PROJECT REPORT

POWER PLANT EARTHQUAKE EMERGENCY SHUTDOWN SYSTEM

(OPTIONAL LAB)

SPECIAL TOPIC IN COMPUTER NETWORKS AND SECURITY
(SECR 4973)
SESSION/ SEMESTER: 2024/2025-2

SECTION: 03
GROUP: 10

GROUP MEMBERS:

SADIK AL MAHMUD	A20EC4049
MUNAIMIN BIN MONJUR	A20EC3026

Faculty of Computing

Universiti Teknologi Malaysia

30 JUNE 2025

DEDICATION

This project is dedicated to the engineers, scientists, and first responders who dedicate their lives to protecting critical infrastructure and saving lives during natural disasters. Their unwavering commitment to safety, resilience, and innovation serves as the foundation and inspiration for our work on this system.

ACKNOWLEDGEMENT

We extend our sincere gratitude to our instructor and lab supervisor for their continuous guidance, insightful feedback, and encouragement throughout the development of this project. We also thank the Faculty of Computer Science for providing a collaborative and resource-rich learning environment. Special appreciation goes to our peers for their support and cooperation, as well as to the developers of MongoDB and the Raspberry Pi Foundation, whose technologies empowered this practical exploration of cloud-based emergency systems.

ABSTRACT

This report outlines the complete development of an Earthquake Emergency Shutdown System (EESS), built using two Raspberry Pi devices integrated with a cloud-based MongoDB backend. The aim was to simulate an emergency alert system for power plants that can switch between safe and hazardous states based on user input and reflect those states in real time through visual indicators.

The project features a **Control Board**, where a physical push button updates the emergency state in the cloud database, and an **Emergency Board**, which polls this state and activates red or green LEDs to indicate system status. The integration of hardware with cloud-based services demonstrates the potential of using Internet of Things (IoT) and database technology in industrial safety applications.

CHAPTER 1 Introduction

In earthquake-prone regions, power plants must be equipped with fast and reliable shutdown mechanisms. Traditional safety systems are usually hardwired and difficult to modify, especially in legacy infrastructure. The goal of our project was to simulate a cost-effective, modular, and cloud-connected emergency shutdown solution that can be activated remotely or through physical controls.

To achieve this, we used two Raspberry Pi microcomputers, an online MongoDB database (MongoDB Atlas), and Python for backend scripting. The system was divided into two main roles: the Control Board (for initiating emergency status) and the Emergency Board (for responding to it visually). Together, these two boards form a small-scale but functional model of a distributed emergency signaling system.

CHAPTER 2 System Architecture

2.1 System Overview

The system is divided into two functional blocks:

- **Control Board (Pi 1)**: Contains a physical push button wired to a GPIO pin. When the button is pressed, it toggles the current `"alarm"` status in the MongoDB database between `"false"` (safe) and `"true"` (emergency).
- **Emergency Board (Pi 2)**: Continuously polls the database every 200 milliseconds. Depending on the alarm status, it lights either a red LED (emergency) or a green LED (safe). This serves as an immediate visual cue indicating the system's operational status.

2.2 Cloud Database Configuration

We used **MongoDB Atlas**, a cloud-hosted NoSQL database, due to its simplicity and flexible integration with Python through the `pymongo` library.

- **Cluster Name**: Cluster0 (Free-tier on AWS)
- **Database**: `earthquake_system`
- **Collection**: `alarm_status`

To initialize the database:

```
python
CopyEdit
from pymongo import MongoClient
MongoURI =
"mongodb+srv://admin:abcd1234@cluster0.ertrr3e.mongodb.net/?retryWrites=true&w=major
ity"
client = MongoClient(MongoURI)
db = client["earthquake_system"]
col = db["alarm_status"]
if col.count_documents({}) == 0:
    col.insert_one({"alarm": "false"})
```

Initial issues included URI formatting errors and character encoding problems.

