CS6006 Distributed, Cloud and IoT Systems

*London Metropolitan University Library Sound Alert System and Visualisor*

Report

Names:

21002595 - Hasan Khan

22043887 - Fatma Asad

21020258 - Aisha Akram

22044754 - Fabian Hasan

Module Leader: Dr. Bilal Hasan

Date: Thursday 22 May 2025

# Table of Contents

[Table of Contents 2](#_Toc196409824)

[Table of Figures 4](#_Toc196409825)

[Executive Summary 5](#_Toc196409826)

[Introduction 6](#_Toc196409827)

[Problem Definition 7](#_Toc196409828)

[System Architecture and Design 8](#_Toc196409829)

[Hardware Components 8](#_Toc196409830)

[Software and Cloud Components 8](#_Toc196409831)

[Design Justification 9](#_Toc196409832)

[Implementation 9](#_Toc196409833)

[Hardware Setup: 9](#_Toc196409834)

[Python Script Overview 10](#_Toc196409835)

[1. soundTester.py 10](#_Toc196409836)

[2. DebugST.py 11](#_Toc196409837)

[3. connTest.py 11](#_Toc196409838)

[4. debugAndTester.py 12](#_Toc196409839)

[5. dcbConn.py 13](#_Toc196409840)

[6. SoundData1.py 13](#_Toc196409841)

[Azure Configuration 14](#_Toc196409842)

[Testing and Validation 14](#_Toc196409843)

[Security 15](#_Toc196409844)

[Cost and Performance Consideration 16](#_Toc196409845)

[Costs 16](#_Toc196409846)

[Performance 16](#_Toc196409847)

[Findings 17](#_Toc196409848)

[Analysis 17](#_Toc196409849)

[Recommendation and Future Work 18](#_Toc196409850)

[Conclusion 19](#_Toc196409851)

[References 20](#_Toc196409852)

[Appendices 21](#_Toc196409853)

[Appendix A: Python Scripts 21](#_Toc196409854)

[1. soundTester.py 21](#_Toc196409855)

[1.1 soundTester2.py 21](#_Toc196409856)

[2. debugST.py 22](#_Toc196409857)

[3. connTest.py 22](#_Toc196409858)

[4. debugAndTester.py 23](#_Toc196409859)

[5. dcbConn.py 24](#_Toc196409860)

[Appendix B: Azure Setup Steps 25](#_Toc196409861)

[1. Azure Subscription 25](#_Toc196409862)

[2. iotSoundSensorTemplate and mySoundSensor Device 26](#_Toc196409863)

[3. Device Connection Groups 28](#_Toc196409864)

[Appendix C: Azure Validation Tools 29](#_Toc196409865)

[1. Stream Analytics 29](#_Toc196409866)

[2. Azure Time Series Insights 30](#_Toc196409867)

# Table of Figures

[Figure 1: Physical Set Up of Device 8](#_Toc196410027)

[Figure 2: ARM I2C Interface Enabled and the Installation of I2c Tools 9](#_Toc196410028)

[Figure 3: Terminal Output Showing Results from soundTester.py 10](#_Toc196410029)

[Figure 4: Output Result of debugAndTester.py 12](#_Toc196410030)

[Figure 5: Terminal Output showing Results from dcbConn.py 13](#_Toc196410031)

[Figure 6: Sample Email Alert 14](#_Toc196410032)

[Figure 7: Pricing and Scale 16](#_Toc196410033)

[Figure 8: Noise Level Analytics 17](#_Toc196410034)

[Figure 9: soundTester.py script 21](#_Toc196410035)

[Figure 10: soundTester2.py script 21](#_Toc196410036)

[Figure 11: debugST.py script 22](#_Toc196410037)

[Figure 12: connTest.py script 22](#_Toc196410038)

[Figure 13: debugAndTester.py script 23](#_Toc196410039)

[Figure 14: dcbConn.py script 24](#_Toc196410040)

[Figure 15: Creation of Azure subscription (Smart Meter Analytics) 25](#_Toc196410041)

[Figure 16: iotSoundSensorTemplate 26](#_Toc196410042)

[Figure 17: Creation of mySoundSensor Device 26](#_Toc196410043)

[Figure 18: DTDL of iotSoundSensorTemplate 27](#_Toc196410044)

[Figure 19: Information of Device Connection Groups 27](#_Toc196410045)

[Figure 20: Creation of New Stream Analytics Job 28](#_Toc196410046)

[Figure 21: Deployment of Stream Analytics Job 29](#_Toc196410047)

[Figure 22: Overview of Azure Time Series Insights 29](#_Toc196410048)

# Executive Summary

This report presents the development and implementation of a smart IoT-based system to monitor and evaluate noise levels in the university library. Using a Raspberry Pi integrated with a KY-037 sound sensor and ADS1115 analog-to-digital converter, the system continuously collects sound data across four library zones. These readings were conducted hourly between 8AM and 5PM over one week.

Data is transmitted in real time to Azure IoT Hub and an Azure IoT Central Application. We used Python scripts for sensor communication and IoT central to trigger email alerts to library staff via email when sound threshold is breached. This ensures staff are informed of excessive noise levels, even when physically absent from the monitored zones. This helps to suggest best times for students to study based on historical sound levels.

Our motivation comes from improving student productivity and concentration by providing insights into quiet periods and zones in the library. This report details system architecture, design, implementation, security considerations, costs and key findings - concluding with reflections and future work recommendations.

# Introduction

Our team worked collaboratively throughout this project, effectively assigning tasks and maintaining communication to ensure steady progress. Weekly meetings took place in the library and workshops, providing an ideal setting for hands-on setup and testing.

