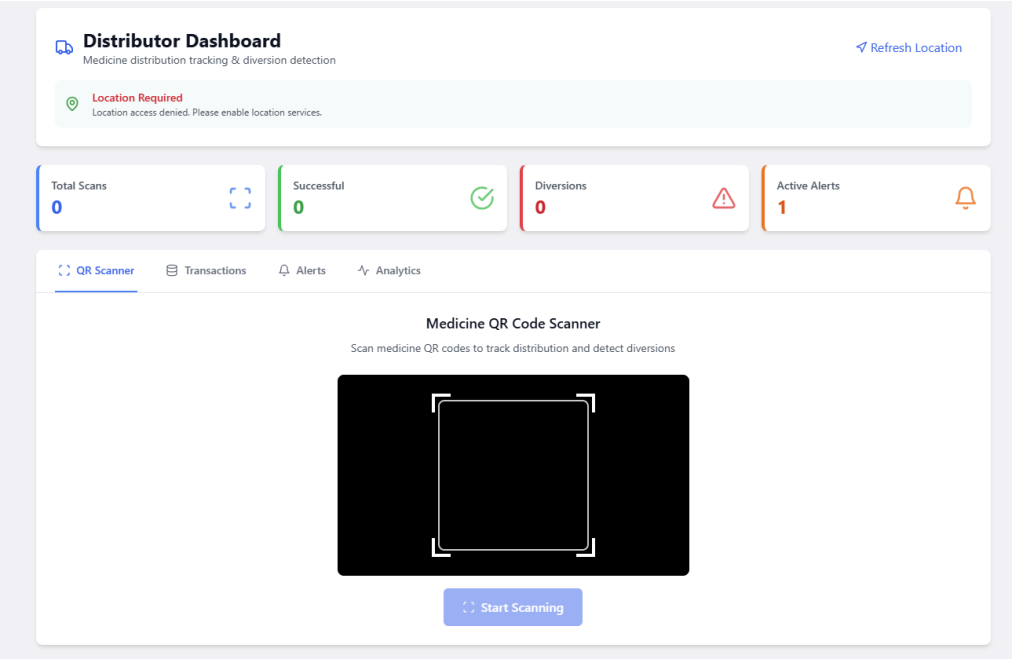

Figure 4.2.4: Location based scan option (Distributors)

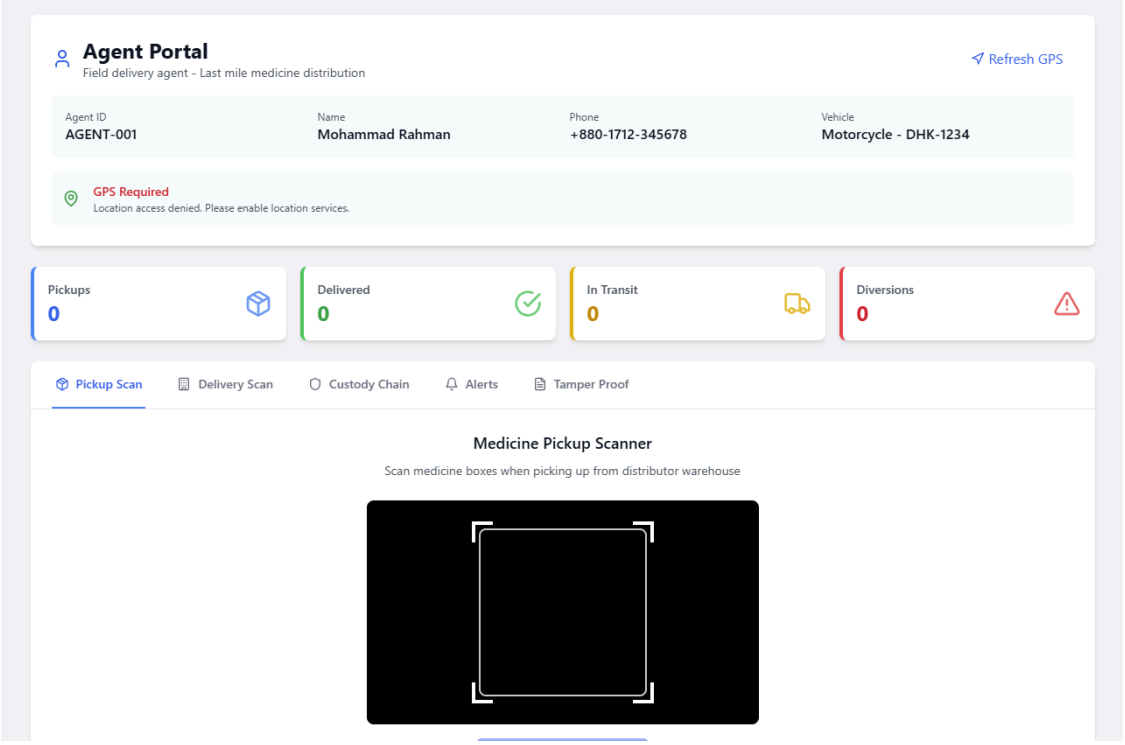

Figure 4.2.4: Location based scan option (agents)

4.3 Communication Protocol

• **Device (Cloud):** ESP8266 communicates over Wi-Fi to Arduino IoT Cloud using the platform's secure connection (TLS under the hood).