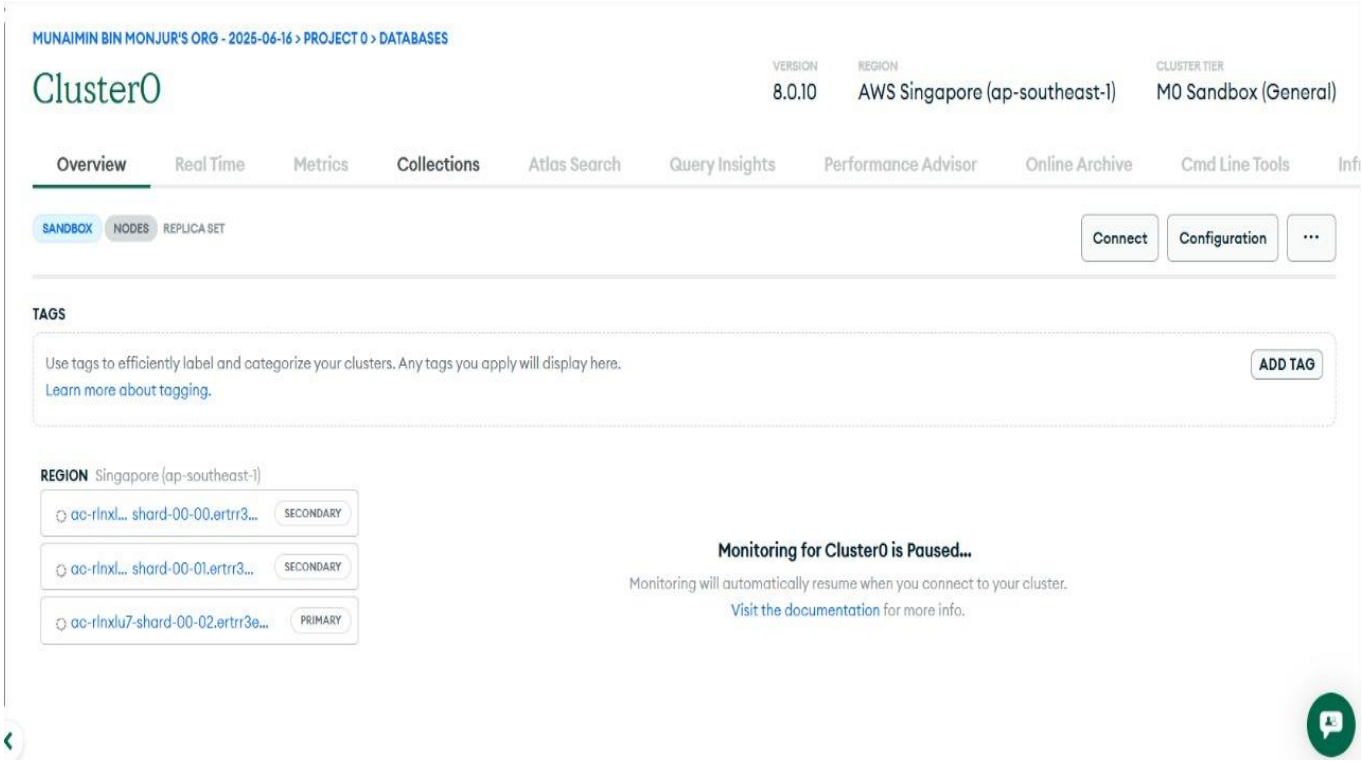

Figure 2-1 MongoDB Atlas Cluster Setup (Cluster0)