We encountered no major issues either technically or personally, which made the process smooth and efficient. The motivation behind the project was our shared desire to create a helpful tool for current and future students. Libraries should be spaces for focus and productivity, but frequent noise complaints suggest otherwise. We believe this project can not only guide students towards the quietest study periods but also empower library staff to enforce noise control policies more efficiently.

This project is part of a much wider goal to foster a culture of collaboration, discipline and academic motivation to reinforce the library’s role as a key hub for learning/betterment.

# Problem Definition

Noise pollution in academic environments is a growing concern. Students often voice frustrations over frequent disturbances in the library which discourages them from using the space for revision and/or study. While the library is designed to support academic focus, uncontrolled noise levels in various zones disrupt that purpose.

Currently, there is no real-time system to monitor or report sound levels across library zones. Library staff often cannot supervise all areas simultaneously, leading to unchecked disturbances during peak times. Manual monitoring methods are inefficient and reactive, rather than proactive.

Our project addresses this gap by developing an IoT-based monitoring and alert system that automatically records sound data and notifies staff when noise exceeds a set threshold. In addition, students can benefit from the visual representation of data, which highlights the quietest hours and zones. This enables them to plan their study sessions effectively. Overall, the system ensures a more productive and supportive academic environment through automation, transparency and accessibility.

# System Architecture and Design

Our system integrates physical hardware and cloud-based services to monitor noise levels in four zones of the library.

## Hardware Components

* Raspberry Pi: Acts as the central processing unit, executing Python scripts to collect and transmit data.
* KY-037 Sound Sensor: Measure sound levels in real-time. It’s sensitive to environmental noise and ideal for indoor usage.
* ADS1115 Analog-to-Digital Converter: Converts analog signals from the KY-037 sensor into calculable and readable by the Raspberry Pi.
* Jumper Wires (Female-to-Female): Facilitate connections between the sensor, ADC and Raspberry Pi pins. We avoided using a breadboard for simplicity and reliability.

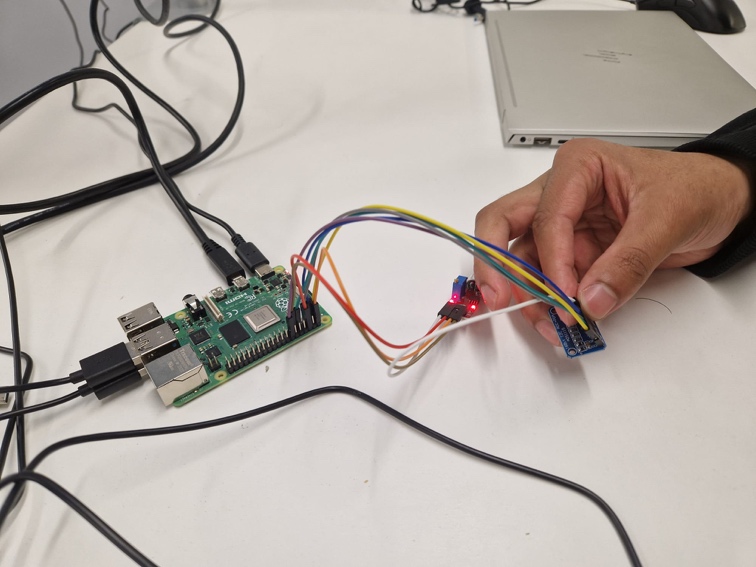


Figure 1: Physical Set Up of Device

Data was collected and logged continuously over a 10-minute span every hour. This design ensures consistent sampling without overwhelming the system or Azure storage limits.

## Software and Cloud Components

* Python: Python scripts on the Raspberry Pi handled GPIO control, data collection/retrieval/transmitting and the Azure IoT Hub as well as the IoT Central Application connections. We had all of our code in virtual environments as to not interfere with the other python libraries.
* Azure Iot Hub: Received data from Raspberry Pi and served as the central hub for real-time and batch processing.
* Azure IoT Central: An application where we hosted the visualization and email alerts. Monitoring and connectivity had to be separate to IoT Hub.

The system was designed to be scalable. All components were chosen for ease of integration, reliability, and support within the Azure ecosystem.

## Design Justification

The KY-037 and ADS1115 provided an affordable yet accurate way to measure the decibel levels. Avoiding the breadboard reduced electrical noise and increased stability. Since we were only running one component, there was no need for a breadboard.

Practically, adding more zones or integrating machine learning models could assist for anomaly detection such as a fly flying too close to the sensor will affect averages and send alerts. Overall, the architecture supports real-time monitoring and long-term data analysis, making it ideal for academic environments.

# Implementation

## Hardware Setup:

We assembled the system by connecting the KY-037 sound sensor to the ADS1115 analogue-to-digital converter and then to the raspberry pi using jumper wires. A monitor, keyboard and mouse were connected for initial setup. Once booted, the Raspberry Pi OS was configured with Python3 and the necessary libraries.

A computer screen with a message

AI-generated content may be incorrect.A computer screen with text on it

AI-generated content may be incorrect.

Figure 2: ARM I2C Interface Enabled and the Installation of I2c Tools

These show early configuration of the Raspberry Pi.

## Python Script Overview

The following scripts were developed to enable noise detection, data formatting and transmission to the Azure IoT Hub and Azure IoT Central Application. The full content of each script is provided in Appendix A, while this section explains each script’s purpose, logic and a sample output screenshot.

soundTester.py (1)

Purpose: This script serves as a quick test to check whether the digital output pin of the sound sensor (KY-037) detects noise.

