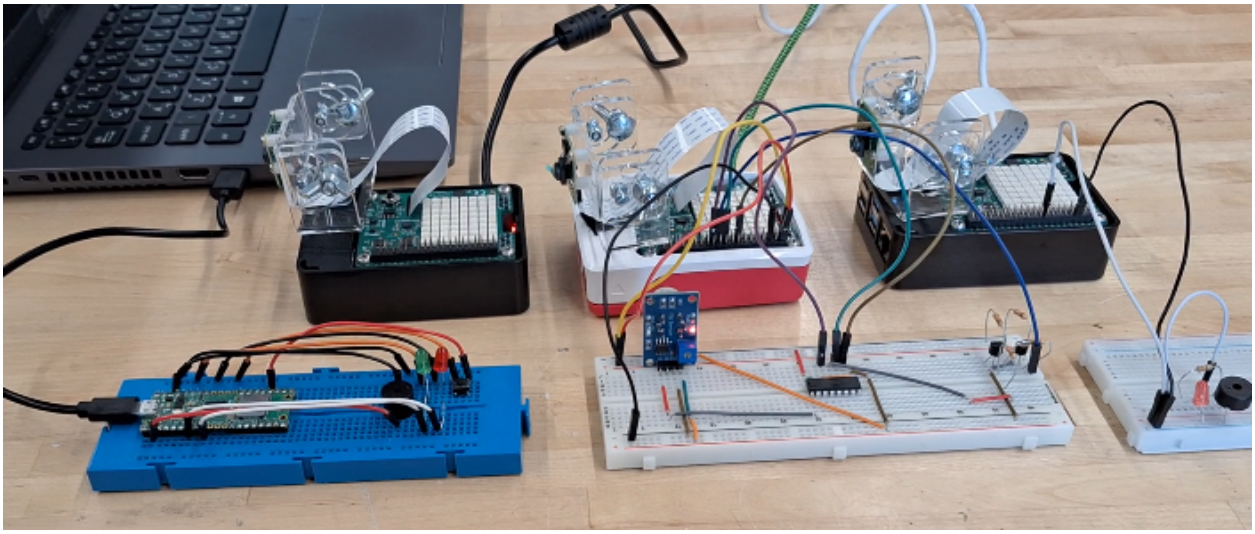
SYSC3010 Computer Systems Development Project
FANS Final Report

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 an emergency is detected, the sensor unit sends an alert over UDP and raises an emergency flag in the cloud database.

The notification system and alarm system are subscribed to UDP messages sent by the sensor data collection node. This allows them to be notified of detected emergencies even when they cannot read the emergency flag in the cloud database.

The haptic alarm and web UI are connect to the system through the cloud database. The haptic alarm activates its emergency response when it sees a change in the emergency flag. 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. All changes to settings from the UI are propagated through the cloud database to the sensor unit and notification system over the internet.

2.2 Message Protocol Table

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.

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.

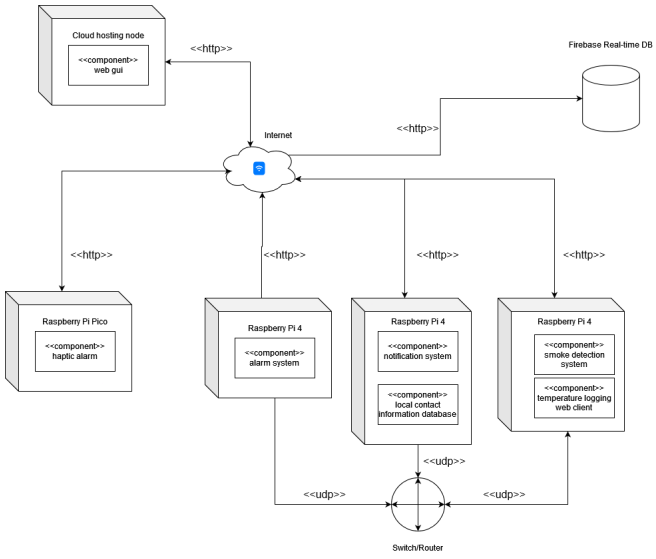


Figure 1: The FANS deployment diagram.