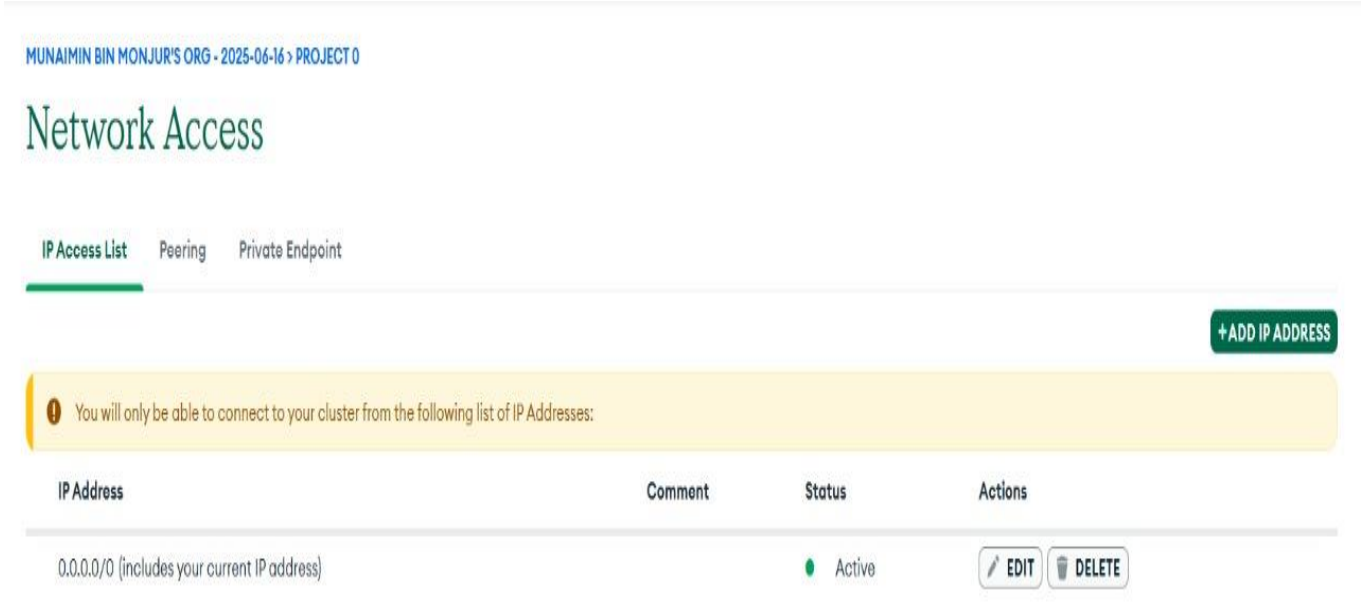

Figure 2-2 User Management Panel in MongoDB Atlas

MUNAIMIN BIN MONJUR'S ORG - 2025-06-16 > PROJECT 0

Database Access

Database Users | Custom Roles

+ ADD NEW DATABASE USER

User	Description	Authentication Method	MongoDB Roles	Resources	Actions
admin		SCRAM	readWriteAnyDatabase@admin	All Resources	EDIT DELETE

Figure 2-3 IP Whitelisting Configuration in MongoDB Atlas

In [8]:
```
db = client["earthquake_system"]
db_collection = db["alarm_status"]
```

In [9]:
```
# INITIAL DOCUMENT
db_collection.insert_one({"alarm": "False"})
```

Out[9]: InsertOneResult(ObjectId('685900d32e8b13d9841cfbb3'), acknowledged=True)

In [12]:
```
print("\n\n ------ THE CURRENT STATUS -------")

# FIRST MATCH
found = db_collection.find_one({"alarm": 'True'})
if found is None:
    alarm_status = 'False'
else:
    alarm_status = found['alarm']

print('Current State =', alarm_status)

```

```
 ------ THE CURRENT STATUS -------
Current State = False
```

Figure 2-4 Initial Database Insertion – "alarm: false"

CHAPTER 3 Hardware Implementation

3.1 Emergency Board (Handled by Us)

- **GPIO Setup**:
  - Red LED → GPIO 21 (Physical Pin 40)
  - Green LED → GPIO 20 (Physical Pin 38)
  - 330Ω resistors were used for each LED to prevent overcurrent.
- **Initial Issue**:
  - The green LED did not function correctly at first. After troubleshooting with a multimeter, we found a loose jumper wire connection. Once secured, the issue was resolved.
  - We also experienced confusion over correct GPIO pin numbering versus physical pin layout—solved by referencing Raspberry Pi pinout diagrams online.

