SPECIAL TOPIC IN COMPUTER NETWORKS AND SECURITY

SESSION/SEMESTER: 2024/2025-2

Power Plant Earthquake Emergency Shutdown System

Project Report

Submitted by: Group 2

SADIK AL MAHMUD

MUNAIMIN BIN MONJOUR

Date: June 30, 2025

Submitted to: Faculty of Computer Science

Course: Special Topic in Computer Networks and Security

Dedication

This project is dedicated to our families and mentors who supported us throughout this endeavor. Their encouragement and guidance were invaluable in completing this work.

Acknowledgement

We express our gratitude to our instructor for providing clear guidance and resources for this project. We also thank the Faculty of Computer Science for access to the Prototyping Lab and the PL-App platform. Additionally, we appreciate MongoDB for offering a free-tier cloud database, which was critical to our project's success. Our teamwork and mutual support were instrumental in overcoming challenges and completing this project.

Contents

1 Introduction

The Power Plant Earthquake Emergency Shutdown System project aimed to simulate a factory emergency shutdown mechanism using two Raspberry Pi boards interfaced with a MongoDB cloud database via HTTP-based REST API calls. The system comprises a Control Board, which updates an alarm state in the database using a push button, and an Emergency Board, which monitors the alarm state and controls red and green LEDs to indicate emergency or safe conditions. This project demonstrates the integration of hardware, software, and cloud-based services to create a responsive and scalable system. The following sections detail the implementation of Tasks 1, 2, and 3 from the lab sheet, a reflection on the system's geographical limitations, and supporting appendices.

2 Task 1: Setting Up the MongoDB Cloud Database

Task 1 focused on establishing a MongoDB cloud database to store and manage the alarm state. The process began with registering a free-tier account at https://www.mongodb.com. After account creation, a new cluster was built using AWS as the cloud provider, selecting a free-tier region to ensure cost-free operation. The cluster was named ”EarthquakeSystem” for clarity.

A MongoDB user was created with the username ”admin” and a secure password, assigned ”Read and write to any database” privileges. The IP Whitelist was configured to allow access from anywhere, ensuring seamless connectivity from the lab environment. The connection URI was obtained and integrated into the Python code, with the password URL-encoded to handle special characters.

A database named ”earthquake$_{s}ystem” andacollectionnamed” alarm_{s}tatus” werecreated.Aninitialdocu$

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

print(”Inserted initial document: {’alarm’:’false’}”)

Challenges included ensuring the correct URI format and handling URL encoding for the password. These were resolved by referencing https://www.urlencoder.org and verifying the connection string.

3 Task 2: Connecting Electronic Circuits

Task 2 involved setting up the hardware components for the Control and Emergency Boards using Raspberry Pi 3 boards, breadboards, LEDs, a push button, resistors, and jumper cables.

3.1 Emergency Board Setup

The Emergency Board was configured to control two LEDs (red and green) connected to GPIO pins 21 (red, physical pin 40) and 20 (green, physical pin 38) using the BCM numbering scheme. Each LED was paired with a 330 Ω resistor to limit current. The circuit was assembled on a breadboard, with jumper cables connecting the LEDs to the Raspberry Pi. Initially, the green LED failed to illuminate due to a loose connection, which was resolved by re-securing the jumper cables and verifying continuity with a multimeter.

3.2 Control Board Setup

The Control Board featured a push button connected to GPIO pin 21 (physical pin 40) with a pull-down resistor to ensure a logical LOW state when unpressed. Pressing the button sent a 3.3V signal to the GPIO pin, registering a logical HIGH state. The setup was tested by pressing and releasing the button, confirming state changes in the console output. Issues with resistor placement were encountered, requiring careful reassembly to match the schematic.

Images of the circuit setups, included in the appendices, illustrate the connections and components used.

4 Task 3: Software to Connect the Dots

Task 3 involved developing Python scripts to integrate the hardware with the MongoDB database, enabling real-time alarm state updates and monitoring.

4.1 Emergency Board Software

The Emergency Board script continuously polled the MongoDB database for the alarm state and controlled the LEDs accordingly. If the database indicated {"alarm": "true"}, the red LED illuminated, signaling an emergency; otherwise, the green LED lit, indicating safety. The script used the RPi.GPIO module for GPIO control and included a 0.2-second polling interval to balance responsiveness and system load. The code snippet below outlines the implementation:

```
import RPi.GPIO as GPIO
GREEN = 20
RED   = 21
GPIO.setmode(GPIO.BCM)
GPIO.setup(GREEN, GPIO.OUT)
GPIO.setup(RED,   GPIO.OUT)
while True:
    doc = col.find_one()
    state = doc.get("alarm", "false")
    if state == "true":
        print("ALARM true → RED")
        GPIO.output(GREEN, False)
        GPIO.output(RED,   True)
    else:
        print("ALARM false → GREEN")
        GPIO.output(GREEN, True)
        GPIO.output(RED,   False)
    time.sleep(0.2)
```

An issue arose when the database returned an unexpected alarm value, which was mitigated by adding normalization logic to reset invalid states to "false".

