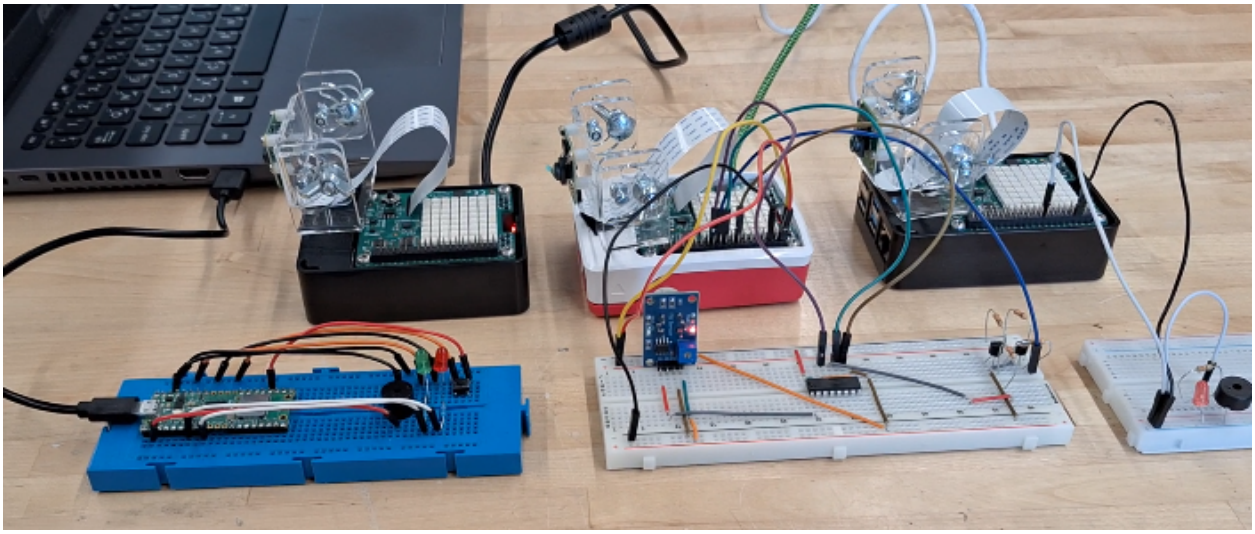
SYSC3010 Computer Systems Development Project
FANS Final Report

Group L1-G8



https://github.com/SYSC3010-W24/sysc3010-project-l1-g8

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

Created: April 3rd, 2024
Modified: April 9, 2024

Contents

1 Project Description

1.1 Motivation

The motivation for the FANS project was to address the critical problem of preventable fire-related deaths in Canada. By leveraging modern technological advancements, the FANS system aimed to significantly enhance the effectiveness of fire alarm systems, reducing response times, and ultimately saving lives. The importance of this project could not be overstated, as it directly impacts the well-being of Canadians.

1.2 Problem Statement

The Fire Alarm Notification System (FANS) was created to address the alarming number of preventable fire deaths in Canada. There are 220 annual fire-related casualties, and at least one in seven of these deaths occurs in a home without working smoke alarms [1]. This emphasizes the shortcomings of traditional fire alarm systems, which often lacked advanced communication capabilities and robust monitoring capabilities [2].

The FANS project addresses these issues by integrating smoke and temperature sensors with Internet of Things (IoT) technology. This approach not only covers 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.

With configurable thresholds for smoke and temperature alarms and adjustable timeouts that prevent the system from being unsafely disabled, FANS uses technological advances to significantly improve the effectiveness of fire detection systems. The ultimate goal of the project is to improve overall fire safety and emergency response, and thereby mitigate fire disasters and protect the well-being of individuals, families, and communities across Canada [3].

1.3 Overview of Solution

The FANS system is comprised of several key components in a distributed system for greater efficacy of emergency detection. The system's architecture is outlined in section 2.

Our smoke detection system continuously monitors the environment for smoke and temperature changes using different digital and analog sensors. It serves as the detection mechanism for emergencies, and is responsible for activating the system's emergency response when abnormal measurements are detected. Emergency activation is done both over local network communication to other nodes, and over the internet to nodes that may be connected to different networks. This creates a redundancy for inter-node communication, which can be critical in case of internet connection loss during an emergency.

The alarm system triggers loud audible and visual alerts in the event of a fire, ensuring occupants are immediately warned in emergency situations. It can be stationed in high traffic areas for maximum efficacy.