How it Works:

1. GPIO Setup:

Configures a Raspberry Pi GPIO pin (11) as an input pin.

1. Noise Detection: The script continuously monitors the input.
   * If it’s there’s noise, the system prints “NOISE DETECTED!!!”
   * If there isn’t noise, it prints: “No Sound.”

Example Output:

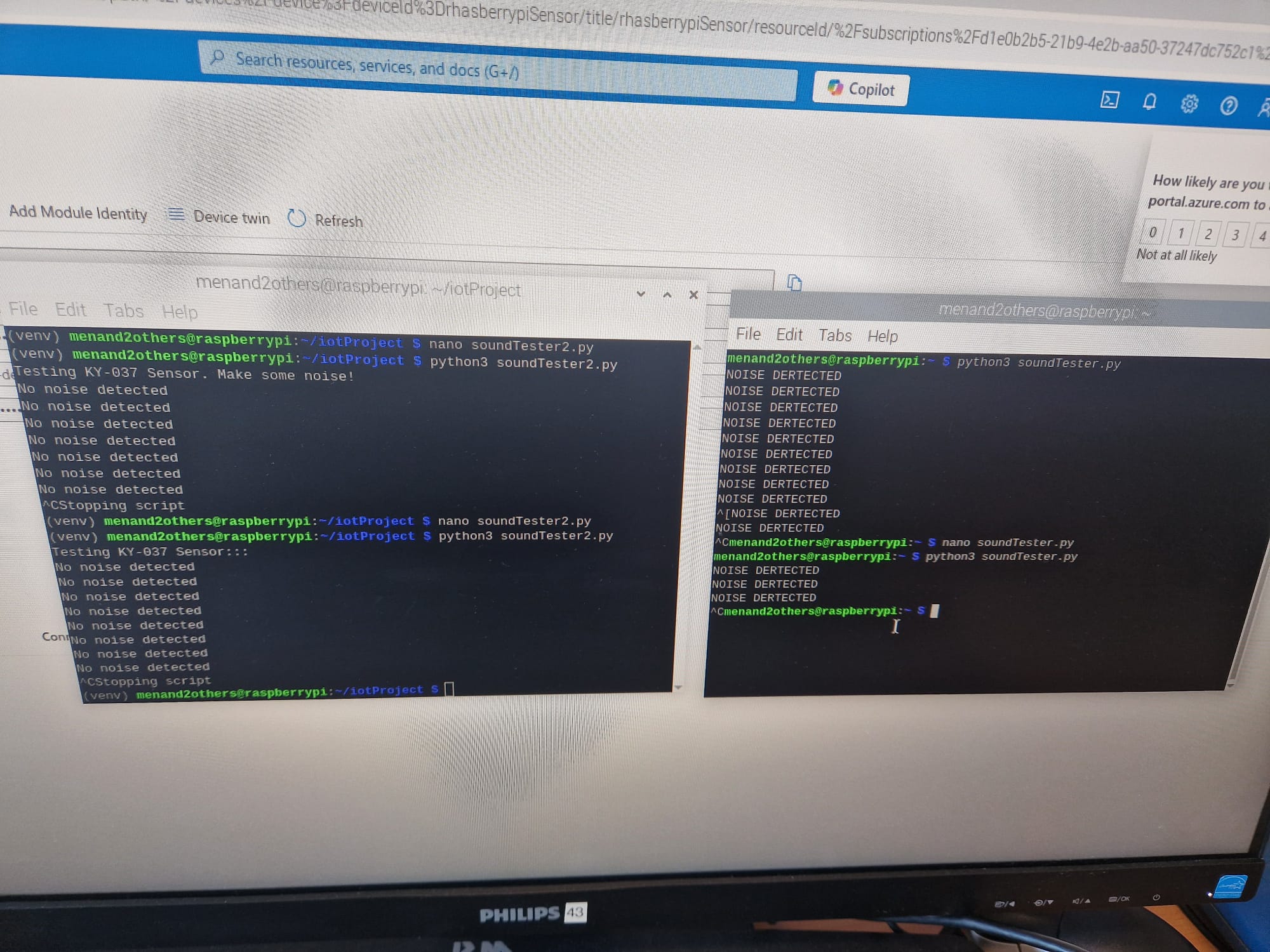


Figure 3: Terminal Output Showing Results from soundTester.py

## debugAndTester.py (2)

Purpose: This script reads a digital signal from the KY-037 sound sensor using the pigpio library on the Raspberry Pi, then determines if noise is detected and then sends that information to Azure IoT Hub.

How it Works:

1. GPIO Setup: Initialises pigpio to receive the KY-037’s digital output
2. Noise Detection: Reads the digital state (0 = noise detected, 1 = no noise) in a continuous loop.
3. Azure IoT: Uses the connection string to connect to the IoT Hub and sends telemetry (noise\_detected: true/false) every second.

Example Output:

A computer screen shot of a computer screen

AI-generated content may be incorrect.

Figure 4: Output Result of debugAndTester.py

## dcbConn.py (3)

Purpose: This script measures the readings from the sound sensor through the ADS1115 and calculates the decibel (dB) reading.

How it Works:

1. Connection: It creates an I2C connection to the ADS1115 and reads the microphone signal from channel pin A0 on the ADS1115.
2. Sampling: In each loop iteration, it collects multiple samples to compute the RMS voltage.
3. dB Calculation: It converts the RMS voltage to decibels using a predefined reference voltage (reference\_voltage = 0.00631), which serves as a rough baseline for microphone sensitivity.

