SYSC3010 Computer Systems Development Project
FANS Detailed Design Report

Group L1-G8

[1]

Grant Achuzia, 101222695
Javeria Sohail, 101197163
Matteo Golin, 101220709
Saja Fawagreh, 101217326
TA: Sean Kirkby

Created: March 11th, 2024
Modified: March 13, 2024

Contents

1 Problem Statement

The need for the Fire Alarm Notification System (FANS) stems from the alarming number of preventable fire deaths in Canada, where 220 people die in fires each year, with at least one in seven of these deaths occurring in homes without working smoke alarms [2]. This critical problem underscores the shortcomings of traditional fire alarm systems, which often lack advanced communication capabilities and real-time monitoring capabilities [3]. These limitations not only contribute to delayed response times, but also to preventable deaths, underscoring the urgent need for an advanced fire alarm solution.

Traditional systems' shortcomings, coupled with the fact that smoke detection systems are sometimes unsafely disarmed by users to avoid false alarms—especially those installed close to kitchen spaces—further exacerbate the problem. The FANS project seeks to address these issues by integrating smoke and temperature sensors with Internet of Things (IoT) technology. This approach not only aims to cover scenarios where smoke may not reach the detecting device but also offers real-time notifications via SMS and email, thus notifying homeowners immediately in the event of an emergency.

By providing configurable thresholds for smoke and temperature alarms and adjustable timeouts that prevent the system from being deactivated in an unsafe manner, FANS aims to use technological advances to significantly improve the effectiveness of fire detection systems. The ultimate goal of the project is to reduce response times, improve overall fire safety and thereby mitigate fire disasters and protect the well-being of individuals, families and communities across Canada [4].

1.1 Functional Requirements

1. **Accurate and Reliable Detection**
   Implement dual-sensor technology combining photoelectric and thermistor-based sensors for accurate smoke and temperature detection, alongside self-testing routines to ensure reliability.
2. **Real-time Monitoring and Alerts**
   Utilize a cloud-based platform for continuous real-time data analysis, with a reliable communication channel for immediate alert dispatch.
3. **User-configurable Settings**
   Provide an intuitive interface for users to customize alarm thresholds and notification preferences, including quiet hours and primary notification channels.
4. **Comprehensive Notification System**
   Initiate on-site alarms and send email to users and designated contacts to ensure widespread awareness during emergencies.
5. **Integration with Emergency Services**
   Facilitate automatic emergency service alerts upon fire detection, including crucial details like location and building type, to expedite response times.

By adopting these key requirements, FANS aims to deliver a technologically advanced, user-friendly system for timely fire detection and response, aiming to reduce fire-related fatalities and enhance safety for individuals and communities across Canada.

2 Design Overview

The Fire Alarm Notification System (FANS) is a real-time system designed for efficient fire detection and alerting. The system is composed of several key components, each with a distinct function but integrated to work seamlessly together, ensuring a comprehensive approach to fire safety. The system's architecture is outlined in a detailed deployment diagram in Figure 1, illustrating the interconnections between the various components. These components include:

- **Alarm System:** Based on a Raspberry Pi 4, this system triggers audible and visual alerts in the event of a fire, ensuring that occupants are promptly warned.
- **Smoke Detection System:** Also utilizing a Raspberry Pi 4, this system constantly monitors the environment for smoke using advanced sensors. It is the primary detection mechanism that activates the alarm system and notification system upon detecting smoke.
- **Notification System:** Operating on a Raspberry Pi 4, this system sends out emergency alerts to predefined contacts, including both local authorities and individuals, ensuring rapid response to the detected fire.
- **Cloud Database:** A real-time cloud database is employed to store critical data, including sensor readings (smoke and temperature levels) and system configurations. This facilitates remote monitoring and configuration, ensuring that the system is always functioning optimally.
- **Haptic Alarm:** This device provides haptic feedback, offering an additional layer of alert through physical sensation, ensuring that even those who may not be within hearing range of the alarm or have hearing impairments are alerted.
- **User Interface:** A web-based application for monitoring the system's environment and configuring system settings.

The system's design emphasizes scalability, allowing for easy expansion with additional sensors or alarm units as needed. It also highlights modularity, where each component can function independently, ensuring reliability and ease of maintenance. Communication between the components is facilitated through a local network, with the cloud database supporting remote access and control.

Furthermore, the system incorporates advanced communication protocols for efficient data exchange and alerting. The smoke detection system employs the I2C protocol for sensor communication, while UDP packets facilitate local network communication between the system's nodes. HTTP requests are utilized for interactions with the cloud-hosted real-time Firebase database, ensuring timely updates and access to system data.

Overall, the Fire Alarm Notification System is designed to be a reliable, efficient, and scalable solution for fire detection and notification, leveraging modern technology to provide real-time alerts and ensuring the safety of occupants in any building.