The FANS notification system immediately sends out emergency alerts over email and SMS to a predefined contact list of users, ensuring rapid response to the detected fire. The system maintains a local database copy of the cloud database to ensure that these critical messages are still sent if connection to the cloud service is lost.

Finally, FANS utilized a Pi Pico wearable device to provide immediate feedback to individual users, offering an additional layer of alert through a loud wearable alarm and blinking LEDs for visuals. This is best suited to individuals with hearing impairments or who may not always be within a close distance to the alarm system nodes.

The entire system can be controlled from a web GUI, where a system administrator can subscribe more contacts to emergency notifications, modify detection thresholds, timeout the system and watch real-time visualizations of smoke and temperature measurements being taken.

Our design enables easy expansion with additional sensor or alarm units as needed. Each component functions independently and has multiple redundancies for inter-node communication, ensuring reliability and ease of maintenance.

2 Final Solution

The final solution for FANS is displayed in Figure 1. This solution uses a distributed systems approach to create a multi-node fire response system controlled via a web interface.

2.1 Deployment Diagram

The primary node in our FANS project is the sensor data collection node, which monitors temperature and smoke concentration levels. All sensor measurements are written to the Firebase cloud database for display by the web UI.

When a measurement is detected above the user-defined threshold, the sensor data collection node sends a UDP message signifying an emergency over the local network to the notification system and alarm system nodes. Upon receipt of this message, the notification system will send emails and SMS notifications to all users in its local database. The alarm system node will begin blinking an LED and sounding an alarm.

In addition to the UDP message, the sensor data collection node will raise an emergency flag in the Firebase database which signifies to the UI and the wearable haptic alarm that an emergency is active. The haptic alarm wearable device will then begin blinking an LED and buzzing until the user disables it or the emergency ends.

The UI provides an interface for checking on the status of an emergency and reviewing sensor data in real-time. Users can disable the emergency response from the UI when the emergency has been resolved, and they can also time out the system's emergency response whenever the system is falsely triggered (smokey cooking). Finally, the UI also allows user contact information to be added to the subscriber list for emergency notifications.

2.2 Message Protocol Table

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.

The system's core, the Smoke Detection System, communicates with its temperature and smoke sensors using the I2C and SPI protocols over the GPIO pins of a Raspberry Pi 4.

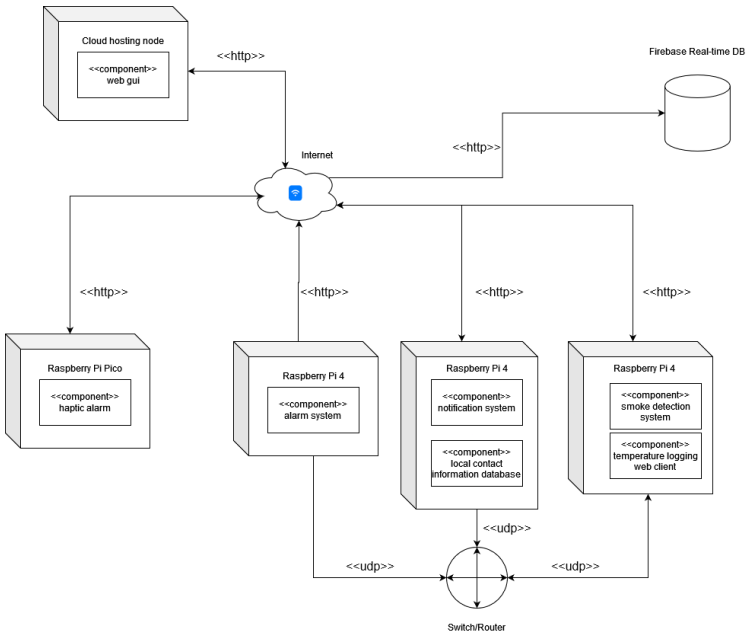


Figure 1: The FANS deployment diagram.

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

Table 1: Messages for I2C communication in FANS.

2.2.1 I2C & SPI Communication

The Raspberry Pi 4 utilizes the I2C and SPI protocol to communicate with temperature and smoke sensors, monitoring environmental conditions to detect potential fire hazards. These messages are outlined in 1.

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 shown in Table 2. The messages sent are in Table 3

2.2.3 Cloud Database Communication

Messages between each node and the Firebase database are shown in Table 4.