Example Output:

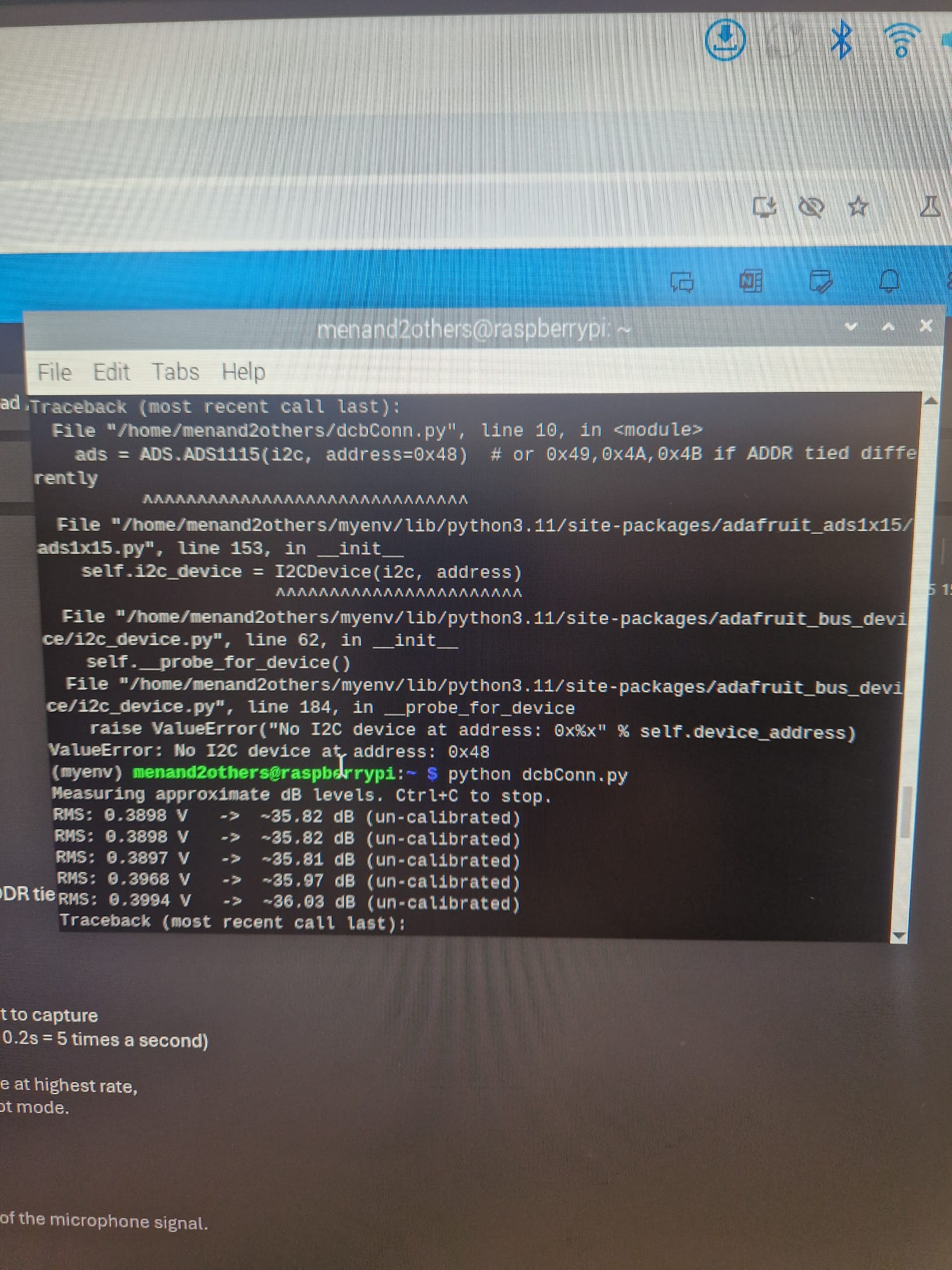


Figure 5: Terminal Output showing Results from dcbConn.py

## SoundData1.py (4)

Purpose: This script measure approximate noise levels (in decibels) via the microphone sensor connected to the ADS1115 ADC and sends these measurements to Azure IoT Central.

How it Works:

1. I2C/ADC Setup: Uses and I2C connection (busio.I2C) to communicate with the ADS1115 ADC, reading the microphone voltage from analog input A0.
2. IoT Central Connection: Authenticates using the Device credentials (ID\_SCOPE, DEVICE\_ID, PRIMARY\_KEY) and then records it to Azure IoT Central.
3. Telemetry Sending: Packages the decibel value into JSON and sends it to Azure IoT Central in a continuous loop, sleeping a few seconds between sends.

A computer screen with white text

AI-generated content may be incorrect.

## Azure Configuration

1. Device Registration: Raspberry Pi was registered as an IoT device in Azure IoT Hub and Azure IoT Central Application.
2. Visualisation: The data was then turned into graphs outputting the highest/lowest average as well as a line graph to display the average levels alongside time. We also displayed the last known average value.
3. Alert System: This system monitored for values above the set threshold (e.g., 60+ dB) and triggered email alerts using Office 365 connectors.

*See Appendix B for Azure Setup Steps*

## Testing and Validation

We tested the system in all four zones. Screenshots of Azure dashboards and email alerts were taken to validate functionality. The result was a reliable, real-time feedback system that can provide actionable insights to staff and students alike.