2.1 System Overview Design

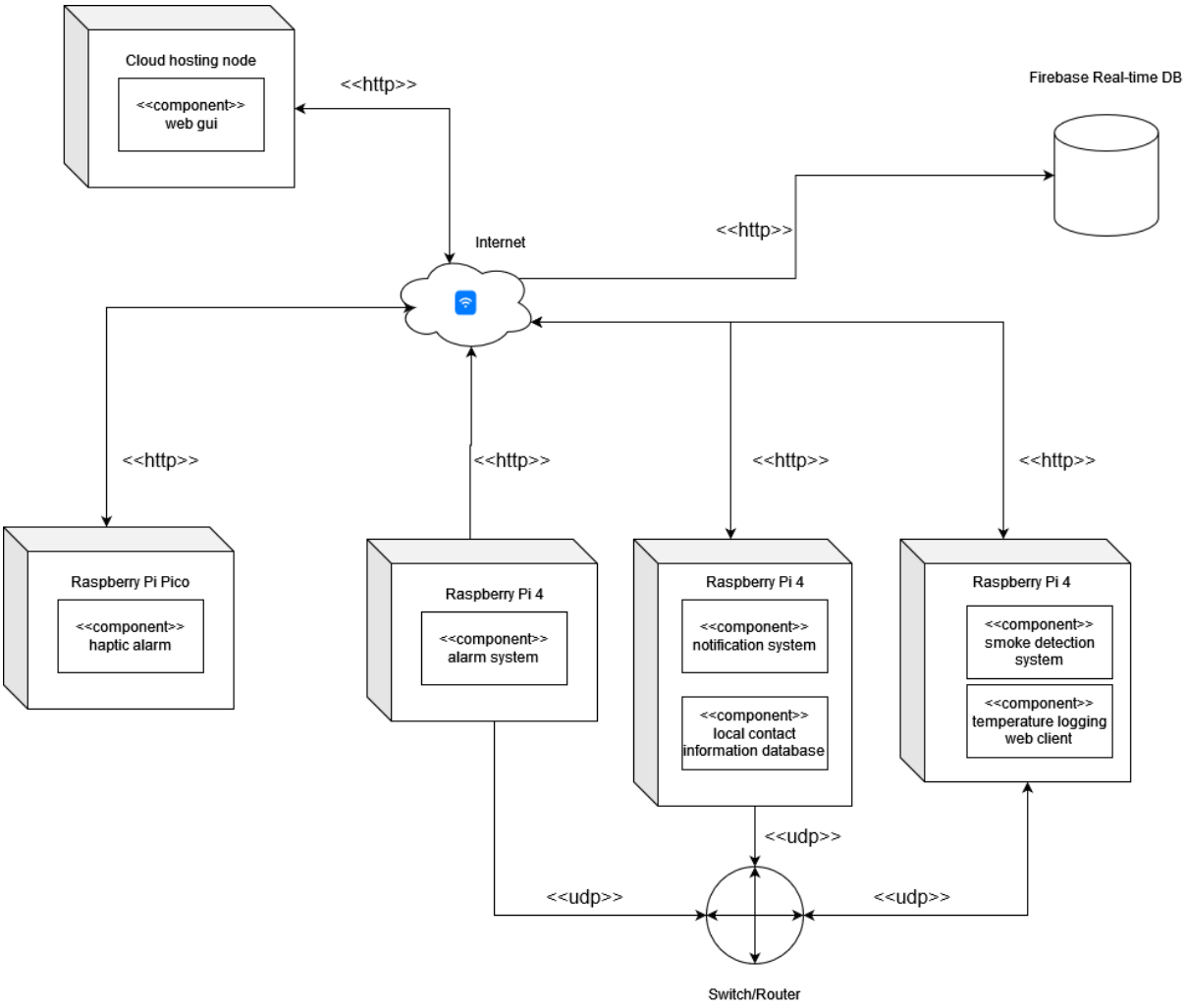


Figure 1: The deployment diagram for FANS.

2.2 Communication Protocols

In the Fire Alarm Notification System (FANS), a variety of communication protocols are meticulously integrated to ensure seamless interaction among the system components and with the external cloud database, facilitating a robust and responsive fire alarm solution.

The system's core, the Smoke Detection System, communicates with its temperature and smoke sensors using the I2C protocol over the GPIO pins of a Raspberry Pi 4. This protocol choice is pivotal for real-time data acquisition, allowing the system to monitor environmental conditions continuously and detect any signs of fire immediately. For interactions within the local network, including communication between the smoke detection system, the alarm system, and the notification system, UDP packets are employed. This approach is selected for its efficiency and speed, ensuring that critical data is transmitted quickly and reliably across the system components without the overhead of establishing and maintaining a connection, which is crucial in emergency situations

where every second counts.

The integration with the cloud is achieved through HTTP requests to a Firebase real-time database. This cloud database is essential for storing sensor data, system configurations, and user information. It supports various operations, including the posting of sensor data by the smoke detection system, retrieval of data for the web GUI, fetching contact information for the notification system, and polling for emergency flags by the haptic alarm system. These interactions are facilitated by HTTP's flexibility and its widespread support across internet infrastructure, enabling the FANS to leverage cloud computing benefits for enhanced data management and accessibility.

Lastly, the notification system's ability to reach users through email and SMS text notifications is realized through standard internet SMS and email protocols. This ensures that in the event of a fire, users are promptly informed regardless of their location, providing critical information and instructions to enhance their safety and response effectiveness.

Together, these communication protocols form the backbone of the Fire Alarm Notification System, enabling it to function as a cohesive, efficient, and highly responsive fire safety solution.

The Fire Alarm Notification System (FANS) employs a variety of communication protocols to ensure seamless interactions between its components and with the external cloud database. This section provides detailed tables describing each communication pathway.

2.2.1 I2C Communication

The Raspberry Pi 4 utilizes the I2C protocol to communicate with an array of temperature and smoke sensors, monitoring environmental conditions to detect potential fire hazards. This is not visible in the sequence diagrams as they show the bigger picture of the communication of the nodes with the sensors.

Sender	Receiver	Message	Data Format	Protocol
Raspberry Pi 4	Temperature Sensor	`read_temp`	See section 6.2.1 of datasheet [5]	I2C
Raspberry Pi 4	Smoke Sensor (via ADC)	`read_smoke`	See figure 1.1 of datasheet [6]	SPI [6]

Table 1: Messages for I2C communication in FANS.

2.2.2 Local Area Network Communication

Nodes within the FANS (smoke detection, alarm, and notification systems) communicate over a local network using UDP packets, facilitating real-time alerts and system coordination.

The messages sent over UDP use numerical value to encode messages. The representation agreed upon is as follows:

Message	Value
Emergency	0
No Emergency	1

Table 2: Numerical representation of messages over UDP in FANS.

Sender	Receiver	Message	Data Format	Protocol
Smoke detection system	Notification system	Emergency	0	UDP
Smoke detection system	Alarm system	Emergency	0	UDP
Smoke detection system	Notification system	No emergency	1	UDP
Smoke detection system	Alarm system	No Emergency	1	UDP

Table 3: Messages for local area network communication in FANS.