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.

Sender	Receiver	Message	Data Format	Protocol
Smoke Detection	Cloud DB	`put_sensor_data()`	See listing 1	HTTP (JSON)
GUI	Cloud DB	`update_threshold(new_threshold)`	See listing 2	HTTP (JSON)
Notification	Cloud DB	`query_contact_information()`	See listing 3	HTTP (JSON)
Haptic Alarm	Cloud DB	`emergency()`	See listing 4	HTTP (JSON)
GUI	Cloud DB	`get_sensor_data()`	See listing 5	HTTP (JSON)

Table 4: Messages for cloud database communication in FANS.

Listing 1: Update sensor data message.

```
{
  "method": "PUT",
  "path": "/sensor-data/temperature",
  "headers": {
    "Authorization": "Bearer YOUR_ACCESS_TOKEN",
    "Content-Type": "application/json"
  },
  "body": {
      "data": {
          "temperature": 21.2,
          "timestamp": "2024-03-13T08:37:22"
      }
  }
}
```

Listing 2: Threshold update message.

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

Listing 3: Request for user contact information.

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

Listing 4: Request for emergency flag.

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

Listing 5: Request for latest sensor data.

```
{
  "method": "GET",
  "path": "/sensor-data",
  "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 6	SMTP (Email)

Table 5: Messages for user notification communication in FANS.

Listing 6: Email notification for detected emergency in FANS.

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

Dear [User's Name],

This is an emergency notification. Please exit the building.

Emergency detected: [Date, Time]

Stay safe!
```

2.3 Sequence Diagrams

The FANS system satisfies several core use cases. The three most critical use cases are shown below, and more detail can be found in Appendix B.

2.3.1 Trigger Emergency Use Case

The trigger emergency use case shown in Figure 2 describes the critical functionality of FANS, where the system detects a potential fire and initiates a series of automated responses to mitigate the situation. As depicted in Figure 2, the process begins when the smoke detection system identifies smoke and potentially high temperatures indicative of a fire. This detection triggers a UDP notification to the alarm and notification systems to alert them to begin their automated emergency responses. It additionally raises a flag in the Firebase cloud database for non-local nodes (such as the haptic alarm) to begin their responses.

2.3.2 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 personal and environmental conditions. This use case, shown in Figure 3, involves a user accessing the web GUI to modify the threshold settings. Once the user submits their new thresholds, the web GUI sends an update request to the database and displays a confirmation to the user. The smoke detector unit periodically checks the thresholds in the database for updates, and updates its local values when it detects a change.

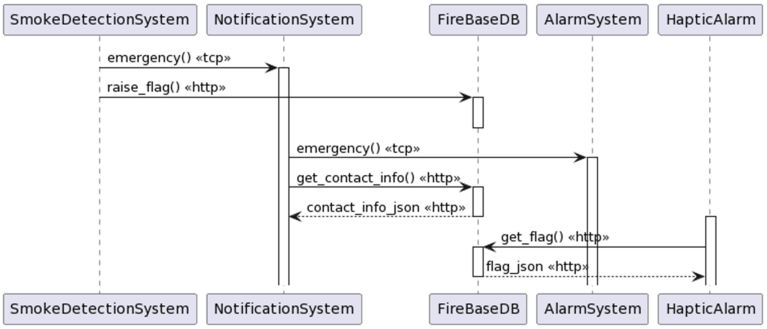


Figure 2: The sequence diagram for the trigger emergency use case.

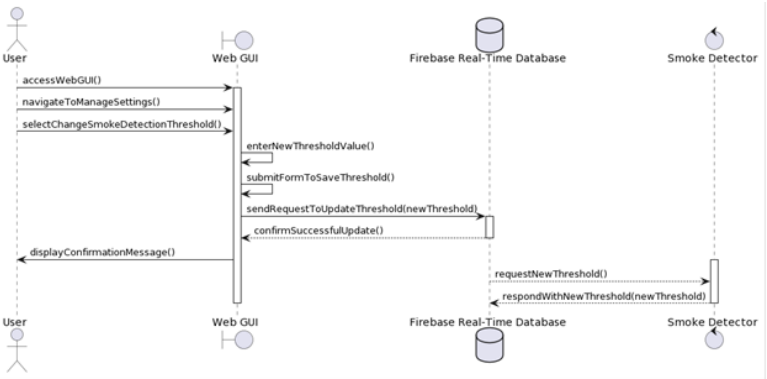


Figure 3: The sequence diagram for the changing emergency threshold use case.