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.

2.2.2 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.

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"
      }
  }
}
```

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 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.

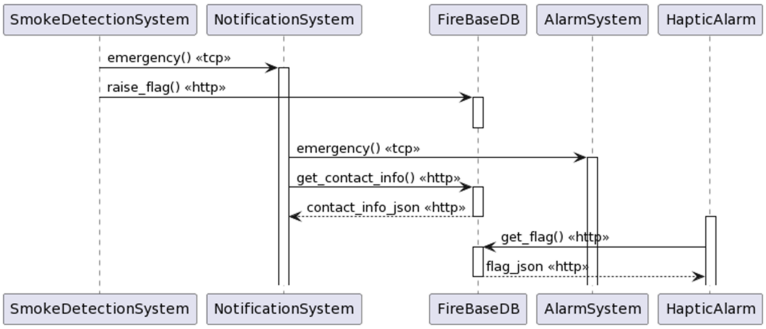


Figure 2: The sequence diagram for the trigger emergency use case.

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.

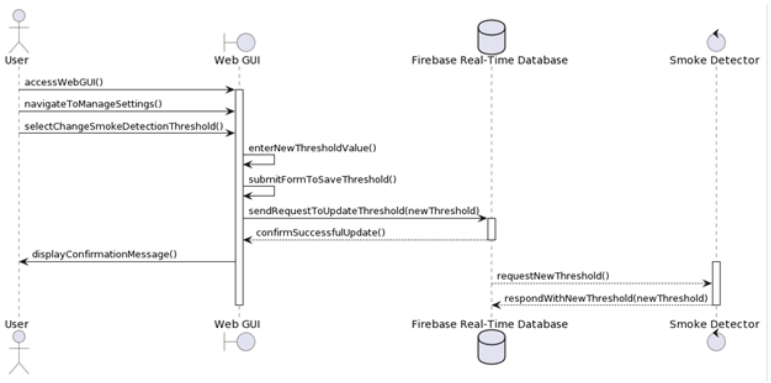


Figure 3: The sequence diagram for the changing emergency threshold use case.

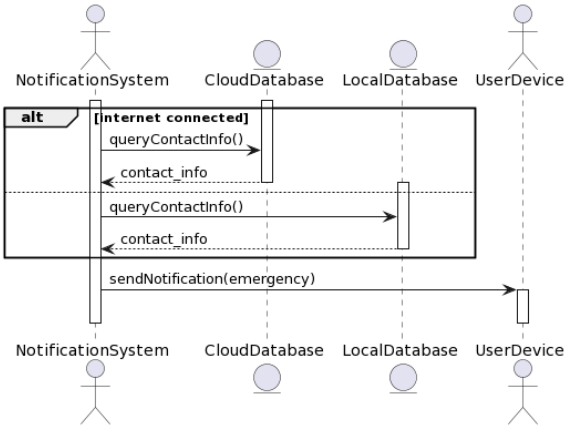


Figure 4: The sequence diagram for the emergency notification use case.

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

Although our team adhered to the specifications and objectives originally outlined in our proposal, we did make some modifications to our original plan as we learned more and got further into our project.

Changes Made

The first change that was made to FANS from the original specification was the LCD screen component. We had originally planned to add a monitor to the smoke detection unit, but realized during development that it would not have a practical use since the unit would likely be ceiling mounted where the monitor is not visible.

The second change to our original proposal, which was made several times, was our project timeline.

As we developed our system we realized that our time estimates were not completely accurate. We adjusted our timeline several times to allocate more resources to development of more difficult portions. For instance, we had to order new components after the component request period in order to properly interface with our smoke sensor. This caused us to push back the smoke detection portion of the system until the components arrived.