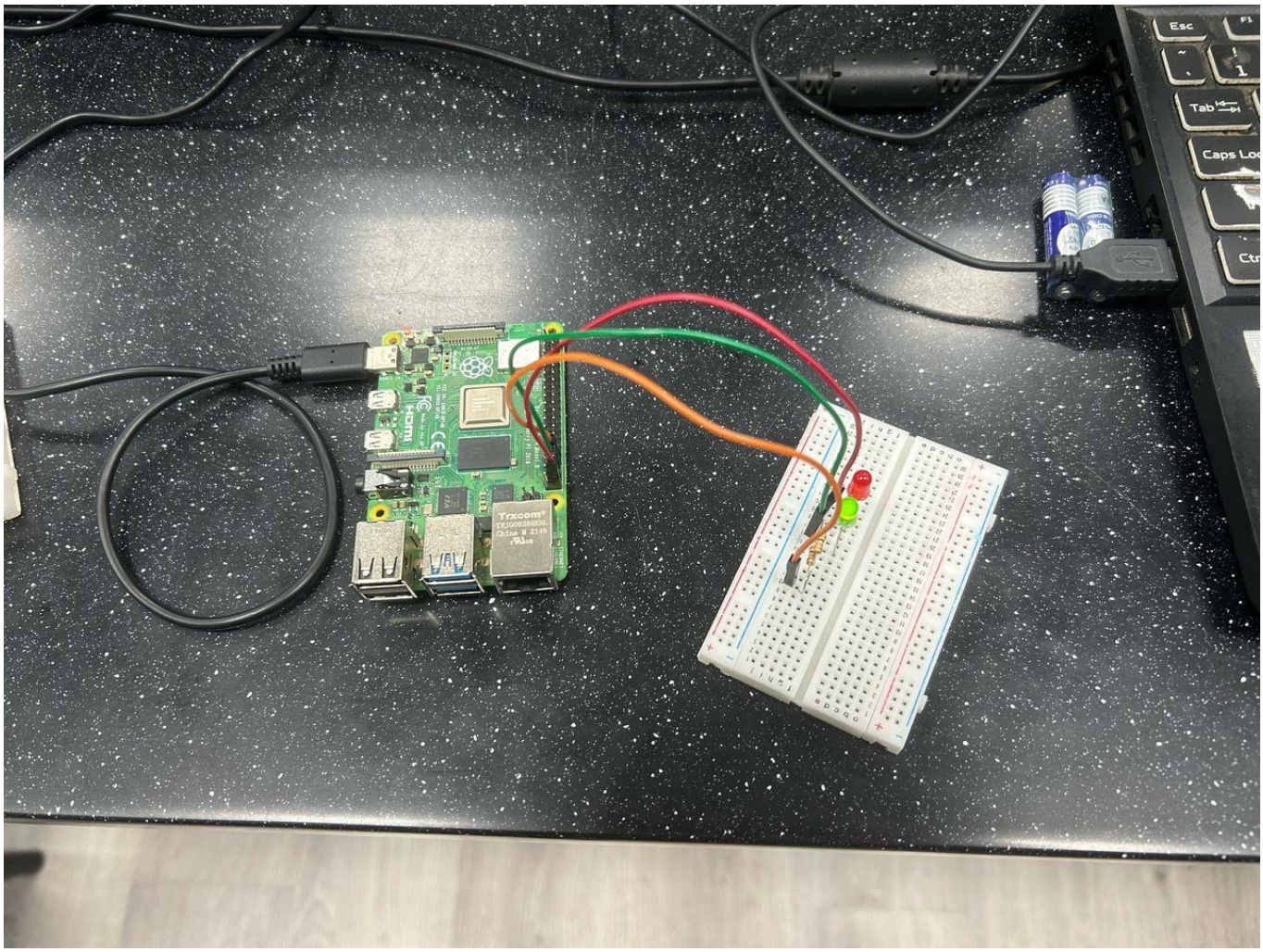

Figure 3-1 Assembled Emergency LED Circuit on Raspberry Pi

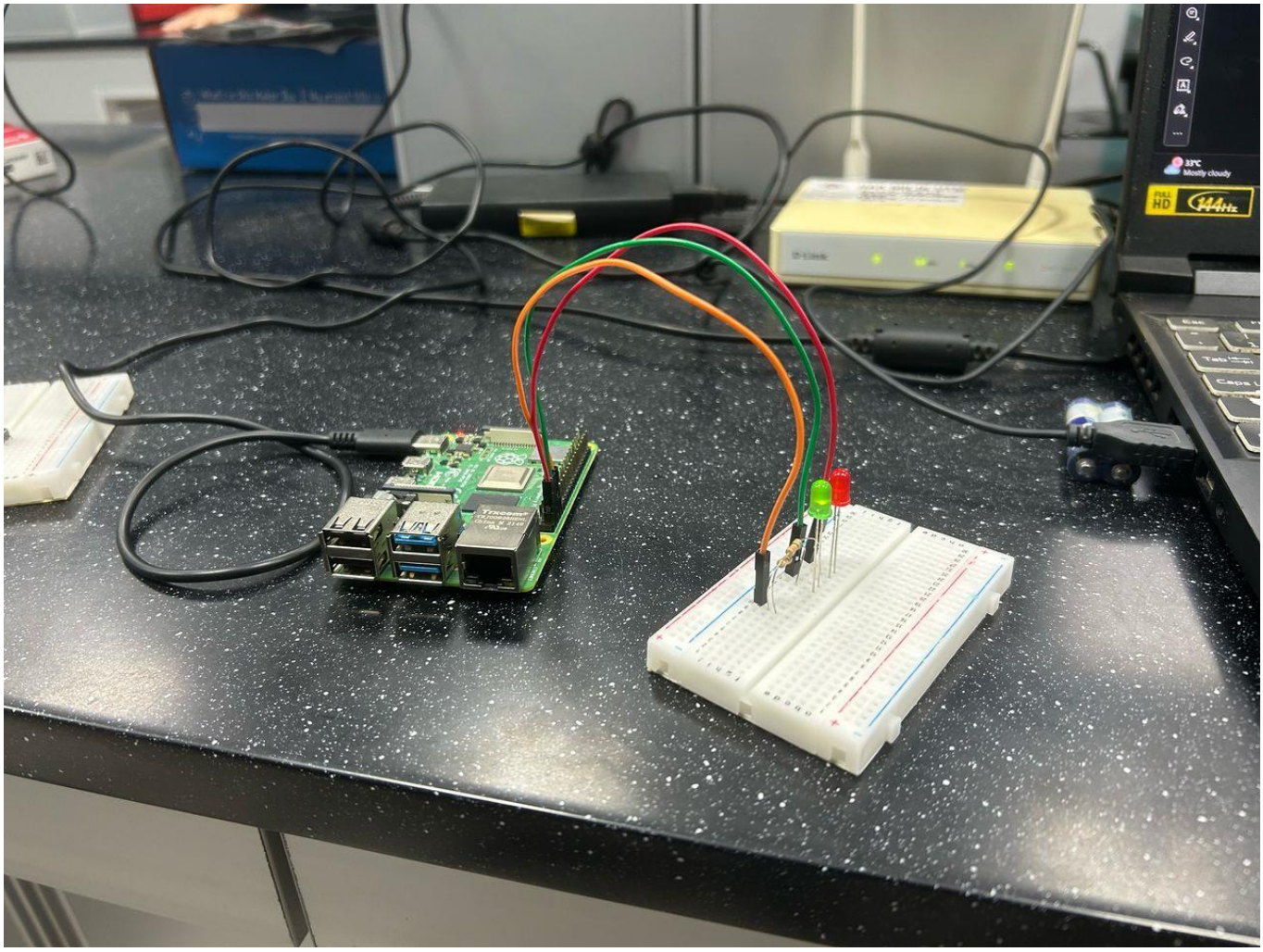

Figure 3-2 Final LED Output — Red and Green Indicators Working

3.2 Control Board (Handled by Groupmate)

- **GPIO Setup**:
  - Push Button → GPIO 21
  - A 10kΩ pull-down resistor was added to stabilize the input and avoid floating states.
- **Wiring Problems**:
  - Early on, we had problems with incorrect resistor placement, which prevented the button from triggering. This was fixed by cross-checking schematic diagrams and using a multimeter for verification.
- **Wi-Fi Issue**:
  - Initially, the PL App used to flash Raspberry Pi OS had network connectivity issues. Changing from an iPhone hotspot to a TP-Link router solved the issue and allowed us to configure the memory cards correctly.