2.2.3 Cloud Database Communication

Sender	Receiver	Message	Data Format	Protocol
Smoke Detection	Cloud DB	`put_sensor_data()`		HTTP (JSON)
GUI	Cloud DB	`update_threshold(new_threshold)`	See listing 1	HTTP (JSON)
Notification	Cloud DB	`query_contact_information()`	See listing 2	HTTP (JSON)
Haptic Alarm	Cloud DB	`emergency()`	See listing 3	HTTP (JSON)

Table 4: Messages for cloud database communication in FANS.

Listing 1: Threshold update message.

```
{
  "method": "PUT",
  "path": "/system/threshold",
  "headers": {
    "Authorization": "Bearer YOUR_ACCESS_TOKEN",
    "Content-Type": "application/json"
  },
  "body": {
    "newThreshold": 50
  }
}
```

Listing 2: Request for user contact information.

```
{
  "method": "GET",
  "path": "/user/contact",
  "headers": {
    "Authorization": "Bearer YOUR_ACCESS_TOKEN"
  }
}
```

Listing 3: Request for emergency flag.

```
{
  "method": "GET",
  "path": "/system/emergency",
  "headers": {
    "Authorization": "Bearer YOUR_ACCESS_TOKEN"
```

```
    }
}
```

2.2.4 User Notification Communication

The notification system communicates with users through email, employing standard internet protocols to ensure timely and effective alerts.

Sender	Receiver	Message	Data Format	Protocol
Notification System	User Inbox	Emergency notification	See listing 4	SMTP (Email)

Table 5: Messages for user notification communication in FANS.

Listing 4: Email notification for detected emergency in FANS.

```
FROM: notification@example.com
TO: user@example.com
SUBJECT: Fire Alarm Notification

Dear [User's Name],

Fire alarm detected. Evacuate immediately.

- Location: [Location]
- Date/Time: [Date/Time]

Stay safe!
```

2.3 Message Sequence Diagrams

To implement the functional requirements described in Section 1.1: Functional Requirements, the FANS system will satisfy six core use cases highlighted and illustrated in the message sequence diagrams below.

2.3.1 Trigger Emergency Use Case

The Trigger Emergency Use Case represents the critical functionality of the FANS, where the system detects a potential fire through its smoke detection system and initiates a series of automated responses to mitigate the situation. As depicted in the message sequence diagram, the process begins when the smoke detection system identifies smoke and potentially high temperatures indicative of a fire. This detection triggers the system to send a notification to the alarm system and the notification system. The alarm system responds by sounding an audible and visible alert to notify occupants of the building immediately. Concurrently, the notification system sends out emergency alerts via SMS and email to all registered users, ensuring they are informed of the danger regardless of their current location. This response ensures that all the users are promptly alerted to the emergency, resulting in a quick evacuation and response to the detected threat.

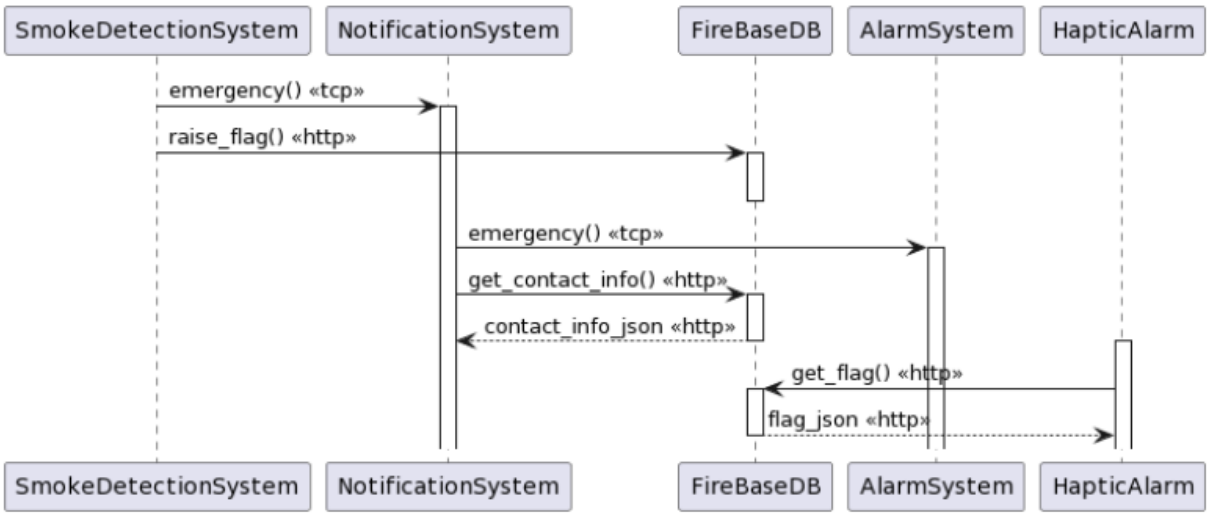


Figure 2: Sequence diagram for the trigger emergency use case.

2.3.2 Add New Contact Information Use Case

This use case outlines the process by which users can add new contact information to the FANS database. Through the web GUI, a user enters new contact details, such as phone numbers and email addresses, that the system can use to send emergency notifications. Upon submission, these details are updated in the real-time database. The message sequence diagram for this use case (not shown) illustrates the interactions between the user, the web GUI, and the database, culminating in the notification system being updated with the new contact information. This ensures that the FANS can reach the user through various channels in the event of an emergency, enhancing the system's effectiveness in communicating critical alerts.