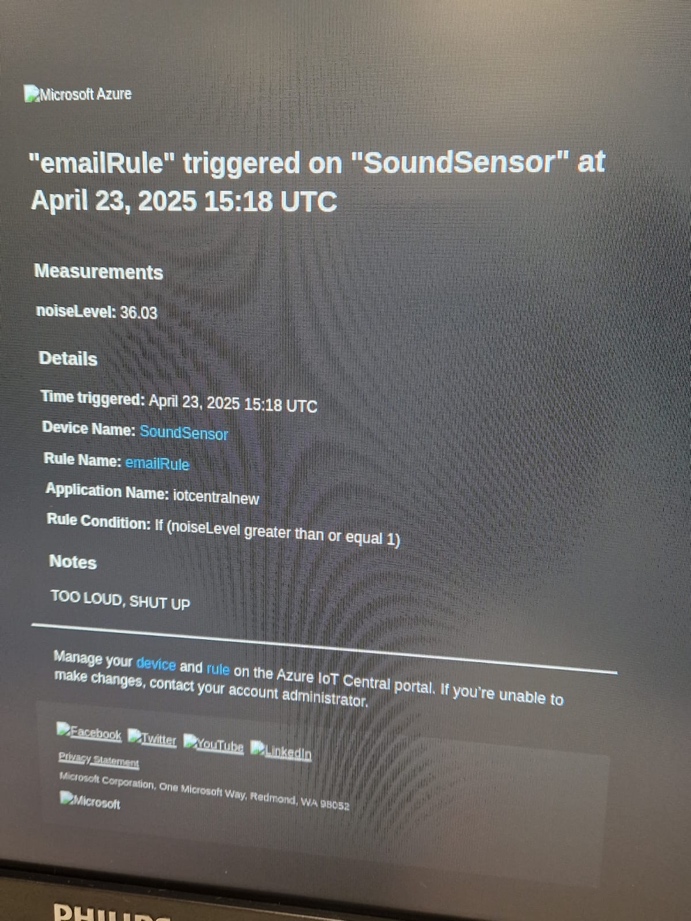
**

Figure 7: Sample Email Alert

Figure 6 displays an automated email notification triggered when noise levels exceed a predefined threshold. In a practical deployment, this threshold would typically be set at around 50-60 decibels or higher, depending on the acceptable noise level for each zone.

*See Appendix C for further Azure Validation Tools*

# Security

Security consideration included:

1. Data integrity: Ensuring that the data sent from the Raspberry Pi was not tampered with. Azure IoT Hub provided secure communication using symmetric keys.
2. Access Control: Only authorised devices and users had access to the Azure resources. We used role-based access control (RBAC) for management.
3. Email Vulnerabilities: Email alerts, if intercepted, could reveal location data or usage patterns. We used university accounts with MFA enabled.
4. Hardware Tampering: The physical setup could be disrupted by unauthorised access in the library, especially the unsoldered ADS1115, which required delicate handling.

Furthermore, anomalies like sudden loud events (e.g., maintenance work or celebrations) could skew data which reduces overall system accuracy. Future versions could use machine learning to filter out anomalies.

# Cost and Performance Consideration

## Costs

|  |  |
| --- | --- |
| Component | Price |
| KY-037 Sensor | £5.50 |
| ADS1115 ADC | £6.99 |
| Jumper Wires | £7.00 |

Total hardware cost: ~ £19.5

Total Azure cost: ~ £0.00 (we used the free tiers)

We used free-tier services where possible. However, a fully deployed version could cost more due to storage, email triggers, data processing, etc.