2.3.3 Emergency Notification Use Case

The scenario in Figure 4 highlights FANS capability to notify users in the event of a detected fire. Upon being notified of an emergency, the notification system queries the cloud database for contact information and sends out alerts via SMS and email to all users. If the cloud database is not accessible, the local database is instead used.

3 Discussion of Final Design

Throughout the development of FANS, our team adhered to the specifications and objectives outlined in our initial proposal and detailed design document. Our goal was to develop a system that would not only detect fire based on smoke and temperature changes but also immediately notify people of emergencies.

1. Complete implementation: We are proud that our project was realized completely and without omissions. Every component, from the smoke detection system to the emergency call system, was implemented as planned. Thanks to our commitment to rigorous planning and effective teamwork, we were able to overcome challenges and ensure that all project objectives were met.

2. Consistency of protocols: There was no deviation from the originally proposed communication protocols. The use of the I2C protocol for sensor communication, UDP for local network communication and HTTP requests for interaction with the Firebase real-time database were

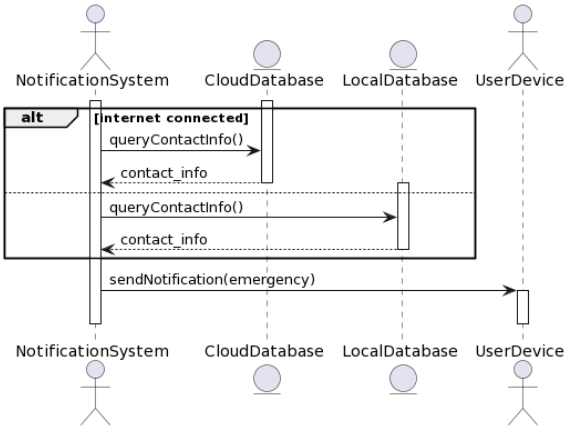


Figure 4: The sequence diagram for the emergency notification use case.

maintained throughout the project.

Our project not only met the expected design specifications, but also performed excellently in the test phases. Here are some important results and data from our tests:

- Sensitivity of the sensors: the smoke and temperature sensors used in the FANS showed high sensitivity and accuracy during testing. They were able to detect slight changes in environmental conditions, ensuring early detection of potential fire hazards.
- Efficiency of the notification system: The notification system was tested to ensure that it could immediately trigger an alarm when a fire was detected. The test results showed that the SMS and email notifications were sent without significant delay, usually within seconds of the emergency being detected.
- Scalability of the system: Tests confirmed that adding new components did not affect the performance of the system, proving the flexibility and adaptability of the design to different environments.
- GUI functionality: FANS UI provides an interactive platform for real-time monitoring and control, enabling direct alarm management and seamless integration with the system components and real-time database.

The fire detection system represents a significant advance in fire protection, effectively meeting the need for an integrated, responsive and reliable fire detection and notification system. Through unwavering commitment to our project goals and meticulous adherence to our design, we have developed a system that will significantly impact the well-being of Canadians and provide a new level of protection from fire hazards.

4 Contributions

This section lists all of the contributions of each member on the FANS development team.

4.1 Code Contributions

Code file contributions listed in this table will use Unix file paths to describe which files were worked on by each contributor. The asterisk signifies the wildcard operator, so `alarm-system/*` captures all files within the alarm-system/ directory. All paths correspond to paths within the project repository on GitHub.

For a more granular view of contributions, where multiple authors have worked on the same file, please review the Git commit history.

Author	Code files
Grant Achuzia	• `docs/*` • `web-app/*` • `pico_alarm/*` • `README.md`
Saja Fawagreh	• `docs/*` • `notifier/*` • `notifier/tests/test_email_notification_system.py` • `README.md`
Javeria Sohail	• `docs/*` • `pico_alarm/*` • `haptic_alarm_test/*` • `README.md`
Matteo Golin	• `alarm-system/*` • `sensor-pi/*` • `web-app/main.py` • `web-app/templates/index.html` • `web-app/templates/settings.html` • `docs/*` • `notifier/templates/*` • `notifier/tests/test_email.py` • `notifier/messages.py` • `README.md`

Table 6: The individual code contributions of each FANS team member.