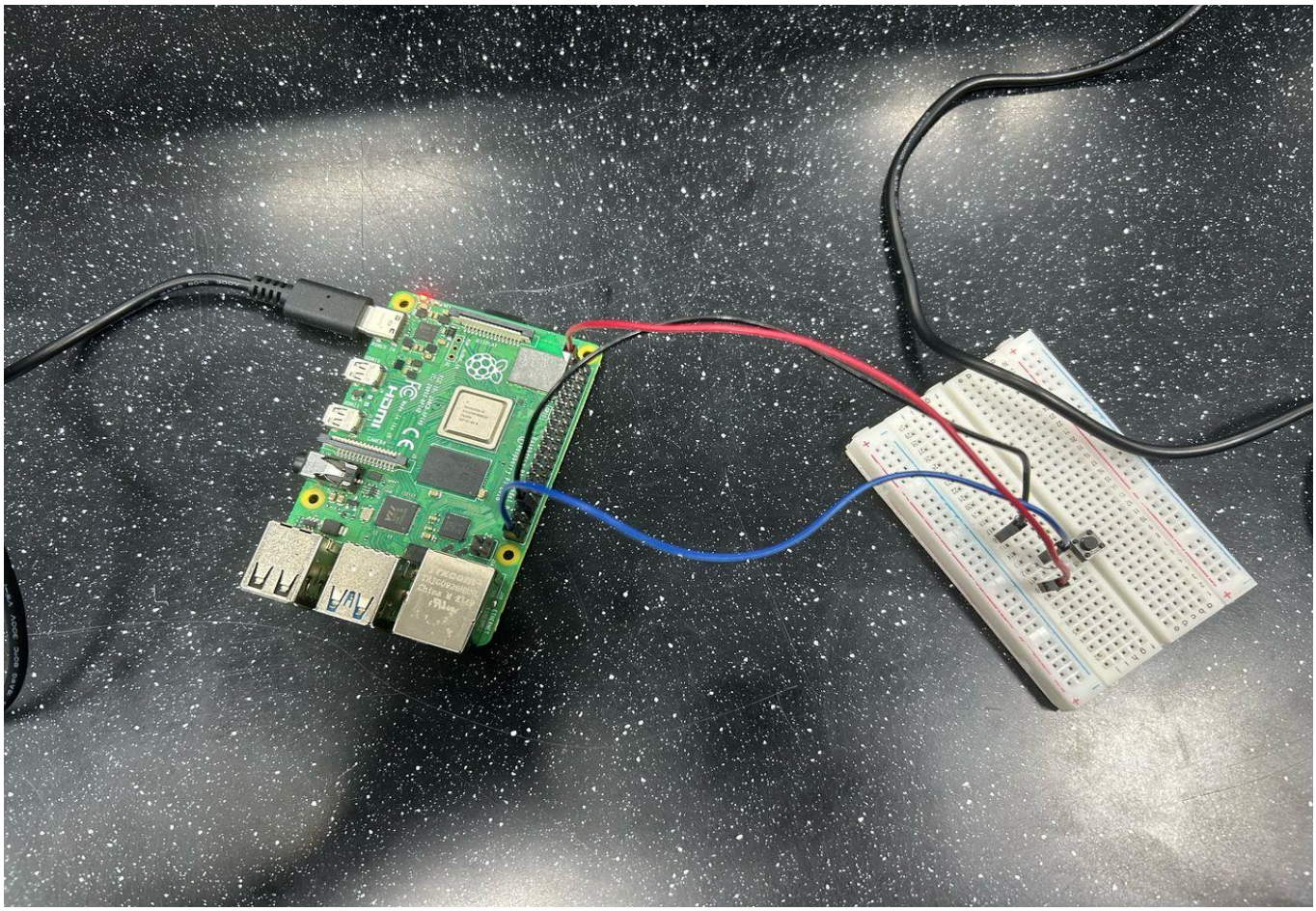

Figure 3-3 Button Setup for Emergency Triggering

Figure 3-4 GPIO Pin Setup for Control Board

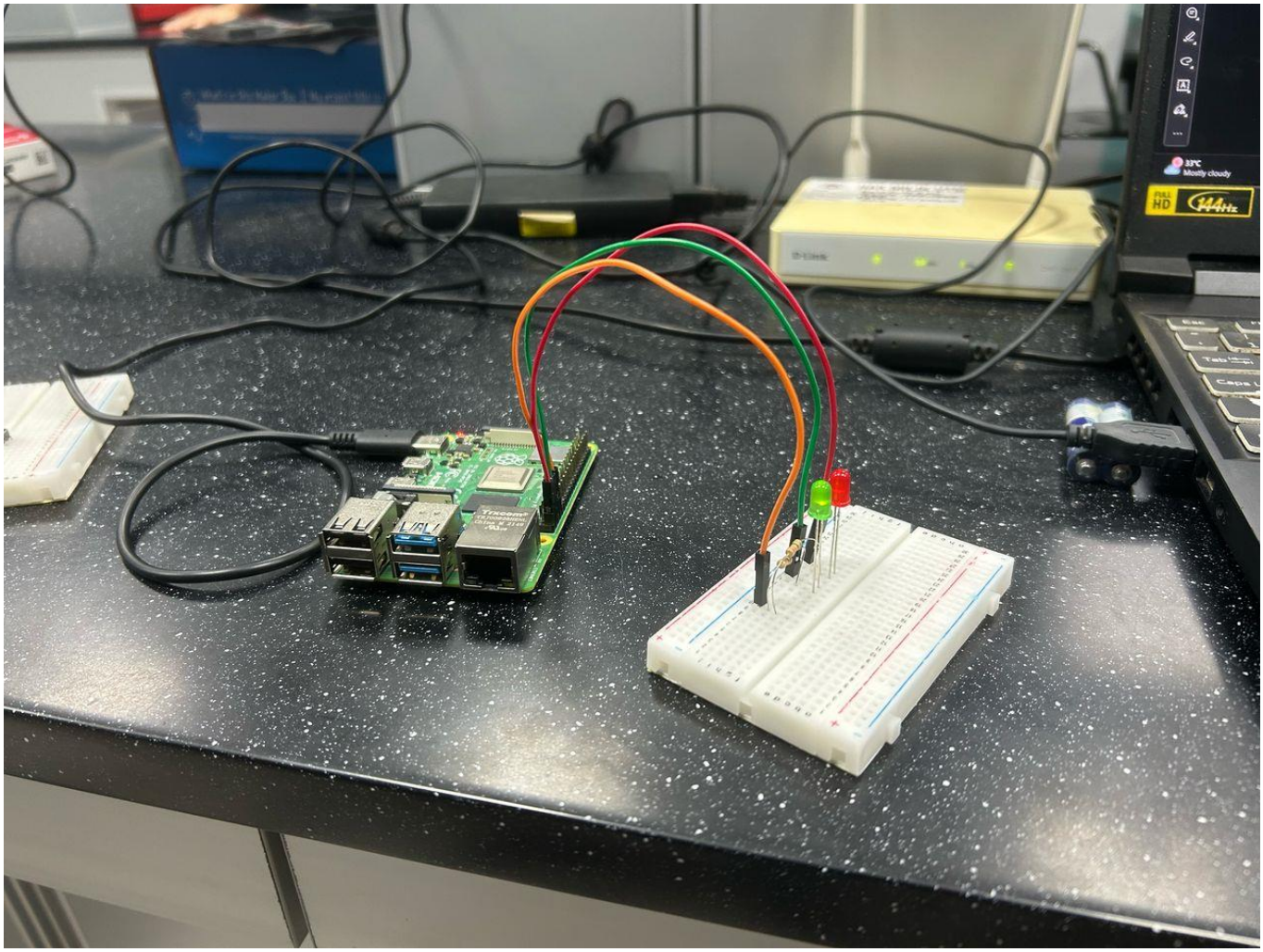

Figure 3-5 Control Board Fully Wired and Ready

CHAPTER 4 Software Implementation

All Python scripts were written and tested directly on the Raspberry Pi using Thonny IDE.