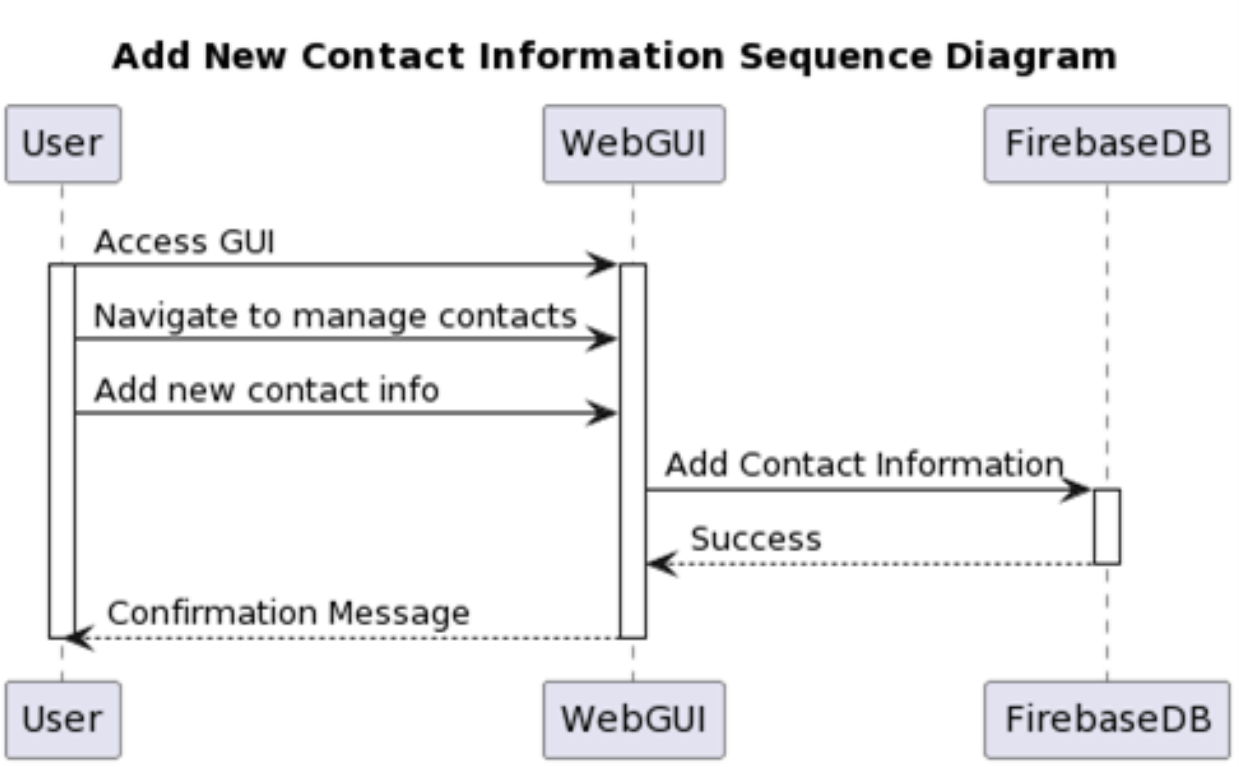


Figure 3: Sequence diagram for the add contact info use case.

2.3.3 Change Emergency Threshold Use Case

Adjusting the smoke detection threshold is an essential feature that allows users to customize the sensitivity of the FANS based on environmental conditions and personal preferences. This use case involves a user accessing the web GUI to modify the threshold settings that determine when the smoke detection system should trigger an alert. The message sequence diagram (not shown) visualizes the steps involved, from the user's interaction with the GUI to update the threshold to the real-time database's role in storing this new setting. The smoke detection system then retrieves and applies the updated threshold, ensuring that the FANS operates according to the user's specifications, balancing sensitivity and false alarm minimization.

Figure 4: Sequence diagram for the change threshold use case.

2.3.4 Emergency Response Use Case

This scenario highlights the FANS's capability to notify users in the event of a detected fire. Upon detecting a fire, the notification system queries the cloud database for contact information and then sends out alerts via SMS and email to all users. The message sequence diagram for this use case (provided above) captures the sequence of these interactions, showcasing how the system ensures that occupants are informed and ready to address the emergency promptly.

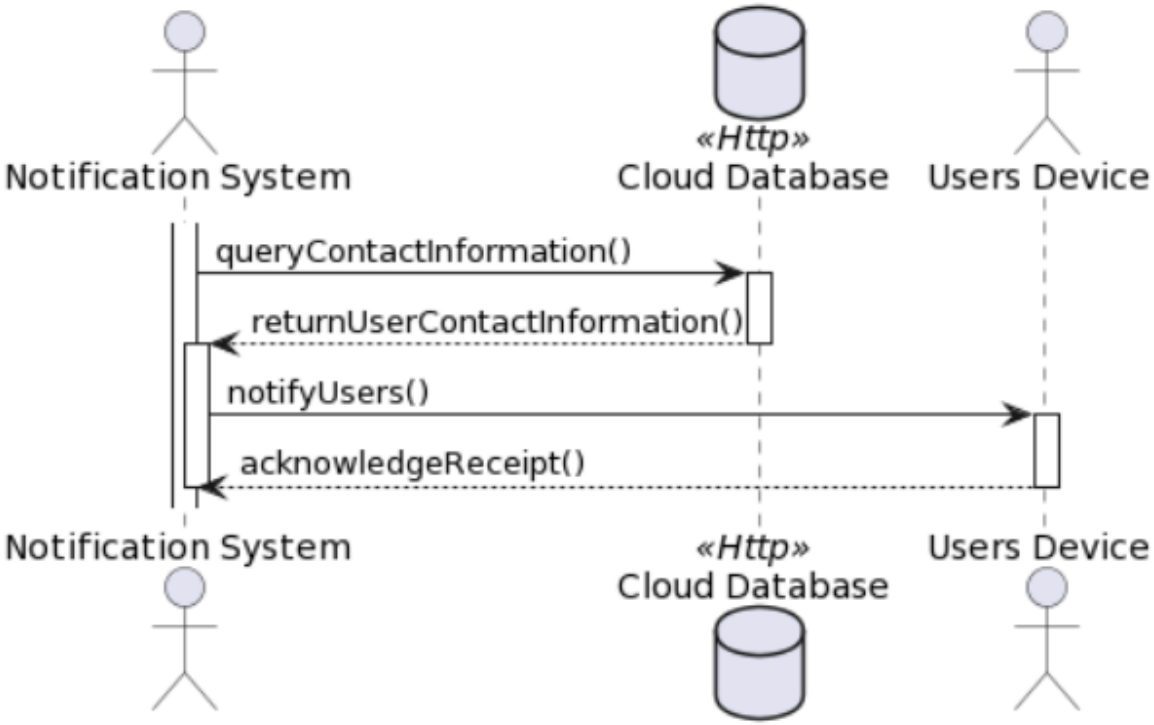


Figure 5: Sequence diagram for the emergency response use case.