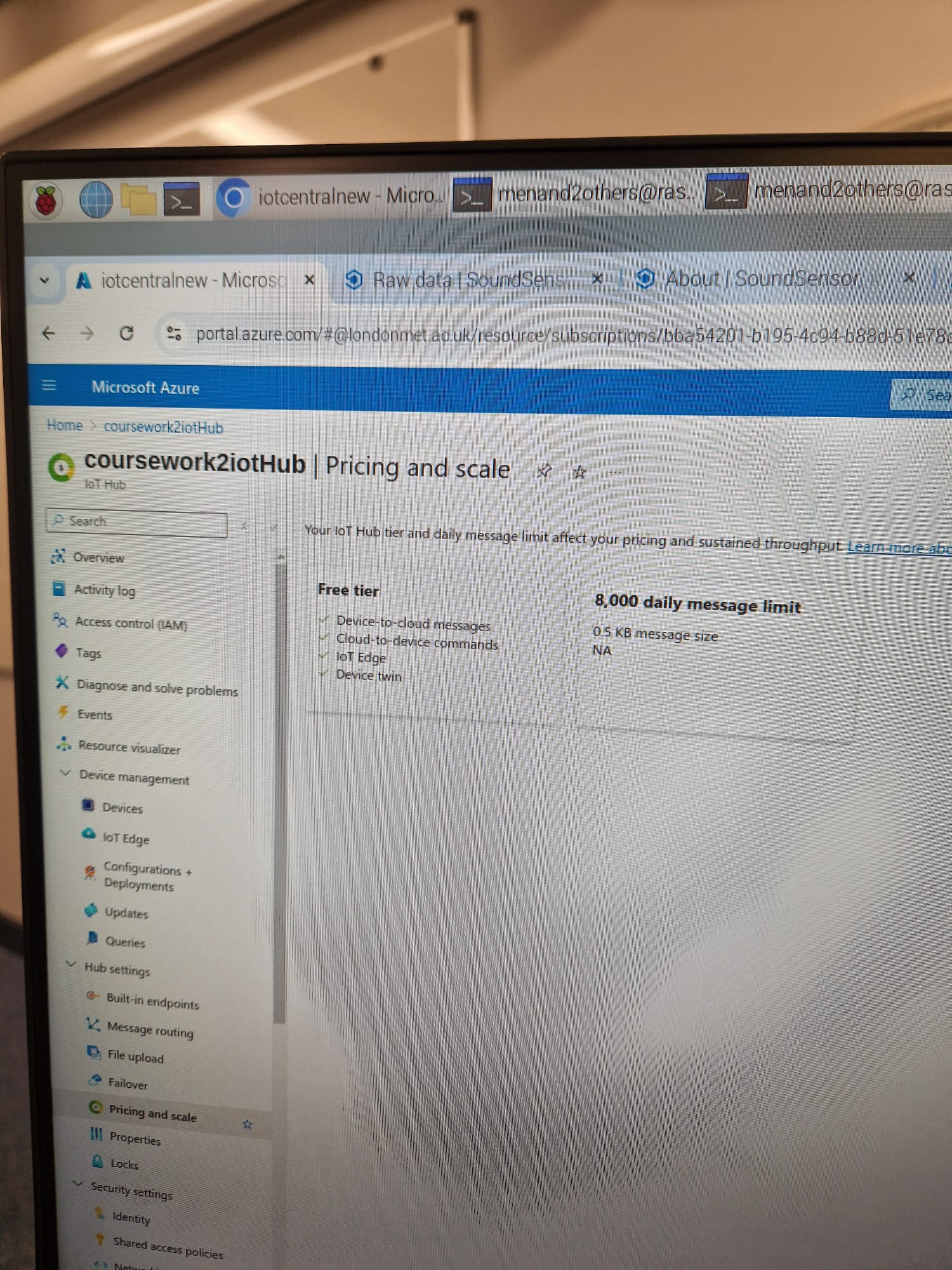


Figure 8: Pricing and Scale

## Performance

1. Sensor Sensitivity: The KY-037 performed well but was affected by placement and obstructions.
2. Connection Stability: Raspberry Pi required a stable Wi-Fi connection to maintain real-time sync.
3. Data Accuracy: Manual holding of the ADS1115 resulted in occasional connection drops and having to restart the 10-minute recordings. Soldering would improve stability.

# Findings

Visualisation of our data over one week revealed the following:

1. Quietest Times: Typically, between 8AM – 10AM
2. Noisiest Zones: near the entrance during midday (12PM – 2PM)
3. Email Alerts: Successfully triggered multiple times and matched with noise spikes.

## Analysis

Data suggested that students frequently used the lounge area during breaks, contributing to increased sound levels. In contrast the upper levels remained quiet consistently, likely due to its isolated layout.

The system achieved its objective of identifying ideal study periods and alerting staff of noise breaches. However, several limitations were noted:

1. Inconsistent Wi-Fi in some library zones
2. Temporary disconnections of unsoldered hardware
3. Occasional spikes in data due to external noise interference

Despite these, the collected data was sufficient for actionable insights and can be extended to other university spaces. The image below shows data along a 10minute period that was interupted by wifi signal failure and we did another test to show how visualisations are effected.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Noise Level Analytics

# Recommendation and Future Work

We recommend soldering all connection for long-term stability and making sure the microphone is not obstructed. The system could be extended to cover more zones and could be integrated with mobile apps for real-time student access.

Future work includes:

1. Expanding to other university areas (public access campus environments such as the courtyard).
2. Using Power BI to display real-time dashboards which could be used in presentations for future building work.
3. Exploring machine learning to filter anomalies

This would enhance the system’s usability and reliability, supporting a better study environment.

# Conclusion

This project successfully demonstrated a working prototype of an IoT-based sound monitoring system. It provided real-time alerts and visualised historical data to help students choose optimal study times.

Our solution addresses core issues of manual monitoring and inconsistent noise regulation in academic environments. Key lessons learned included the importance of component integration, cloud services familiarity and testing under real conditions. The hands-on setup also deepened our understanding of sensor calibration and cloud architecture.

While not flawless, the system provides a strong foundation for future iterations that can enhance accuracy, automation and coverage.

# References

* Alam, Mamtaz. “How to Use ADS1115 16-Bit ADC Module with Raspberry Pi.” *How to Electronics*, 17 Aug. 2023, how2electronics.com/how-to-use-ads1115-16-bit-adc-module-with-raspberry-pi/.
* Microsoft. “IoT Concepts and Azure IoT Hub.” *Learn.microsoft.com*, 24 Feb. 2024, learn.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub.
* Microsoft. “Azure IoT Hub Documentation.” *Learn.microsoft.com*, 2023, learn.microsoft.com/en-us/azure/iot-hub/.
* “Raspberry Pi Documentation.” *Www.raspberrypi.com*, www.raspberrypi.com/documentation/.
* logMaker360. “Connect Sensor to Raspberry Pi with Python.” *YouTube*, 20 Apr. 2017, www.youtube.com/watch?v=5A35o5mEZYQ. Accessed 23 Apr. 2025.
* Phazer Tech. “Sensors on Raspberry Pi & Python - Distance Sensor & Inertial Measurement Unit.” *YouTube*, 19 July 2024, www.youtube.com/watch?v=nGYZChwjiMo. Accessed 23 Apr. 2025.
* Shilleh. “Getting Started with the KY-037 Sound Sensor and Raspberry Pi: Detecting Sound Using Python.” *YouTube*, 2 Oct. 2024, www.youtube.com/watch?v=P5hbVwG6eBs. Accessed 23 Apr. 2025.
* Microsoft. “What Is Power BI? - Power BI.” *Learn.microsoft.com*, 22 Mar. 2024, learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview.