• **Update Policy:** Fixed 5-second gap for reading and sending new data from sensors to the cloud server.

5. Implementation

5.1 Sensor Data Acquisition

- **Read cycle:** The ESP8266 samples every 5 seconds (fixed interval) to balance the freshness of information and manage the power to consume as little as possible while maintaining useable quality.

- **Temperature (DHT11):** dht.readTemperature() is polled; invalid (NaN) readings are then discarded in order to avoid the noisy updates.

- **GPS (NEO-6M):** NMEA bytes are fed into TinyGPSPlus via the SoftwareSerial class; when a valid fix is present, the device then proceeds to update the cloud location variable with {lat, lon}.

- **Property updates:** On each cycle, the device sends temperature data reading and location reading to the Arduino IoT Cloud.The serial logs provides basic diagnostics.

- **Fallbacks:** If a GPS reading is somehow unavailable, the last known location is stored. If the Wi-Fi/cloud is temporarily unavailable, the firmware tries again with simple backoff before the next cycle.

- **Threshold:** A safe margin of temperature is set, which is between 2-8 degree celsius for vaccines.

- **Alert mechanism:** An alert (email and in dashboard) is sent to the manufacturer if the threshold value is not maintained. Used <ESP_mail_client.h> library for emailing an alert to the users.

5.2 Web Application Features

- Real-time panels include a temperature gauge, a map showing live position, and time-series graphs for historical trends

- Status indicators show a “Last update” timestamp and GPS fix status to help identify outdated data.

- Thresholds and alerts use cloud rules that can be set up to flag out-of-range temperature and unexpected movement. Notifications are sent via email, SMS, or push, depending on settings.

- Usability features a straightforward layout with large, clear widgets, making quick checks easy for health workers on their phones or the web.

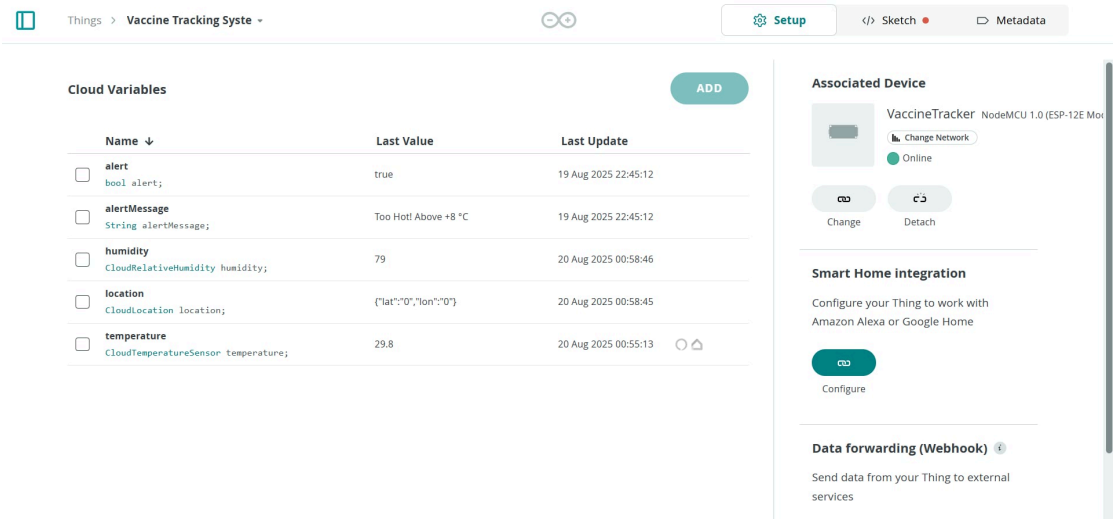

Figure 5.2.1: Cloud variables used to build the dashboard and alert system.

5.2.1 MediVerify (QR code verification portal)

- Scanning option for root level customers, manufacturers, agents and distrubutors.
- Easy csv upload option to generate qr code for a bulk quantity of medicine.
- Security scan for 3 layers - manufacturers, distributors, agents.