2.4 Database Table Schema

The FANS project will employ a Firebase Real-Time Database to store and manage data crucial for its operation. Given the real-time nature of the FANS system, the choice of Firebase facilitates immediate updates and retrieval of data, which is vital for emergency detection and notification. The data structure is designed to ensure efficient data storage, retrieval, and management while supporting the scalability and real-time data processing requirements of the system.

Figure 6: A visual of the FANS database design schema.

This schema optimizes the FANS system's performance by enabling efficient data storage, quick access to historical data, and seamless integration between the system's components. The design also supports scalability, allowing for easy addition of new sensors and users without significant alterations to the underlying database structure.

3 Software Design

Each node in the FANS system has its software design outlined below. Simple system nodes with an algorithmic design have their control flow described using state machine diagrams as an aid. Class diagrams are included for nodes that use an object-oriented approach for their design.

3.1 Sensor Data Collection System

The sensor collection system will be programmed using the Python programming language for simplicity and its large collection of libraries.

The structure of the program will be that of two continuous polling loops, running as separate processes to utilize the CPU fully and get around the Global Interpreter Lock (GIL) of Python that prevents maximizing traditional thread performance.

The first polling loop will be responsible for collecting sensor data continuously and placing it on a queue for the other loop. It will collect data from all sensors, and that is its sole responsibility.

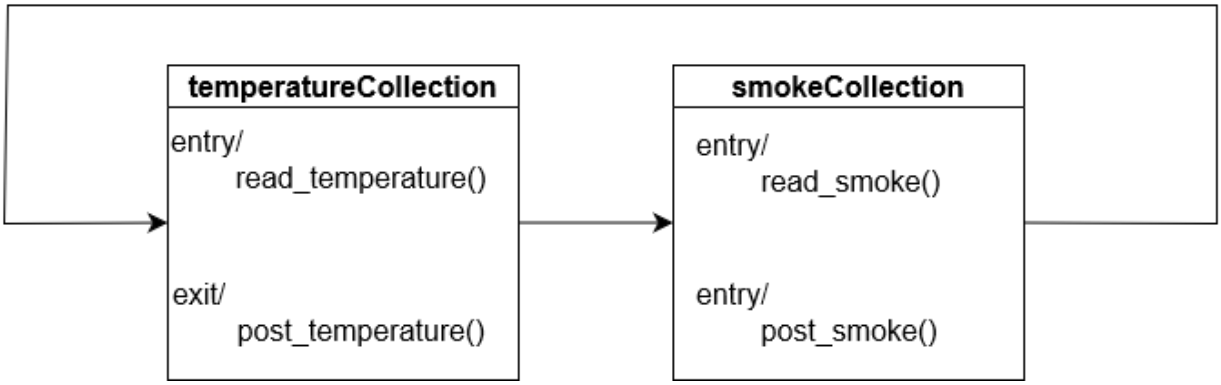


Figure 7: The polling loop for sensor data collection.

The second polling loop will be responsible for reading the sensor data from the shared queue and writing it to the Firebase database. It will also be responsible for performing logical checks on this data to determine whether or not there is an active fire emergency. It will update the emergency flag in the database whenever a sensor data measurement is above the specified threshold, and also communicate the emergency over UDP to the notification system and alarm system. Once all sensor data has been posted to the database, this loop will check for configuration updates in the sensor data thresholds from the Firebase database and apply them to the next iteration.

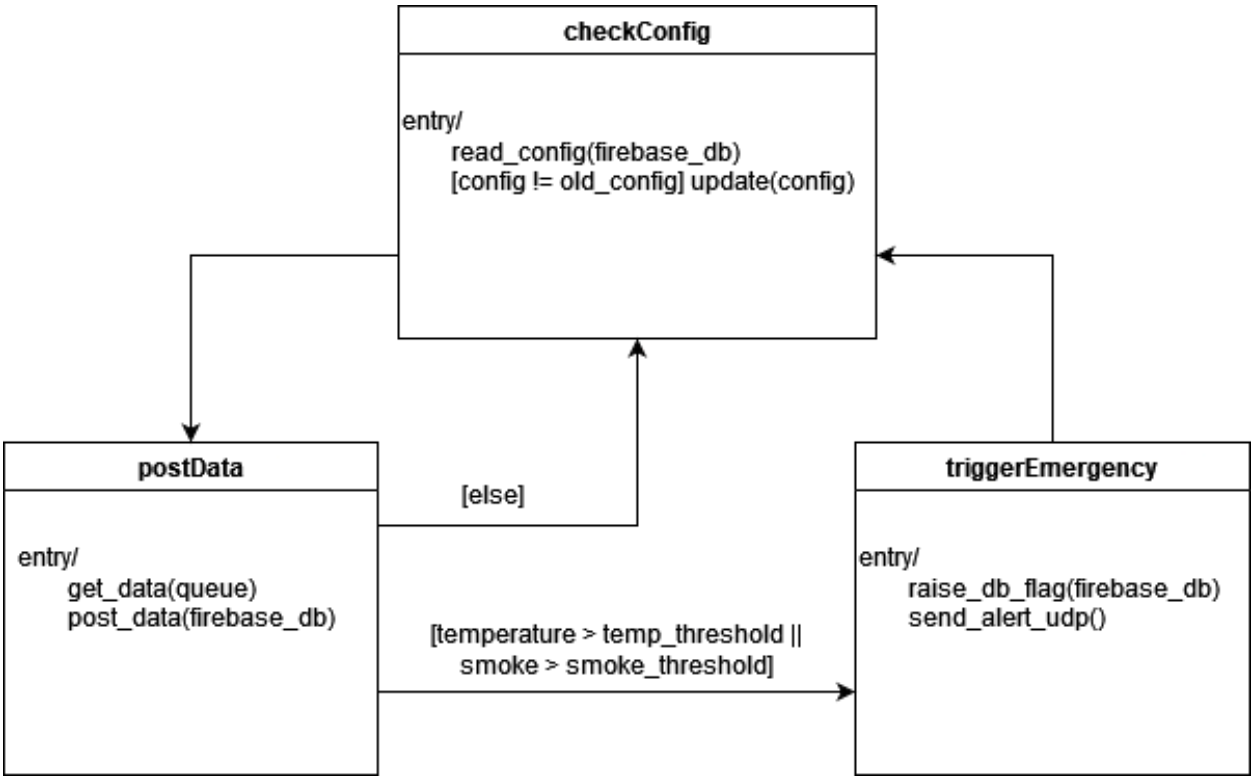


Figure 8: State machine diagram for the second process of the sensor data collection system.