Findings

The smoke detection unit for our system was not entirely accurate due to electrical noise and some calibration requirements that could not be met. The data sheet advised to measure air resistance of smoke, but we could not find any fully reliable values for this online. We simply tuned the sensor to not trigger it's onboard LED in the absence of smoke. Our system reports a very small PPM measurement of smoke in clear environments due to the calibration issues and the electrical noise of our breadboard (likely due to the home-made logic level converter).

The temperature sensor was reliable and could accurately detect temperature changes, but had a baseline of around 38 degrees Celsius due to the heat from the CPU accumulating inside the Pi case. Removing the case or mounting the sensor further away from the CPU could help reduce this.

During our testing, we determined that from moment the smoke detection unit triggered an emergency, the response from all nodes in the system (alarms, lights, web UI) took a maximum of one second. This is likely dependent on the network speed, but our system's UDP messages are deliberately short for maximum transmission speed. The longest response time was for email notifications due to latency in the mail carrier outside of our control, but the notification system and other local nodes responded to the emergency trigger in under a second.

Scalability was also confirmed by doubling up on specific nodes during testing. We would convert our team's Pis to be two sensor nodes or two alarm nodes. Since our messages are sent over UDP and internet in a way that can be consumed by multiple listeners, all of our system nodes could be present in any number without any changes to the system's logic.

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 for files with multiple authors, 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, Appendix B
Javeria Sohail	Motivation, Problem Statement, Final Design Solution, Appendix B
Matteo Golin	Contributions, Deployment Diagram, Project Extensions, Sequence Diagrams, Discussion

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

5 Reflections

5.1 Project Success

Overall, the team considers our FANS to be 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 and multiple 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 reliable during testing. The system is easily expanded by simply running each distributed node on more devices, which is a key use case for installing the system in larger buildings. Our GUI provides an intuitive monitoring and control interface for users, allowing the user to check the emergency status, review the system's environmental conditions, and manage alarm timeouts.