4.2 Control Board Software

The Control Board script monitored the push button's state and toggled the alarm state in the MongoDB database on each button press (from ”false” to ”true” or vice versa). Debouncing was implemented with a 50ms delay to prevent false triggers. The script used the pymongo library to update the database and included error handling for unexpected states. The following code snippet demonstrates the implementation:

```
import RPi.GPIO as GPIO
BUTTON = 21
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
prev_state = GPIO.input(BUTTON)
while True:
    curr_state = GPIO.input(BUTTON)
    if curr_state == 1 and prev_state == 0:
        time.sleep(0.05)
        if GPIO.input(BUTTON) == 1:
            doc = col.find_one()
            current = doc.get(”alarm”, ”false”)
            new = ”true” if current == ”false” else ”false”
            col.replace_one({’_id’: doc[”_id”]}, {’alarm’: new})
            while GPIO.input(BUTTON) == 1:
                time.sleep(0.01)
            time.sleep(0.05)
        prev_state = curr_state
        time.sleep(0.01)
```

Challenges included initial code errors causing inconsistent database updates, which were resolved by refining the debouncing logic and ensuring consistent lowercase values in the database.

5 Reflection

The Power Plant Earthquake Emergency Shutdown System relies on an internet connection to interface with the MongoDB cloud database, which introduces geographical limitations. The Control and Emergency Boards can be located anywhere with reliable internet access and no filtering of port 443 (HTTPS), as required for MongoDB communication. This flexibility allows deployment across different locations, such as separate control rooms or facilities, provided both Raspberry Pi boards have stable network connectivity.

However, limitations arise in areas with poor internet infrastructure or strict firewall policies blocking port 443. In such cases, the system would fail to update or retrieve alarm states, rendering it ineffective. Additionally, latency in database queries could delay alarm responses in real-time applications, particularly if the boards are geographically distant from the MongoDB cluster's region. To mitigate this, selecting a MongoDB cluster region close to the deployment site is advisable. For critical applications, a local MongoDB instance could be considered, though this would require additional hardware and maintenance.

The project highlighted the importance of robust error handling and hardware reliability. Issues with loose connections and initial software bugs underscored the need for thorough testing and documentation in real-world implementations.

6 References

1. MongoDB. (2025). MongoDB Cloud Database. Available at: https://www.mongodb.com

2. Cisco Networking Academy. (2019). Power Plant Earthquake Emergency Shutdown System Lab. Faculty of Computer Science, 2024/2025.

3. URL Encoder. (2025). Available at: https://www.urlencoder.org

A Appendices

A.1 Distribution of Tasks

The project was a collaborative effort:

- Control Board and MongoDB Setup: Handled configuration of the push button circuit and MongoDB database integration, including user account creation and database initialization.
- Emergency Board Setup: Managed the LED circuit assembly and software development for monitoring the alarm state.

Both team members contributed to debugging, testing, and documentation, ensuring a cohesive project outcome.

A.2 Supporting Materials

- MongoDB Website: Screenshots of the MongoDB cluster setup and user configuration are included in the submitted softcopy.
- Teamwork Pictures: Photographs of the team assembling circuits and testing the system are included.
- Raspberry Pi Images: Images of the Control and Emergency Boards with connected circuits are provided.
- Video Demonstration: A video showcasing the system in operation, with the button toggling the alarm state and LEDs responding accordingly, is available at: https://example.com/video (placeholder link; actual link to be provided in submission).

A.3 Issues Encountered and Resolutions

- PL-App Configuration Issue: Initial difficulties with the PL-App for Raspberry Pi memory card setup were resolved by switching from an iPhone's Wi-Fi to a TP-Link Portal Router, ensuring stable connectivity.
- Green LED Failure: A loose connection in the Emergency Board's green LED circuit was fixed by re-securing jumper cables and verifying with a multimeter.

- Resistor Placement Errors: Incorrect resistor connections in both circuits were corrected by carefully following the schematic diagrams.
- Software Bugs: Inconsistent database updates were addressed by implementing debouncing in the Control Board script and normalizing alarm values in the Emergency Board script.