3.2 Alarm System

The alarm system will be programmed using the Python programming language, again for its simplicity. The alarm system will be composed of one continuous loop, which waits to receive incoming UDP messages. When a UDP message signifying an emergency is received, the alarm system will trigger both an alarm buzzer and flashing LED lights for an infinite duration. The alarm response will be interrupted only when the system receives another UDP message signifying that the emergency has ended. This behaviour is similar to a state machine, so the software will be created using the state design pattern.

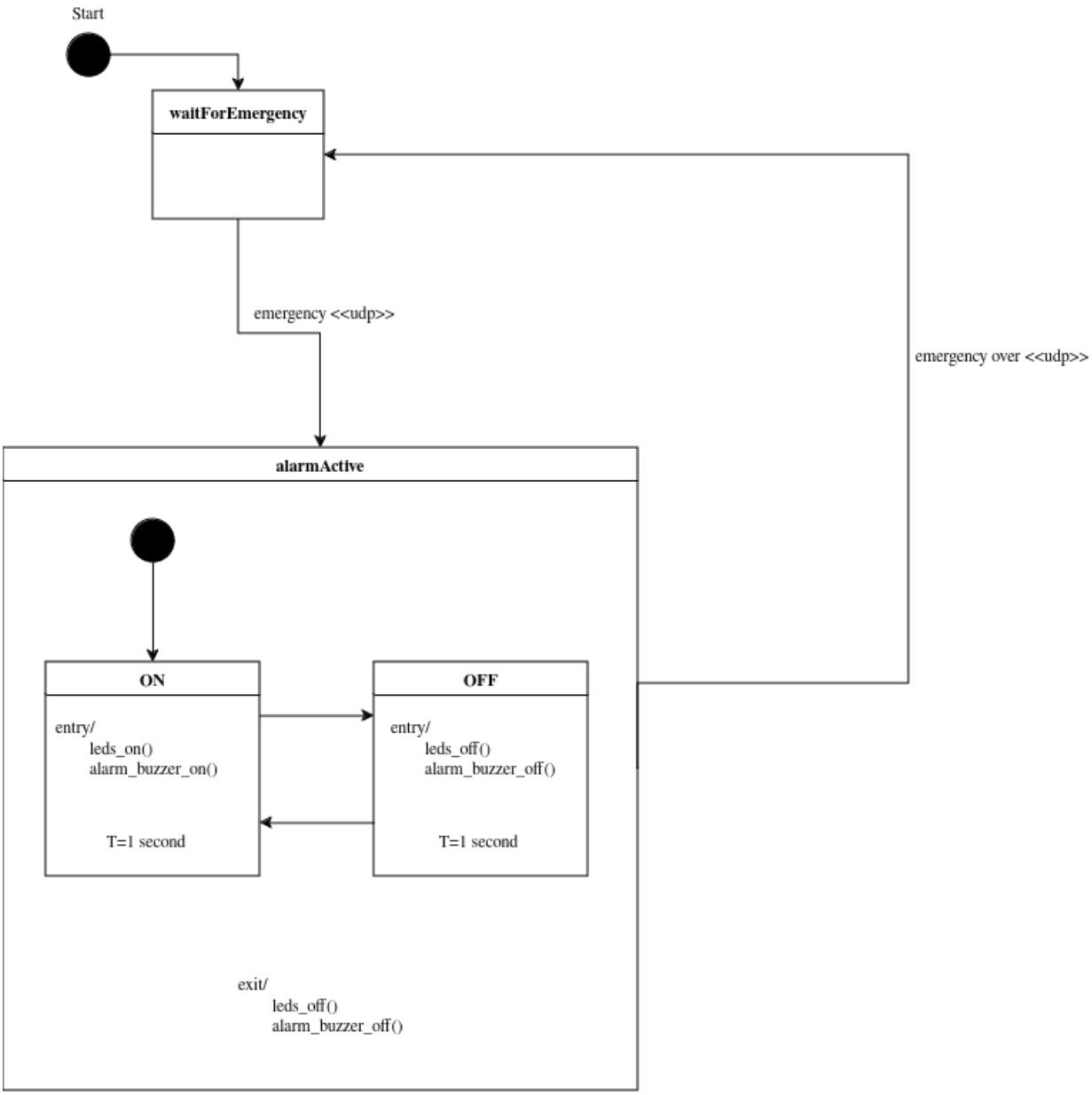


Figure 9: State machine diagram for the alarm system node.

3.3 Notification System

The notification system will be written in the Python programming language, also due to its simplicity. The notification system will listen for a UDP message signifying an emergency, and then send out email and SMS notifications to all users in the Firebase database. Once a UDP message signifying the end of the emergency is received, the system will send a followup email to all users that the emergency has been resolved.

The notification system will keep a local cache of user contact information as a backup for failing internet connectivity. It will periodically update its local cache with any changes in the upstream

Firebase database.