The FANS project can be considered a success due to its functionality, reliability, and scalability. It meets all of the functional requirements originally laid out in the project proposal.

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, and applying our electronics knowledge to build circuits connected to microcontrollers were all essential to the growth of our technical ability.

Our time-management skills were also strengthened, as we learned to write our reports alongside development to keep records of our progress and design evolution. Communication amongst the team was critical, and we learned to leverage our varying skill sets across 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 inclusion of geographically aware system nodes would make FANS a more robust system. Making FANS nodes geographically aware would allow the emergency response to only target affected users. In addition, haptic alarm buzzers would only buzz if the wearer is within a certain range of the affected area. Only users 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, making it more effective.

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).

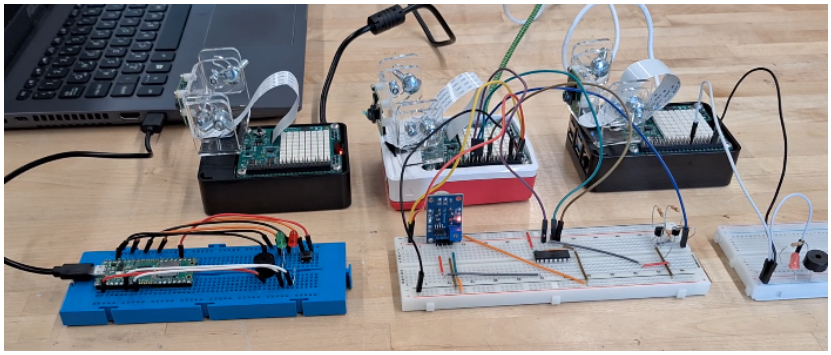
[4] "Mems pressure sensor: 260-1260 hpa absolute digital output barometer." (), [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiGq4Dy4_GEAxX1HNAFHSf-BRwQFnoECBcQAQ&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fdatasheet%2Flps25hb.pdf&usg=AOvVaw2FTksZHmMhimTAl6Qq3eFF&opi=89978449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

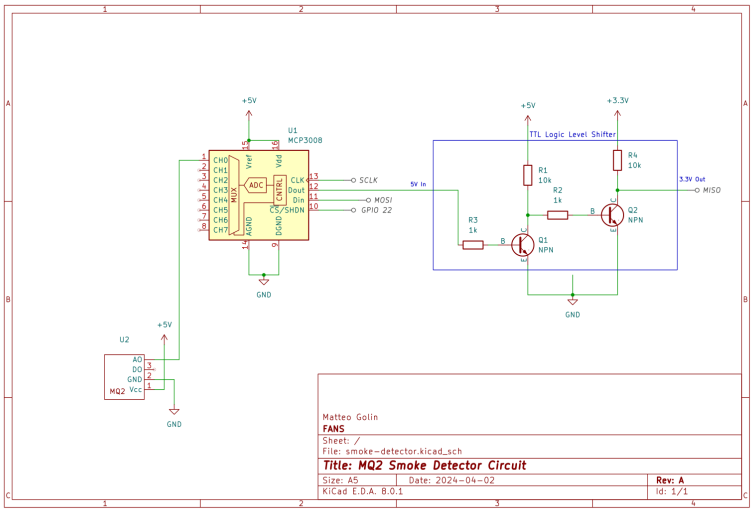
Guides to install and run each system node are provided below. Once all nodes have been set up, the system will be functional.

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:

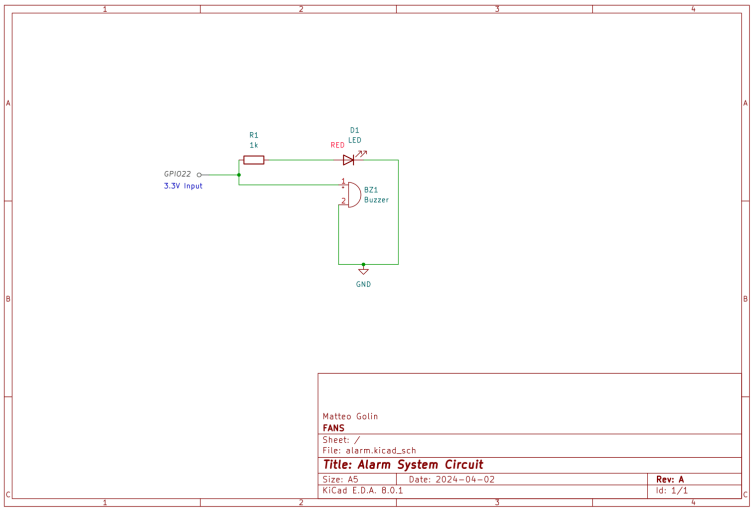
```
git clone https://github.com/SYSC3010-W24/sysc3010-project-l1-g8.git
cd sysc-project-l1-g8/sensor-pi
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python3 main.py
```

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:

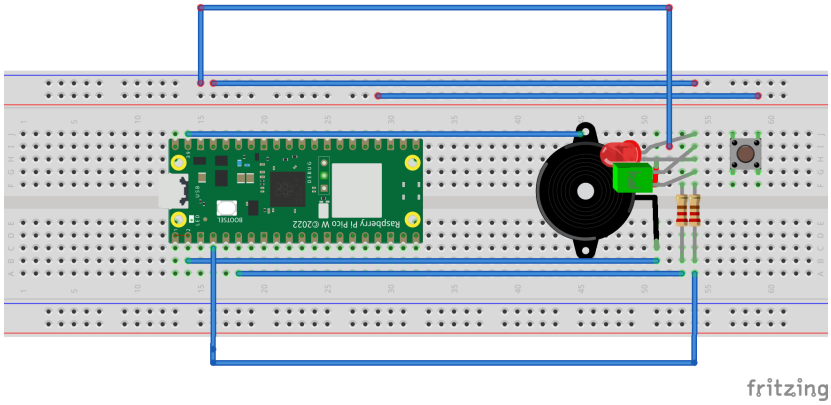
- `fans_credentials.json`: This file should contain the username and password required for authentication to the email server, enabling the system to send notifications via email.
- `firebase_config.json`: This file should include the configuration details required to connect to the Firebase database, 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

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.