4.1 Control Board Script

Continuously monitors the button and toggles the "alarm" status in the MongoDB database:

python

CopyEdit

```
if curr_state == 1 and prev_state == 0:
    time.sleep(0.05)  # debounce
    if GPIO.input(21) == 1:
        doc = col.find_one()
        new_state = "false" if doc["alarm"] == "true" else "true"
        col.replace_one({"_id": doc["_id"]}, {"alarm": new_state})
```

Key considerations:

- Debouncing was critical to avoid false multiple toggles.
- The script normalizes any unexpected "alarm" values (e.g., uppercase or null) back to "false" to avoid misbehavior.

4.2 Emergency Board Script

Polls the alarm status every 0.2 seconds and sets LED outputs:

python

CopyEdit

```
if state == "true":
    GPIO.output(GREEN, False)
    GPIO.output(RED, True)
else:
    GPIO.output(GREEN, True)
    GPIO.output(RED, False)
```

Key considerations:

- Code checks for unexpected states in the database and resets them if necessary.
- Clean shutdown on KeyboardInterrupt was included to avoid GPIO pin locking.

In [2]:
```
!pip install pymongo dnspython
```

```
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting pymongo
  Downloading https://www.piwheels.org/simple/pymongo/pymongo-4.7.3-cp37-cp37m-linux_armv7l.whl (550kB)
     |████████████████████████████████| 552kB 209kB/s eta 0:00:01
Collecting dnspython
  Downloading https://files.pythonhosted.org/packages/12/86/d305e87555430ff4630d729420d97dece3b16efcbf2b7d7e974d11b0d86c/dnspython-2.3.0-py3-none-any.whl (283kB)
     |████████████████████████████████| 286kB 928kB/s eta 0:00:01
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.3.0 pymongo-4.7.3
```

In [5]:
```
from pymongo import MongoClient
from pprint import pprint
import warnings
warnings.filterwarnings('ignore')
```

In [7]:
```
from pymongo import MongoClient

MongoURI = "mongodb+srv://admin:abcd1234@cluster0.ertrr3e.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
client = MongoClient(MongoURI)
```

Figure 4-1 Installing and Importing Dependencies for Control Script

In [16]:
```
# SET NEW SEARCH VARIABLE
found = db_collection.find_one({"alarm": "True"})
if found is None:
    db_collection.replace_one({"alarm": "False"}, {"alarm": "True"})
    new_find = "True"
else:
    db_collection.replace_one({"alarm": "True"}, {"alarm": "False"})
    new_find = "False"

print('\n\n New Status')
pprint(db_collection.find_one())

print('\n\n ------ SET THE VARIABLE -------')

# RE-FIND THE FIRST MATCH USING THE new_find VARIABLE SET ABOVE
found = db_collection.find_one({"alarm": new_find})
alarm_status = found['alarm']
print('Sound the Alarm =', alarm_status)

```

```
 New Status
{'_id': ObjectId('685900d32e8b13d9841cfbb3'), 'alarm': 'False'}


 ------ SET THE VARIABLE -------
Sound the Alarm = False
```