# Appendices

## Appendix A: Python Scripts

Below are all the python scripts mentioned in Python Script Overview:

## soundTester.py

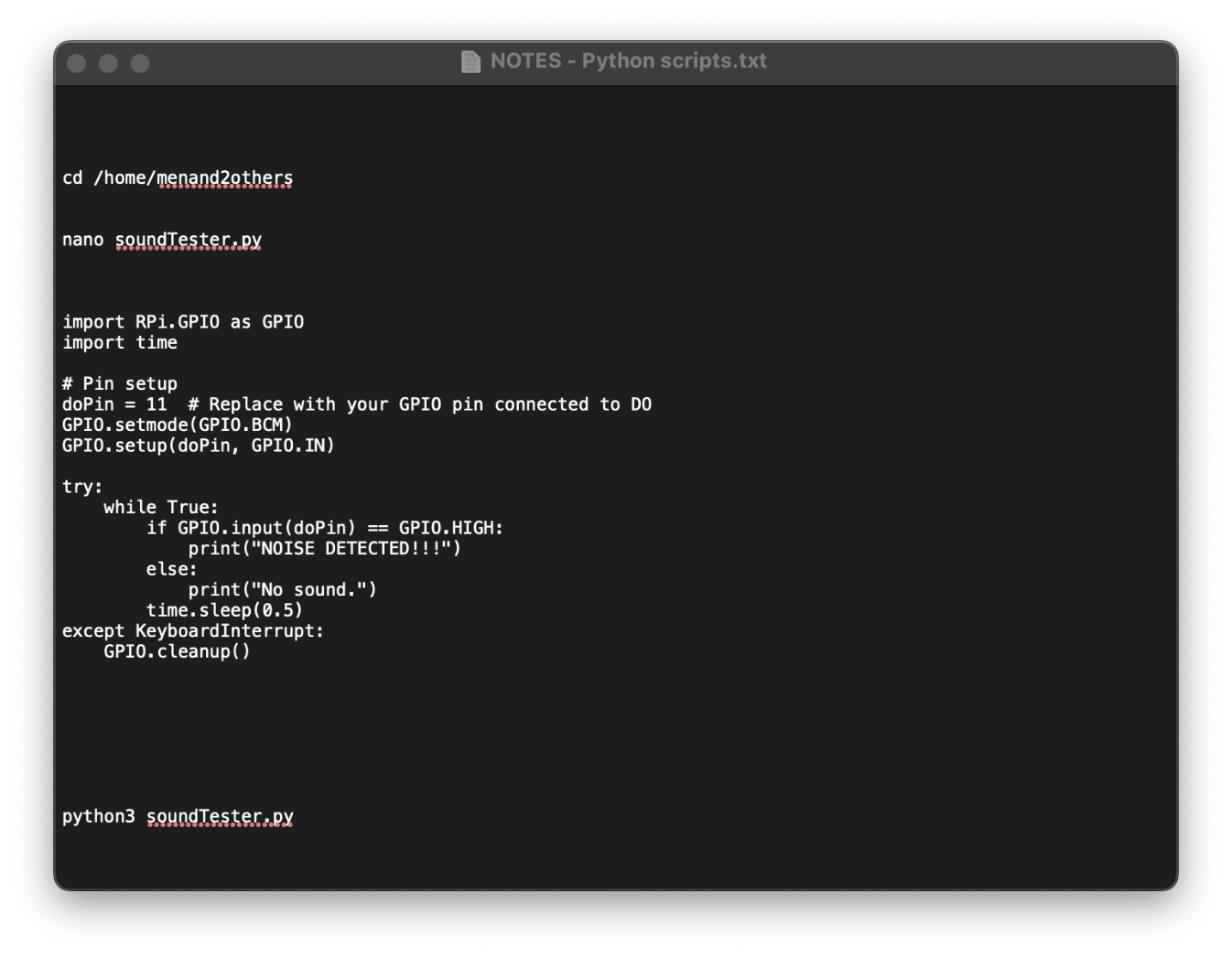


Figure 10: soundTester.py script

## debugAndTester.py

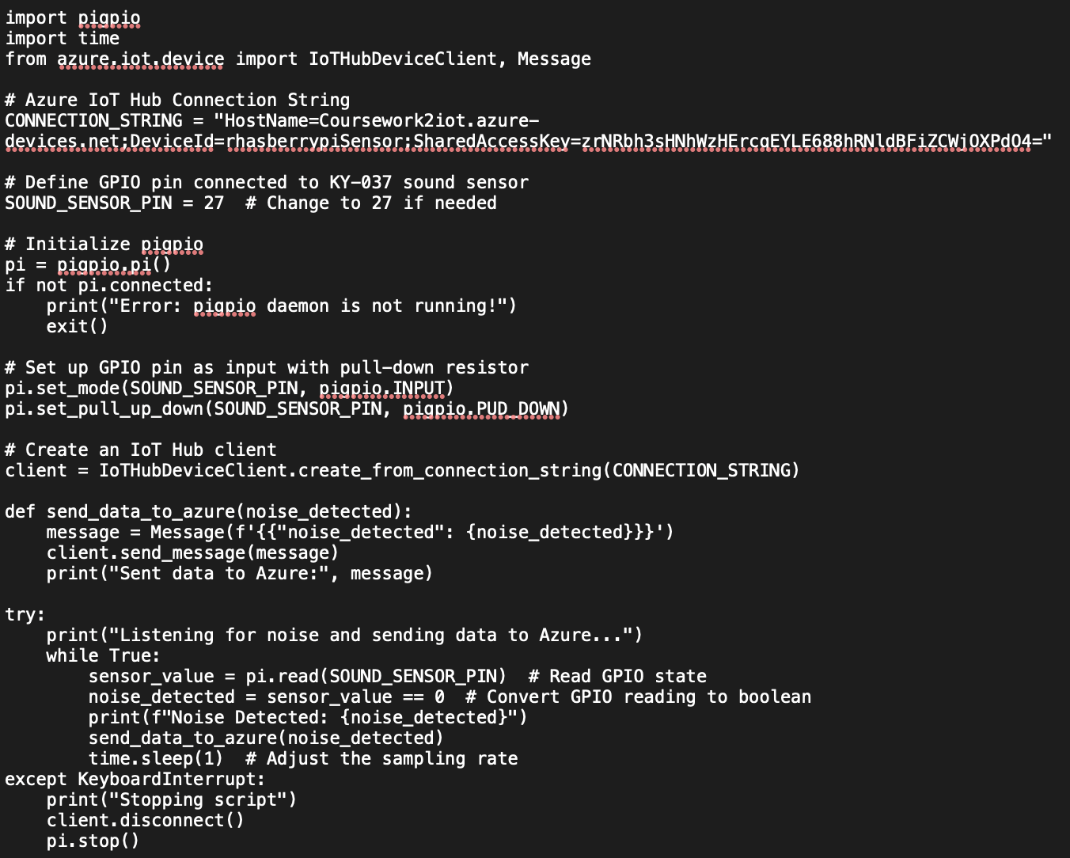