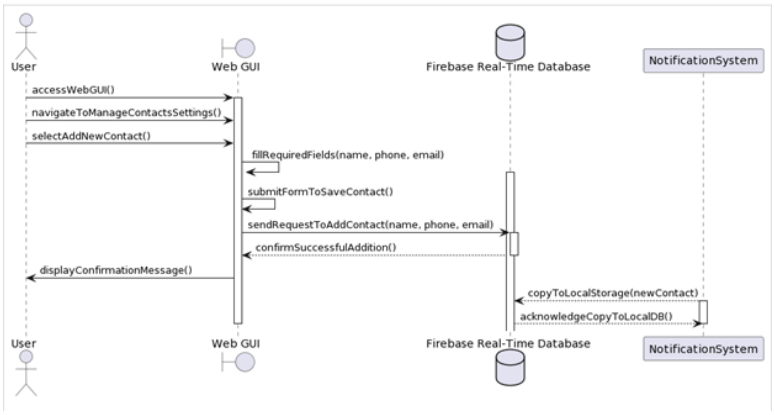


Figure 5: The sequence diagram for the adding new contact information use case.

C Schematics

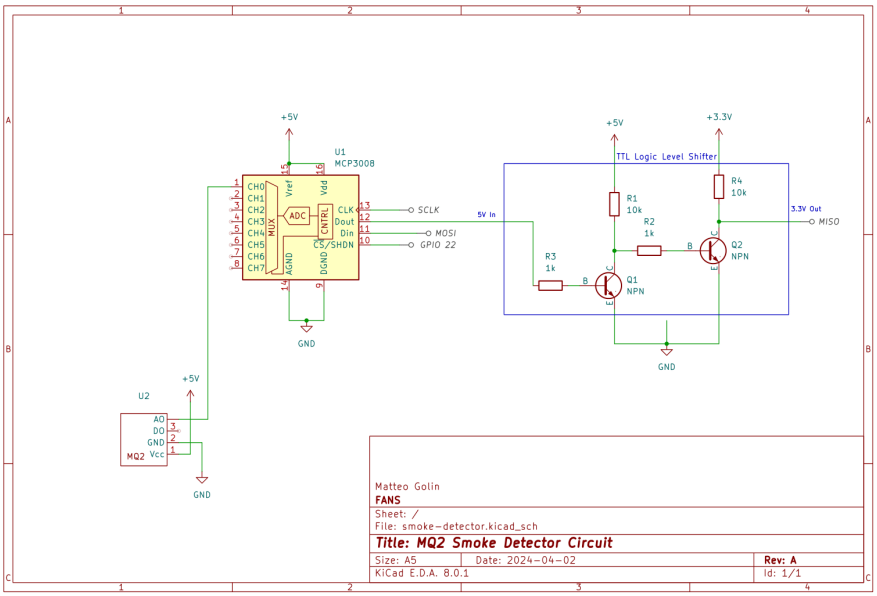


Figure 6: The smoke detection system's electrical connections.

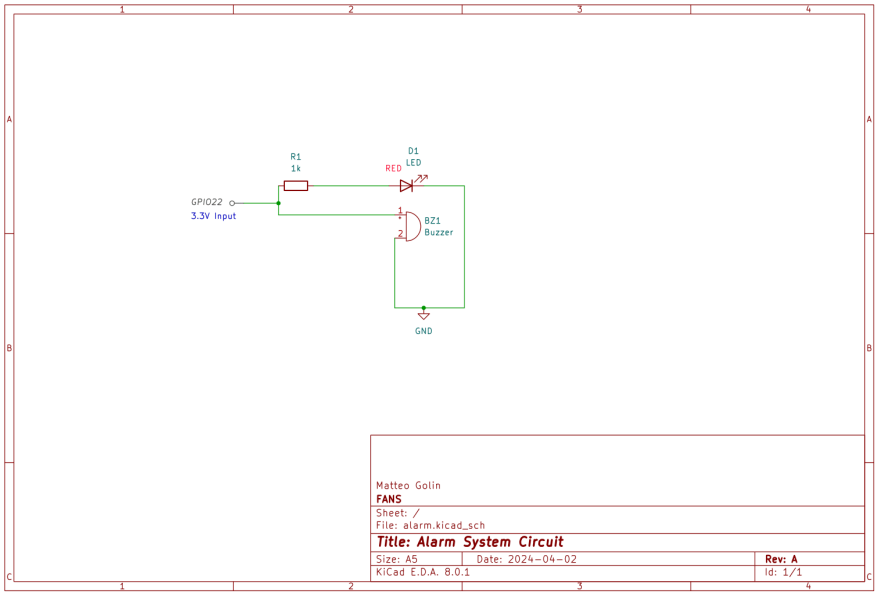


Figure 7: The alarm system's electrical connections.