4.2 Report Contributions

Author	Report sections
Grant Achuzia	Project Success, Learning Experiences, Appendix A
Saja Fawagreh	Message Protocols, Sequence Diagrams, Discussion of Final Design
Javeria Sohail	Motivation, Problem Statement, Final Design Solution
Matteo Golin	Contributions, Deployment Diagram, Project Extensions

Table 7: The individual final report contributions of each FANS team member.

5 Reflections

5.1 Project Success

Overall, the team considers our project, Fire Alarm Notification System (FANS), a success. We were able to develop a comprehensive real-time system that addresses the issues that arise with preventable fire-related deaths in Canada. FANS uses modern web technologies (Flask) and communication protocols (I2C, UDP, and HTTP) to enhance its effectiveness.

FANS functions as we intended: detecting smoke and high temperatures, triggering multiple alarms, and sending prompt notifications to registered users. The system's ability to detect potential fires and alert users in real-time is a major part of the project's success.

FANS sensors were accurate and reliable, ensuring possible fire hazards are detected rapidly. The system can easily be expanded by adding more alarm units and accessory devices that run similar code to FANS's current implementation. Our GUI provides an intuitive monitoring and control interface for users, allowing one to check the emergency status, review sensor data, and manage alarms.

The FANS project can be considered a success due to its functionality, reliability, and scalability.

5.2 Learning Experiences

Developing FANS gave our team valuable experience and insight into creating a multi-faceted product.

One of the most significant learning experiences was gaining hands-on experience with the Raspberry Pi Ecosystem. Getting using the Raspberry Pi's terminal, learning micropython for the Pico W, and applying our Electronics knowledge to build circuits, were all essential to the growth of our technical ability.

Our time-management skills were strengthened as we learned to juggle report writing and coding, while doing other courses. Communication amongst the team was critical, and we learned to balance our varying skill sets with different aspects of the project. We resolved conflicts (scheduling and opinion-based) constructively while working together to meet all our milestones.

5.3 Project Extensions

The first project extension that could be made is the inclusion of geographically aware system nodes. FANS could make system nodes geographically aware so that the emergency response would only target affected users. In addition, haptic alarm buzzers should only buzz if the wearer is within a certain range of the affected area. Only users who would be within the affected area should be notified via email and SMS to steer clear of the area. Using geographic location information, FANS could be modified to subscribe users to emergency response measures only if they are within the affected area.

References

[1] (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).

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

[3] "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).

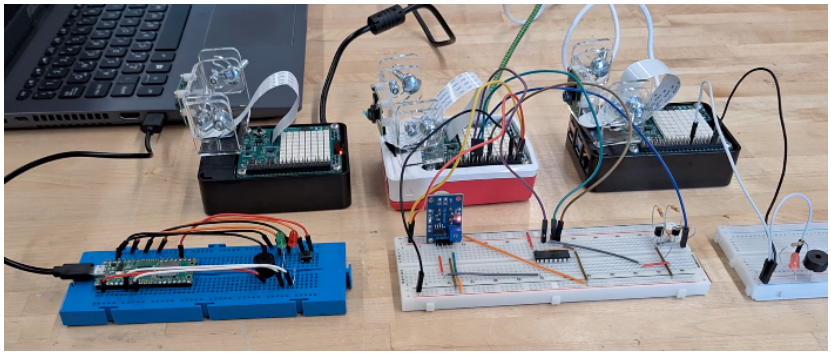
[4] "Mems pressure sensor: 260-1260 hpa absolute digital output barometer." (), [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiGq4Dy4_GEAxX1HNAFHSf-BRwQFnoECBcQAQ&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fdatasheet%2Flps25hb.pdf&usg=AOvVaw2FTksZHmMhimTAl6Qq3eFF&opi=89978449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

[5] "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).

A GitHub Repository README

The following content is the README file taken from the FANS GitHub repository.

FANS (Fire Alarm Notification System)



FANS is a comprehensive multi-system solution for fire-related emergencies that require prompt notification of persons in the affected area.

FANS utilizes smoke and temperature sensors to accurately detect fires in its surrounding environment. Upon detection, FANS issues SMS and email notifications to individuals in the surrounding area.

FANS provides a dashboard for live monitoring of its environment, as well as an interface for customizing things like its sensitivity to pressure changes.

Table Of Contents

Contributors

SYSC3010 group L1-G8, under guidance of TA Sean Kirkby.