5.3 Security Features

- Secure connection: When our device sends the data to the Arduino IoT Cloud, it uses a secured line (TLS). So the data goes as encrypted, unfamiliar users can't easily read it or change it.

- Unique device ID: Arduino IoT Cloud gives our board a special device ID and secret. That means only our device is allowed to publish to our project. Others can't pretend to be us.

- Check bad data: Sometimes DHT11 gives NaN or GPS has no proper fix. In that case we don't upload it. This way wrong values don't enter our history.

- Wi-Fi safety: We connect with WPA2 Wi-Fi. We will not share Wi-Fi password or device secrets in public code/screenshots. In final demo, we also keep Serial.print limited, so the sensitive information don't leak.

- Dashboard privacy: Our Arduino IoT Cloud dashboard is currently a private prototype(only our account). If needed, we add only trusted people. This helps stop random or unauthorised users access.

6. Results

- **Live Data working:** Our NodeMCU sents the temperature value collected by the sensors and GPS data to Arduino IoT Cloud after every 5 seconds. On the dashboard, we can see that the values automatically updates without needing to refresh the page or restarting it.
- **Dashboard View:**
    1) The map widget showes us the current location of the vaccine box.
    2) The temperature gauge displayed the live reading (for example, around 28.6°C, as shown in the screenshot).
    3) The temperature graph plotted the line over time, allowing us to see fluctuations

easily.

- **Latency:** Latency issues: After the device pushes the data, it reaches the cloud, depending on the strength of Wi-Fi connection. For the class demo, this delay was acceptable.
- **Stability notes:**
    1) DHT11 can sometimes give some inaccurate readings, in that case we skip the NaN (not a number) values, so the dashboard doesn't shows error or sends false notification.
    2) GPS is usually faster under the open sky. In indoor situation, it takes a bit more time, or it might just show the last known location if the environment is too dense
- **Usability:** Widgets are large and clear; on the webview, it's easy to quickly check temperature and location. A health worker can understand the status at a glance.

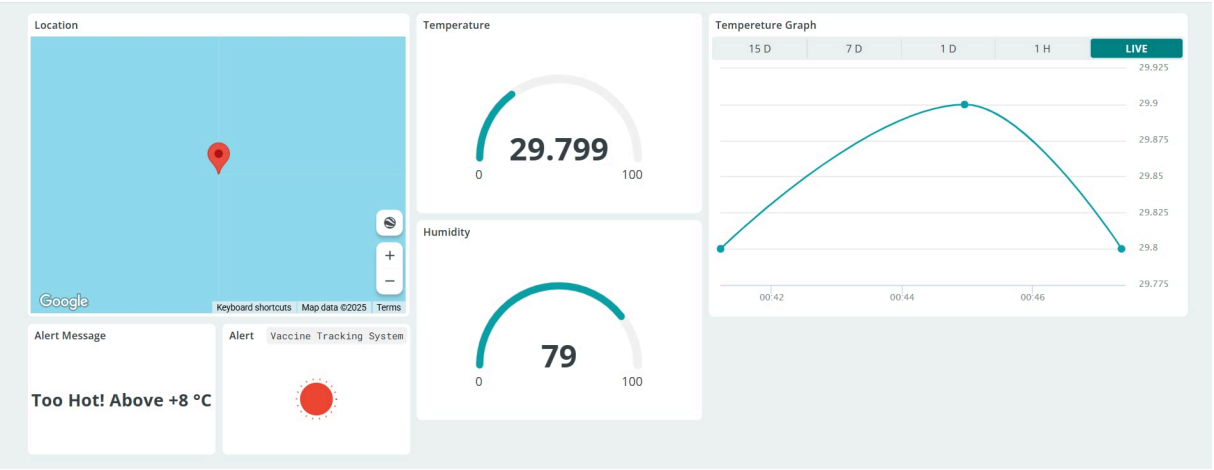

Figure 6.1: Dashboard on web UI

- **Smart Alert System:** We have set a certain threshold level of temperature for the vaccine and the range was 2 to 8 degree celsius. If the temperature crosses that range or stay behind it, the system will send notification to the user about potential spoilage of the vaccine.

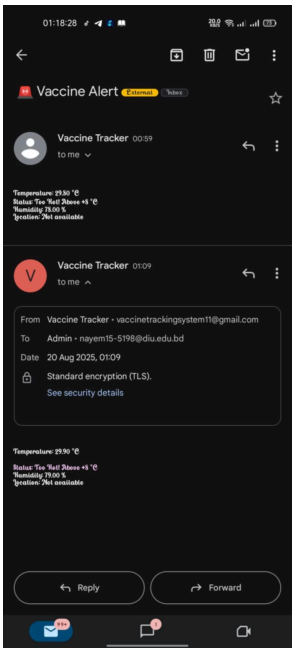

Figure 6.1: Email based alert for temperature

7. Challenges and Solutions

Challenge 1: Wi-Fi drops / Slow net / Connectivity Issue

Issue: Sometimes the prototype did not update the latest value or took too long to update it.