Figure 4-2 Editing MongoDB Query Logic in Python Script

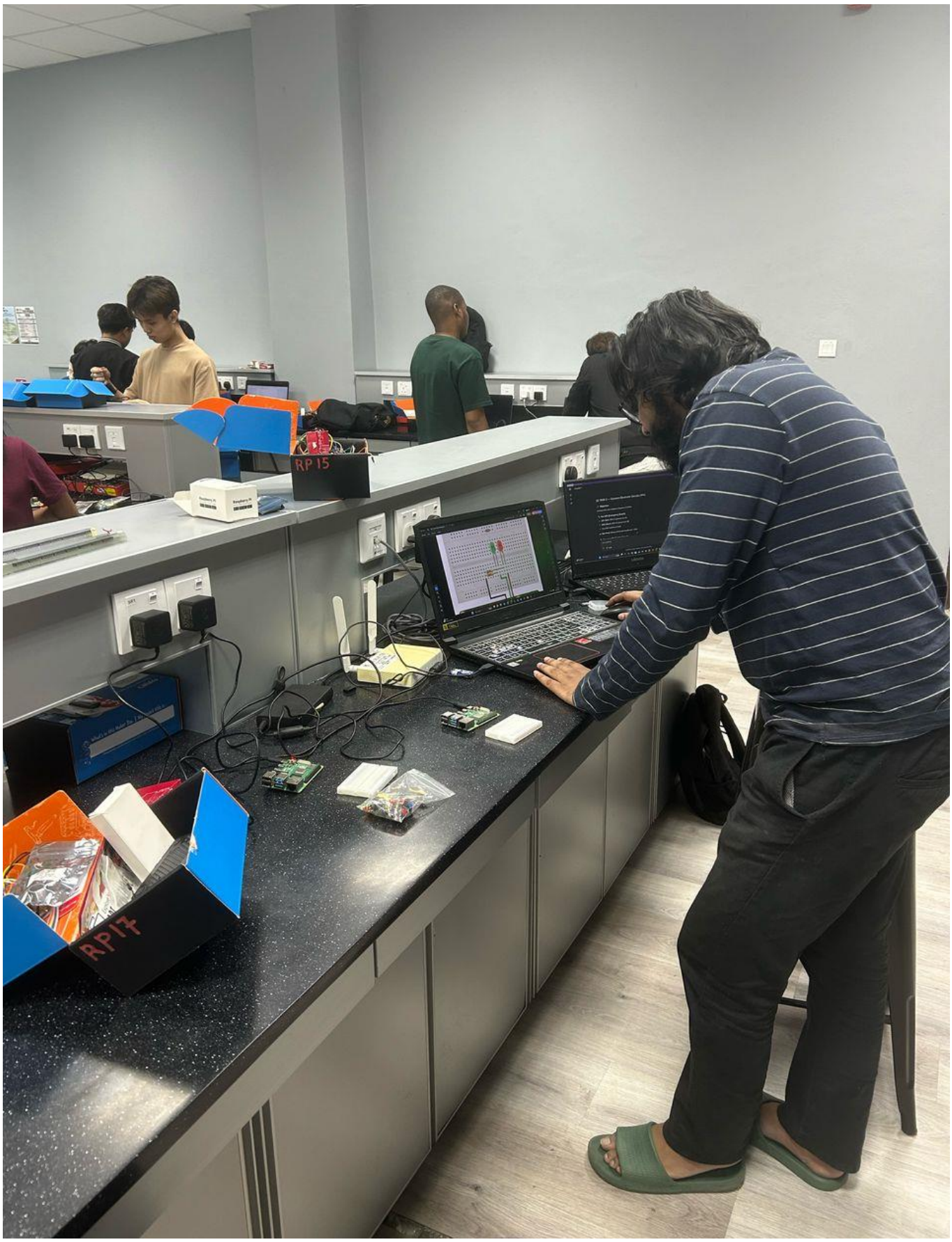

Figure 4-3 Live Coding and Debugging During Project Development

CHAPTER 5 Testing, Troubleshooting, and Results

Throughout development, we encountered and resolved several issues:

Issue	Resolution
PL App couldn't detect Raspberry Pi	Switched from iPhone hotspot to TP-Link router
Green LED not functioning	Fixed loose jumper cables
Incorrect resistor placement	Rewired using correct schematic
MongoDB URI issues	Used urllib.parse.quote_plus() for password encoding
False triggers	Added software debouncing and input validation

Once everything was connected and configured, the system worked reliably. We tested it over multiple Wi-Fi networks and different button press intervals. Response time was consistent at ~200ms.

CHAPTER 6 Discussion and Reflection

The project highlighted the strength of integrating **cloud infrastructure** with **physical hardware** for industrial safety applications. While this is a prototype, it demonstrates a viable pathway to a real-world solution.

However, there are limitations:

- The system depends heavily on internet connectivity. In regions with unstable networks, a local database or offline mode would be essential.
- MongoDB’s cloud latency could be an issue in real-time industrial settings unless deployed with regional replication.

Overall, the division of responsibilities helped in managing parallel development. One of us focused on the Control Board and database, while the other handled the Emergency Board and LED circuits. This collaborative workflow kept things efficient and helped us debug from both hardware and software perspectives.

CHAPTER 7 Conclusion

This Earthquake Emergency Shutdown System is a functional prototype showcasing how IoT and cloud-based services can be combined for emergency infrastructure control. Through Raspberry Pi, MongoDB, and Python, we built a responsive system capable of simulating real-time alerts and shutdown procedures.

The project enhanced our skills in hardware debugging, network configuration, cloud integration, and collaborative software development. We believe this concept can be expanded further into more robust industrial solutions.