- Grant Achuzia
- Javeria Sohail
- Matteo Golin
- Saja Fawagreh

Repository Organization

FANS is composed of several different nodes. Within the project repository, each node is given its own directory containing its required files.

- The data collection system is stored in `sensor-pi/`
- The notification system is stored in `notifier/`
- The web GUI is stored in `web-app/`
- The alarm system is stored in `alarm-system/`
- The haptic alarm system is stored in `pico_alarm/`

In addition, the `docs/` directory contains documentation on our system design. Within `docs/`, the `assets/` folder contains all of the UML class, sequence, state machine and deployment diagrams which describe the FANS system and which are used throughout our reports. The `design-report/` folder contains all of the LaTeX source files that compose the FANS design report, and the `proposal/` folder contains the LaTeX source files for building our initial project proposal.

Installation and Setup Guide

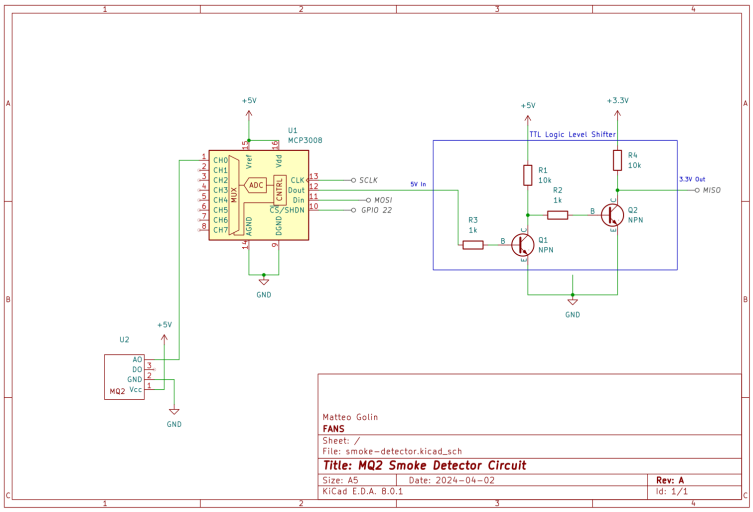
Guides to install and run each system node are provided below. Once all nodes have been set up, the system will be functional.

Sensor-Pi

In order to set up the `sensor-pi` sensor data collection system, you will need:

- Raspberry Pi 4
- Sense hat
- Breadboard
- Wires
- 2x 10k resistors
- 2x 1k resistors
- 2x NPN bipolar-junction transistors
- Flying fish MQ2 smoke detector module

In order to assemble the circuit, place the components on your breadboard according to the following schematic. Power sources can come from the Raspberry Pi or an external source. The labelled pin outputs (GPIO 22, SCLK, MOSI, MISO) are all meant to be connected to the corresponding pin on the Raspberry Pi 4. Please see its pinout sheet to locate this pins.



Ensure that the following files are present in the same directory for it to function properly:

- `firebase_config.json`: This file should include the configuration details required to connect to the Firebase database, such as the API key, authentication domain, database URL, and storage bucket.

Example `firebase_config.json` file:

```
{
  "apiKey": "yourKey",
  "authDomain": "yourDomain",
  "databaseURL": "https://your-database-url",
  "storageBucket": "someStorageBucket"
}
```

Once the circuit is built and connected, the node can be started by running the software located in the `sensor-pi` directory from the Raspberry Pi. Python 3.11 must be previously installed. The following commands will download the repository and start the software when run in the terminal:

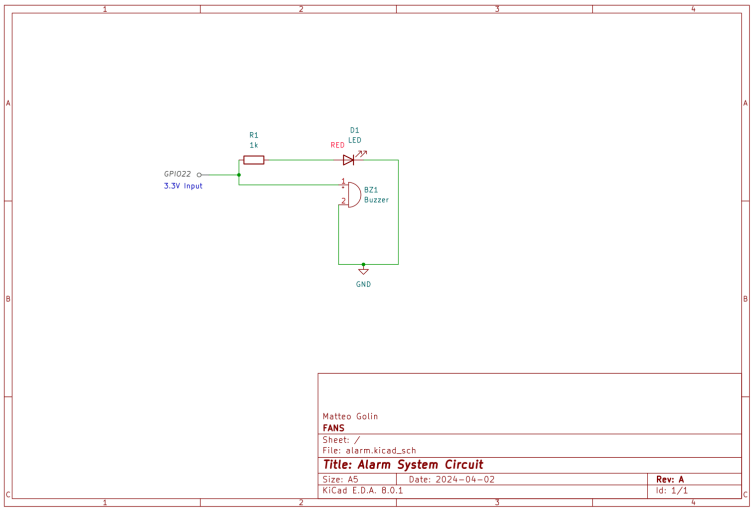
```
git clone https://github.com/SYSC3010-W24/sysc3010-project-l1-g8.git
cd sysc-project-l1-g8/sensor-pi
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python3 main.py
```