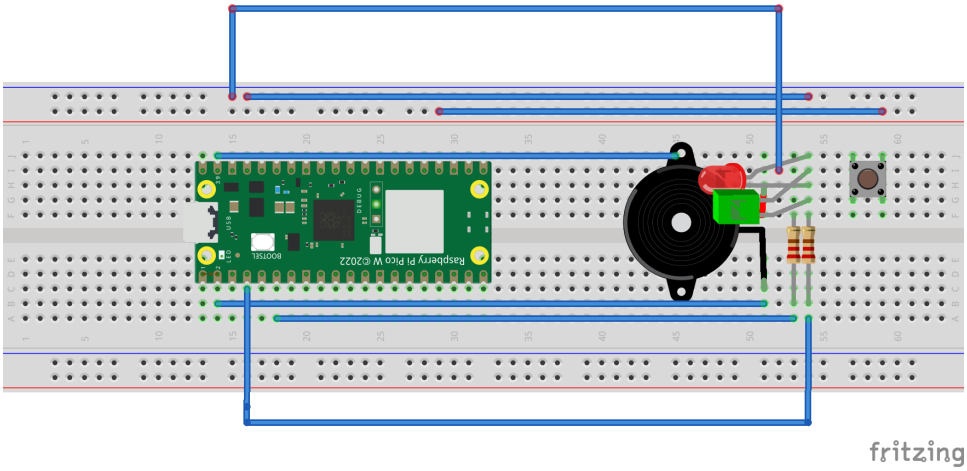


Figure 8: The haptic alarm system's wiring diagram on a breadboard.

D State Machine Diagrams

D.1 Alarm System

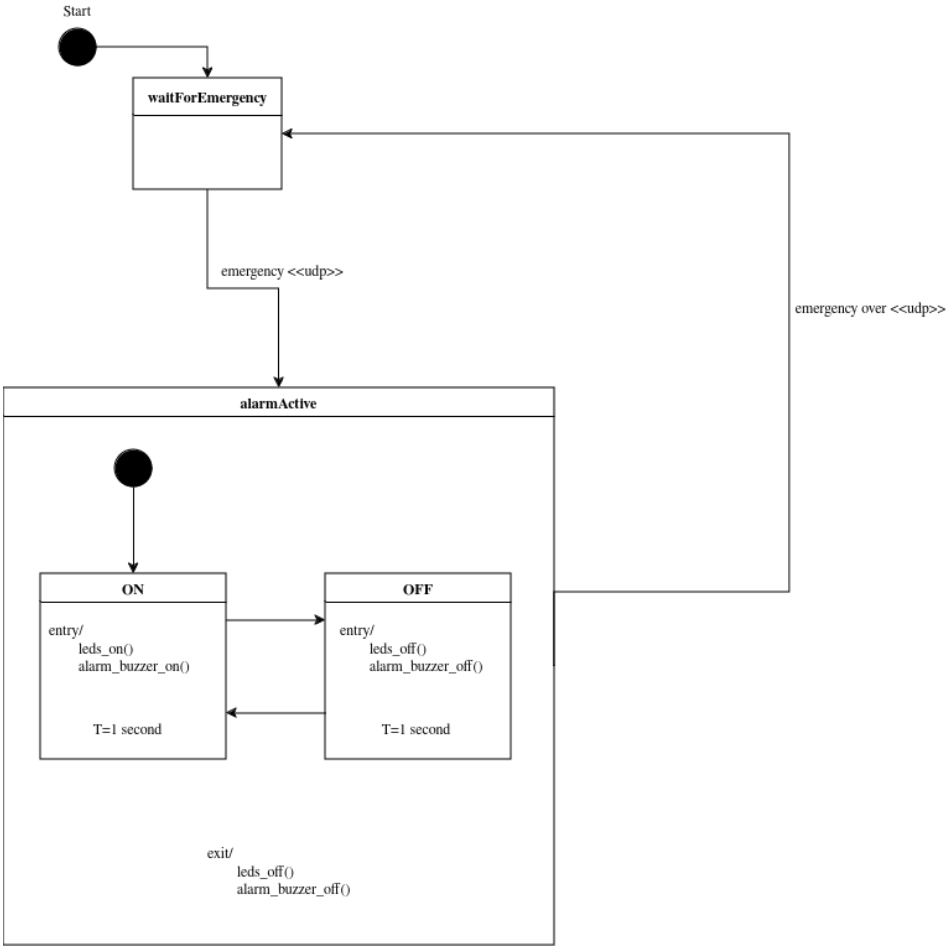


Figure 9: The alarm system's software state machine.

D.2 Smoke Detection System