Solution: Reassembling the ESP8266 solved this issue for us.

Challenge 2: GPS cannot read the signals inside a clustered room environment.

Solution: Placed the near window/blank spot towards the sky to boost the signal of the NEO-6M module.

Challenge 3: Too frequent updates

Issue: The dashboard kept updating continuously, and it crashed the system.

Solution: Fixed interval of **5 sec,** and only pushed data on a valid reading. This is how we slowed the data showing rate in dashboard and prevent it from crashing.

Challenge 3: DHT11 giving wrong values

Issue: Sometimes, the DHT11 sensor gave us some undefined values that was mostly not a number.

Solution: To filter out values that are NaN, we made some airflow around the sensor by placing it differently. And is still the undefined values arrived, we added the filte-out mechanism in code. The system read t = dht.readTemperature(); and then update only if !isnan(t). Otherwise it printed "Failed to read DHT11" and don't send it to cloud.

8. Future Scope

- **Push notification (to add next):** In out current prototype, we only see the alerts on the dashboard and through email. The next step for us is to add the real-time push notification system in the prototype. This can be done by a phone app using the sim module in our IoT prototype. If the temperature goes out of the safe range or the box moves unexpectedly, the user will receive an instant pop-up or sound. This feature is not built yet, but we will integrate it after the class demo.
- **Better temperature sensor & small calibration**: The DHT11 is basic. Later, we will upgrade to the DHT22 or DS18B20 for more accurate readings in the vaccine range. We will conduct a quick calibration using an ice water test and room temperature comparison, and note the offset so the cloud shows a more accurate value.
- **Power saving + battery status:** To ensure long-lasting operation, we plan to use sleep and low-power strategies along with a simple battery percentage reading sent to the cloud via a voltage divider. We will also try long time gaps, such as 10 to 15 seconds, when stable to help reduce Wi-Fi power consumption.
- **Offline buffer & resend:** If the Wi-Fi goes down while on the road, the device will store the last N readings in EEPROM or SPIFFS and upload them later when the internet connection is restored. This way, we won't lose the temperature history during the trip.
- **Trip Report:** After successful delivery, a PDF or CSV report will be generated that includes the map's route and temperature graph along with location stamps and

timestamps, which will be useful for audits or submissions to health officers.

- **Integrating QR-security with IoT:** Currently, we are working to establish a link between the QR based verification portal and the IoT system of this project. This will enable this prototype to operate on a whole another level.

9. Conclusion

In this small project, we created a Smart Vaccine Box Tracker using ESP8266, DHT11, and NEO-6M GPS. We sent the data readings to the Arduino IoT Cloud. In Every five seconds, our device sent the temperature and the location data to the cloud. On the dashboard, we could see a live gauge, map, and graph. We eliminated our bad readings (NaN) and invalid GPS coordinates to keep the data clean for the demonstration in the dashboard. If the temperature crosses a certain level or drops below a certain level, the system alerts the user with a email based alert.

And in the QR part, we have enabled scanning on three different level to ensure maximum security and ensure that the vaccines are accessed from their targeted regions. Also a option to generate qr code for bulk quantity to

From this work, we have gained many practical skills. Such as wiring the sensors, parsing the GPS data, setting Thing Properties in the Arduino Cloud, and designing a simple dashboard that cold chain management related authority can check quickly. We also encountered some real problems like Wi-Fi drops, indoor GPS delays, and the DHT11 accuracy issues. We found some basic solutions such as retrying the connection, improving the antenna position, and skipping the bad readings.

Although the system is not perfect yet, but the main pipeline runs smoothly from start to finish. In the future versions, we are planning to add the push notifications, a better temperature sensor, some improved power-saving features, and an offline buffer to ensure data is never lost during the transit process. Overall, this project proved that even with the low-cost parts, we area able to create a functional IoT cold-chain monitor that is simple enough for class demonstrations and useful in real life as we make further improvements.

10. References

- Arduino IoT Cloud – Things, Properties, Dashboards & Alerts

https://docs.arduino.cc/arduino-cloud/

- Firebase Documentation ESP8266 Arduino Core Documentation (Wi-Fi, networking, examples)
  https://arduino-esp8266.readthedocs.io/

- Adafruit DHT Sensor Guide & Arduino Library (DHT11/DHT22)
  https://learn.adafruit.com/dht

- u-blox NEO-6 GPS Modules – Data Sheet / Product Page
  https://www.u-blox.com/en/product/neo-6-series