Alarm System

In order to set up the `alarm-system` node, you will need:

- Raspberry Pi 4
- Breadboard
- Wires
- Passive piezoelectric buzzer
- Red LED
- 1k resistor

In order to assemble the circuit, place the components on your breadboard according to the following schematic. The 3.3V input signal should be connected to the Raspberry Pi's GPIO 22 pin. Please see the Pi's pinout sheet to connect the pins properly.



Once the circuit is assembled and connected, the node can be started by running the software located in the `alarm-system` directory from the Raspberry Pi. Python 3.11 must be previously installed. The following commands will download the repository and start the software when run in the terminal:

```
git clone https://github.com/SYSC3010-W24/sysc3010-project-l1-g8.git
cd sysc-project-l1-g8/alarm-system
python3 -m venv venv
source venv/bin/activate
```

```
pip install -r requirements.txt
python3 main.py
```

Notifier

In order to set up the `notifier` node, you will need:

- Raspberry Pi 4

The node can be started by running the software located in the notifier directory on the Raspberry Pi. Prior installation of Python 3.11 is required. Execute the following commands in the terminal to download the repository and start the software:

```
git clone https://github.com/SYSC3010-W24/sysc3010-project-l1-g8.git
cd sysc-project-l1-g8/notifier
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python3 email_notification_system.py
```

Additionally, ensure that the following files are present in the same directory for it to function properly:

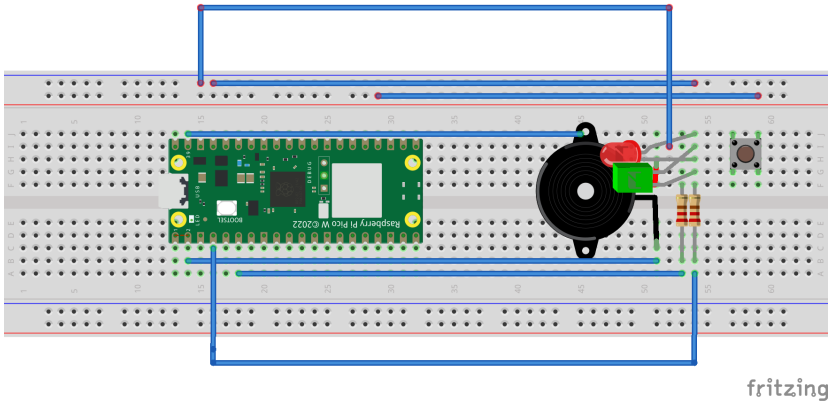
- `fans_credentials.json`: This file should contain the username and password required for authentication to the email server, enabling the system to send notifications via email.
- `firebase_config.json`: This file should include the configuration details required to connect to the Firebase database, such as the API key, authentication domain, database URL, and storage bucket.
- `twilio_credentials.json`: This file should contain the authentication credentials for accessing the Twilio API, including the account SID and authentication token, enabling the system to send notifications via SMS.

Pico-Alarm

In order to set up the `pico_alarm` buzzer alarm and led notifier system, you will need:

- Raspberry Pi Pico W
- Piezoelectric Buzzer
- 1 red and 1 green LED
- Breadboard
- Wires
- 2x 10k resistors

Hardware Assembly



1. **Connect the Buzzer** to GPIO 1 on the Pico W.
2. **Attach the Red LED** to GPIO 6 with a 10k resistor in series.
3. **Attach the Green LED** to GPIO 5 with a 10k resistor in series.
4. **Set up the Button** on GPIO 18, ensuring it's properly debounced with a 10k resistor.
5. **Wire everything** according to the schematic on a breadboard, ensuring secure connections.