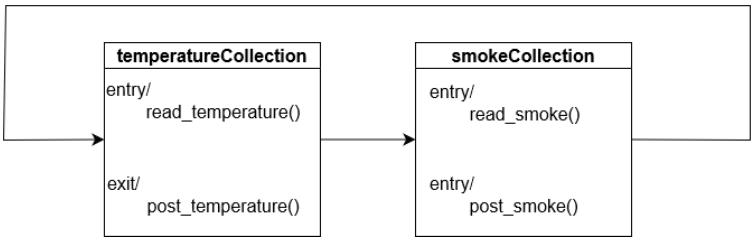


Figure 10: The smoke detection system's software state machine for data collection.

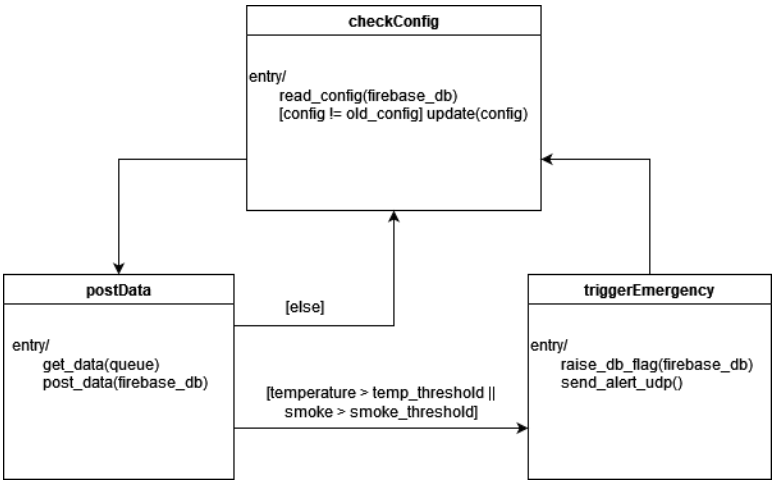


Figure 11: The smoke detection system's software state machine for performing its primary logic.