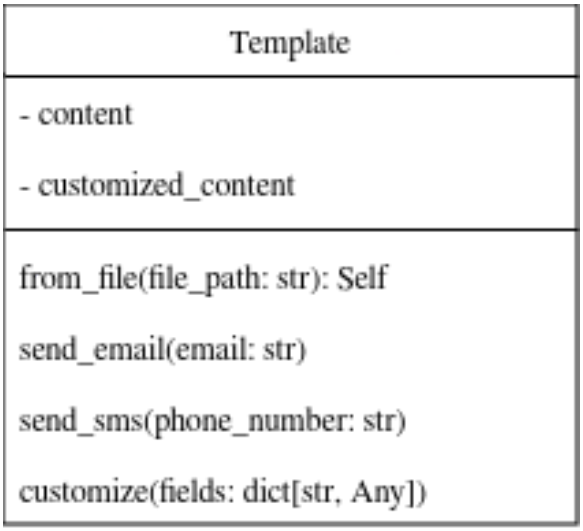
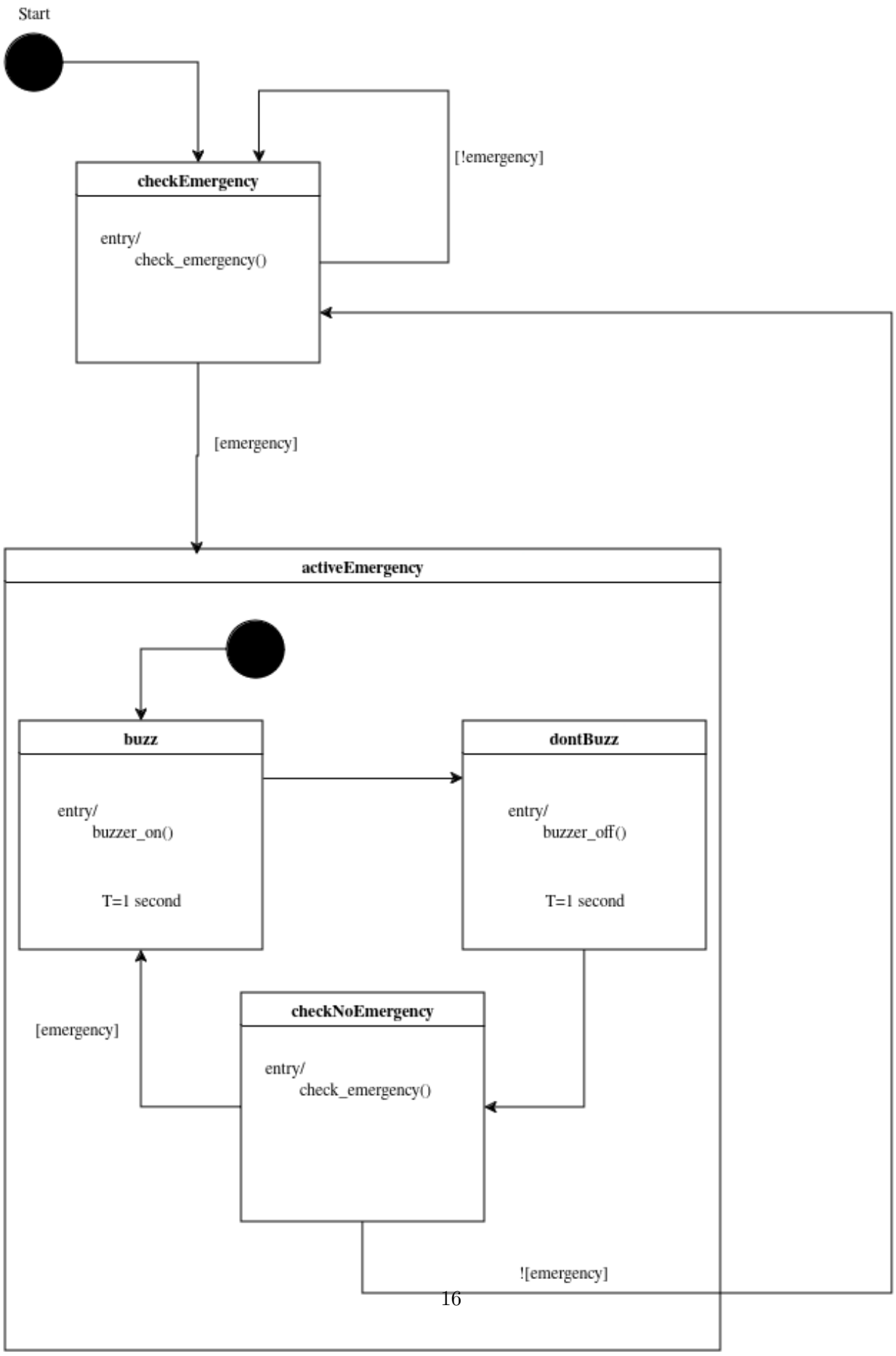


Figure 10: Class diagram for the notification system node.

The notification system will also have locally stored email and SMS templates for notifications. These will be loaded as "Templates", which provide an interface for easy customization and sending of notifications.

3.4 Haptic Alarm System

The haptic alarm wearable also has a very simple software design, and will be written in the Python programming language. The program will continuously poll the Firebase database for changes in the emergency flag. Once the emergency flag has been raised, the haptic alarm will begin to buzz on and off. During this time, it will continue checking for changes in the emergency flag on the Firebase database. Once the emergency flag is lowered, the haptic alarm will stop buzzing, signifying the end of the emergency.


Start
checkEmergency
entry/
check_emergency()
[!emergency]
[emergency]
activeEmergency
buzz
entry/
buzzer_on()
T=1 second
dontBuzz
entry/
buzzer_off()
T=1 second
checkNoEmergency
entry/
check_emergency()
[emergency]
![emergency]

4 Hardware Design

4.1 MQ2 Smoke Sensor

4.2 LCD Display

4.3 AudioHat Module

4.4 Piezoelectric Buzzer

5 GUI Design

5.1 Table of Users/Roles

6 Test Plans

Testing is an important part of any system's design. It ensures that project iterations follow the specification, behave appropriately and are less likely to include bugs.

6.1 End-to-end Communication Demo

The end-to-end communication demo should demonstrate all of the major forms of communication present in the FANS system. In order to achieve this level of demonstration, the following communication examples will be showcased:

- Sensor data upload to Firebase over internet.
- Email notifications to affected users over internet (email protocol).
- Haptic alarm buzzing in accordance with emergency flag in Firebase over internet.
- Display of sensor data on the user interface over internet.
- Emergency signal from sensor data collection system to the alarm system on local network.