Software Installation and Deployment for Pico_Alarm System

1. **Connect the Raspberry Pi Pico W** to your computer. Use a USB cable to establish the connection. Ensure the Pico is in MicroPython mode.
2. **Open Thonny IDE** on your computer. It's recommended to use Thonny for working with MicroPython on Raspberry Pi Pico due to its built-in support.
3. **Prepare the Python Script**: Have the `pico_alarm.py` script ready. This script includes the logic for monitoring the Firebase database and activating the buzzer and LEDs.
4. **Configure the Script**:
   - **WiFi Credentials**: Within the script, locate the `do_connect()` function. Replace `"your_wifi_ssid"` and `"your_wifi_password"` with your actual WiFi network SSID and password.
   - **Firebase URL**: Find the `firebase_url` variable in the script. Change its value to your Firebase database URL where the emergency flag will be checked.
5. **Deploy the Script to Pico W**:
   - In Thonny, select MicroPython (Raspberry Pi Pico) as the interpreter from the bottom right corner.
   - Open the `pico_alarm.py` script in Thonny.
   - Click on 'File' > 'Save As...' and choose to save the script on your Raspberry Pi Pico.
6. **Run the Script**:
   - With the `pico_alarm.py` script open in Thonny, press the green 'Run' button to execute the script on the Pico W.
   - The script will automatically start monitoring the Firebase database for any emergency flags and activate the buzzer and LEDs accordingly.

Note: The Raspberry Pi Pico W must be powered continuously for the alarm system to function. It can remain powered via the USB connection to your computer or through an external 5V power source.

Operational Notes

- The system checks the Firebase database at regular intervals for any emergency flags.
- The buzzer and red LED activate to indicate an emergency, with the green LED indicating normal operations.
- The button serves as an acknowledgment mechanism to stop the alarm and reset the system to its normal state.

Web App

In order to set up the `web-app` node, you will need:

- Access to the internet

FANS web application is made using Flask, an easy to use Python micro framework. The `static/` folder contains css files for styling the app as well as a JS script for dynamically changing between themes. The `templates/` folder contains html files, and the `tests/` folder contains tests for making sure the application gets data from firebase.

Ensure that the following files are present in the same directory for it to function properly:

- firebase_config.json: This file should include the configuration details required to connect to the Firebase database, such as the API key, authentication domain, database URL, and storage bucket. Example firebase_config.json file:

```
{
  "apiKey": "yourKey",
  "authDomain": "yourDomain",
  "databaseURL": "https://your-database-url",
  "storageBucket": "someStorageBucket"
}
```

To gain access to the website, run the following commands in your terminal:

```
git clone https://github.com/SYSC3010-W24/sysc3010-project-l1-g8.git
cd sysc-project-l1-g8/web-app
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python3 main.py
```

In the terminal, click on the second http link to access the web app on a network server. Here's an example of what the second link may look like:
`Running on http://192.168.X.X:5000/`

B Additional Use Cases

The following sub-sections describe additional use cases that are handled by FANS, along with their sequence diagrams.

B.0.1 Add New Contact Information Use Case

The use case shown in Figure 5 outlines the process of adding new contact information to the FANS database. Through the web GUI, a user enters a new contact (name, email, phone, etc.) to be

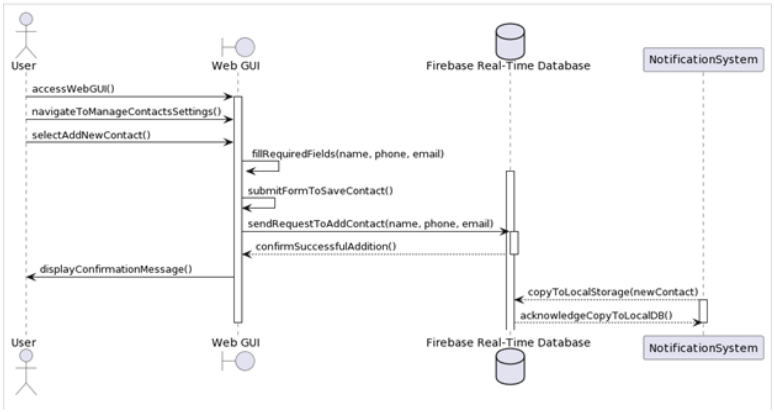


Figure 5: The sequence diagram for the adding new contact information use case.

subscribed to emergency notifications. Upon submission, these details are updated in the real-time cloud database. The notification system also copies these changes to its local database a backup for possible connection loss.