D.3 Notification System

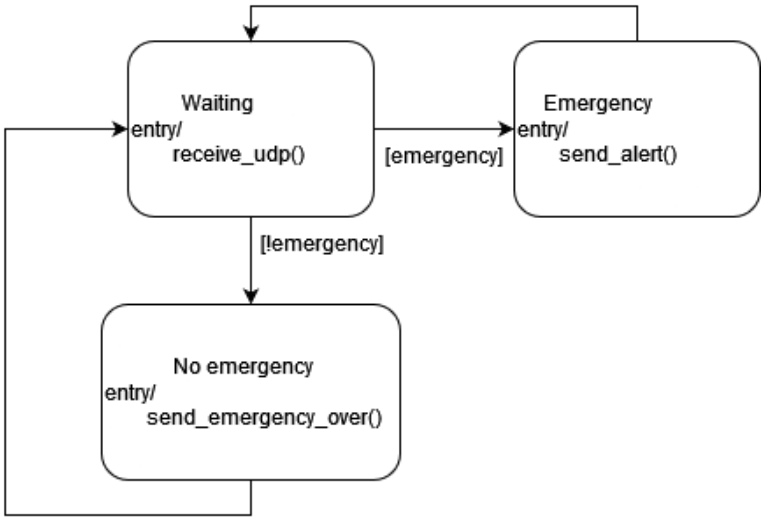


Figure 12: The primary logic of the notification system as a state machine.

D.4 Haptic Alarm

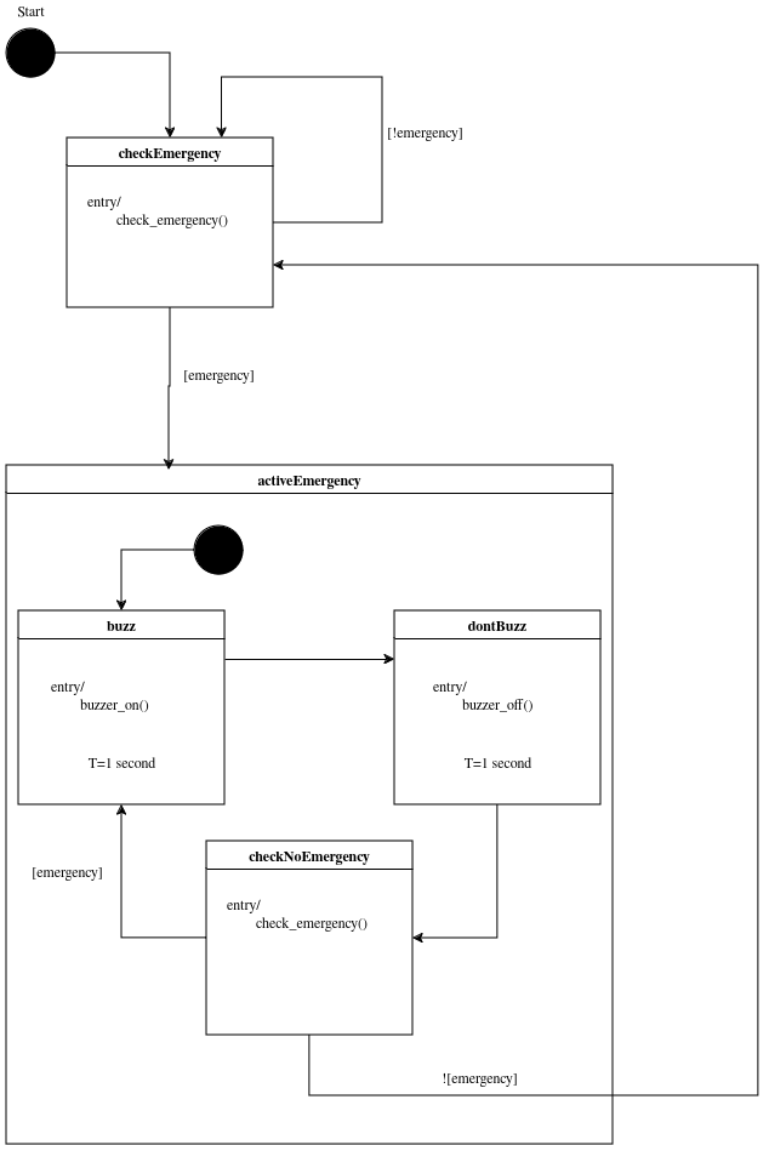


Figure 13: The primary logic of the haptic alarm as a state machine.