It should be noted that the test for the sensor data upload over internet also demonstrates I2C communication between the Pi 4 and the SenseHat board.

First is the uploading of sensor data to the Firebase database over the internet. For this communication sequence, the sensor data collection system will be run on one of the Raspberry Pis. It will read temperature data from the SenseHat and post this data to the Firebase database using Pyrebase. The demonstration can be verified by watching the Firebase database console, since the data will be updated in real-time.

Second will be the email notifications to affected users over the internet using email protocol. To achieve this demonstration, the email notification system will be run on one of the Raspberry Pis. Using Pytest, the program will first send an email from the FANS email back to itself. Then, the program will use `imaplib` to login to the FANS email account and verify that the latest message in its inbox contains the same email message that was sent in the test. If the messages match, Pytest will report a success. This test is fully automated.

Next is the haptic alarm buzzing. For this test, the haptic alarm program will be run on one of the Raspberry Pi's to emulate the Pi Pico (as it has not arrived at the time of writing). When the program is run, it will poll the Firebase database using `pyrebase` to check if the emergency flag has been raised. The flag can either be raised by hand (the tester modifies the flag using the Firebase console) or it can be set by running the sensor data collection system with a low emergency reporting threshold for temperature. In either case, when the flag is raised, the output of the haptic alarm can be verified by observing the console messages. Console printing will be used in place of the buzzer actuator while we wait for components. A console message of "buzz" would indicate the buzzer buzzing, and no console messages indicate no emergency while the alarm is polling the database.

To test the display of sensor data collection on the user interface, the GUI will be run on one of the Raspberry Pis using Flask. When a user clicks the "refresh" button, the Flask API endpoint responsible for returning temperature data will be hit. The endpoint will use `pyrebase` to request the latest temperature data from Firebase and then will respond with JSON. The GUI's JavaScript

logic will simply print this JSON as text content of an HTML paragraph tag to show that the connection is present.

Finally, to demonstrate UDP communication over the local area network, both the sensor data collection system and the alarm system will be run on two separate Pis. Both programs will first upload their public IP to the Firebase database under a 'devices' table, and then read the other device's IP. The sensor data collection system should have a sufficiently low emergency threshold to trigger an emergency at room temperature (only for testing purposes). Once the sensor data collection system detects an emergency, it will send a UDP message containing the numerical data '0' (the agreed upon encoding for signifying an emergency) over the LAN addressed to the alarm system. The alarm system will receive the emergency notification, and instead of activating an actuator it will print "Received 0" to the console. This can be manually verified.

6.2 Unit Test Demo

The purpose of the unit test demo is to demonstrate that the individual components of our system behave as expected and do not contain logical or physical errors. This will sanity check our hardware components and our software logic.

6.2.1 Hardware

The hardware components that will need to be tested are:

- Smoke sensor
- Temperature sensor
- Piezoelectric buzzer
- LCD screen

Smoke Detector

The smoke detector will be difficult to test due to the nature of what it measures. It will not be possible to light a fire within the school or near campus because that will set of existing fire suppression systems and may be dangerous. Testing the smoke detector in smokey environments can be done by:

- Running the data collection from the smoke detector in a smoking zone on campus.
- Running the data collection from the smoke detector off campus and lighting a fire nearby.

For both of these tests, it should be verified that smoke levels are detected. This will be a one-time sanity test of the sensor. The smoke detector can also be tested in smokeless environments by collecting data from the sensor indoors on campus and ensuring that no smoke is detected.

Temperature Sensor

To test the temperature sensor, we can collect data from the sensor in the lab setting and ensure that measurements are within the acceptable range for room temperature.

Piezoelectric Buzzer

To test the piezoelectric buzzer, we can test turning on and off the buzzer and verifying that it produces noise. Additionally, we can set the buzzer to produce different frequencies and verify that

they are set correctly by checking with a instrument tuning app.

LCD Screen

Finally, to test the LCD screen, several test messages can be displayed on screen and visually inspected for correctness as a sanity check.

6.2.2 Database Integrity

To test cloud database integrity, we can perform several test and security checks.

Insecure Access

To test the security of the cloud database, we can attempt to perform read and write operations using an incorrect API key and ensure that the requests are denied.

Stress Test

To stress test the cloud database, several instances of the sensor data collection program can be run on multiple Raspberry Pis, all of which will upload sensor data to the database. The database should:

- Keep up with all of the simultaneous requests.
- Avoid any overwriting/race conditions with concurrent write operations.

6.2.3 Software

6.3 Final Demo

7 Project Update

7.1 Project Milestones

7.2 Schedule of Activities

References

[1] P. Matoušek, *Thick smoke on black background.* [Online]. Available: https://www.freeimages.com/photo/thick-smoke-on-black-background-1633270 (visited on 02/11/2024).

[2] (Oct. 7, 2022), [Online]. Available: https://www150.statcan.gc.ca/n1/pub/11-627-m/11-627-m2022035-eng.htm (visited on 02/12/2024).

[3] A. Erickson. (Sep. 15, 2023), [Online]. Available: https://www.digitize-inc.com/blog/fire-alarm-whats-new-2023.php (visited on 02/12/2024).

[4] "Smoke alarms: A sound you can live with." (Sep. 29, 2020), [Online]. Available: https://www.getprepared.gc.ca/cnt/rsrcs/sfttps/tp201011-en.aspx (visited on 02/11/2024).

[5] "Mems pressure sensor: 260-1260 hpa absolute digital output barometer." (), [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiGq4Dy4_GEAxX1HNAFHSf-BRwQFnoECBcQAQ&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fdatasheet%2Flps25hb.pdf&usg=AOvVaw2FTksZHmMhimTAl6Qq3eFF&opi=89978449 (visited on 03/13/2024).

[6] "Mcp3004/3008." (), [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MSLD/ProductDocuments/DataSheets/MCP3004-MCP3008-Data-Sheet-DS20001295.pdf (visited on 03/13/2024).