Figure 14: debugAndTester.py script

## dcbConn.py

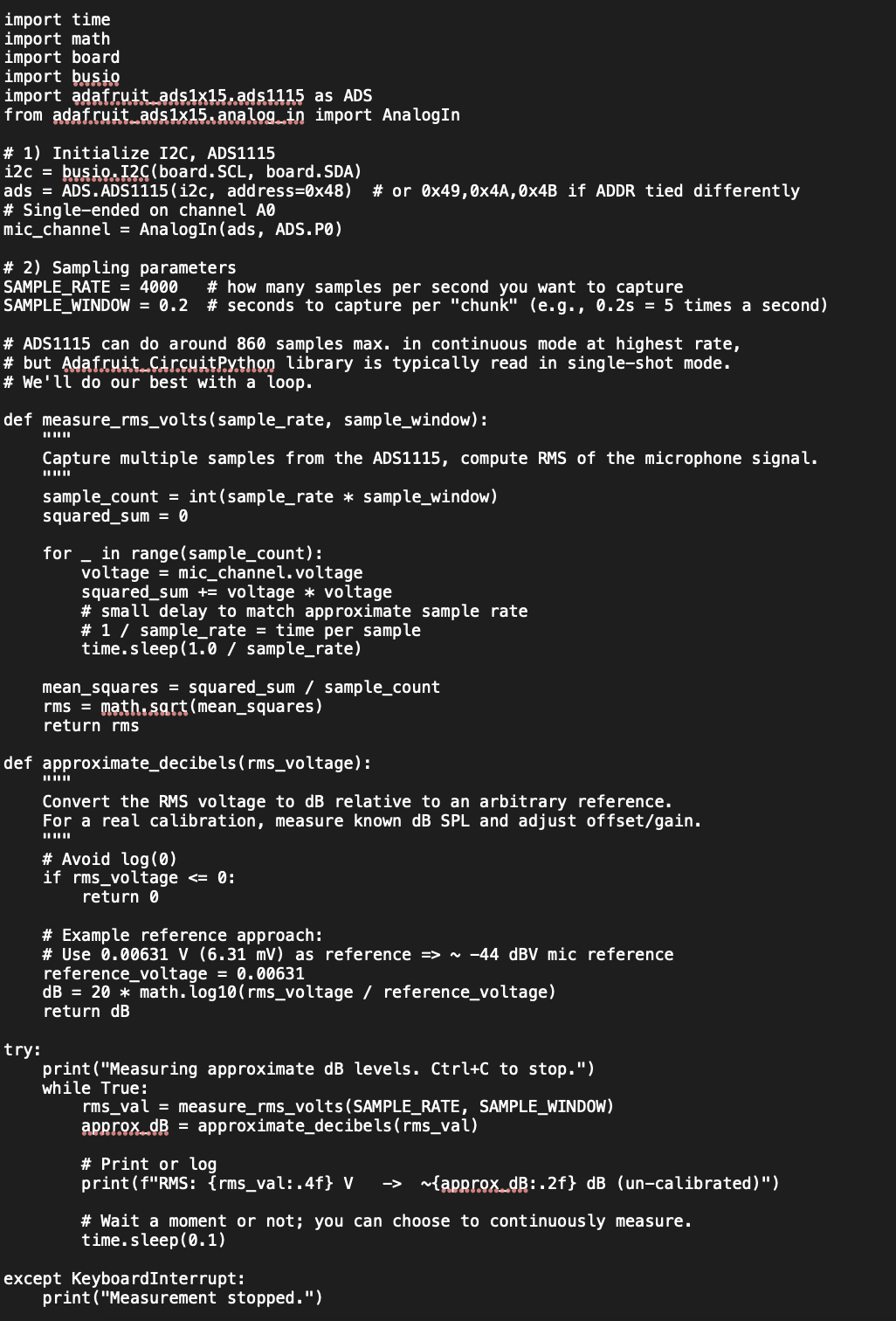


Figure 15: dcbConn.py script

INCLUDE soundData1.py

# Appendix B: Azure Setup Steps

## iotSoundSensorTemplate and mySoundSensor Device

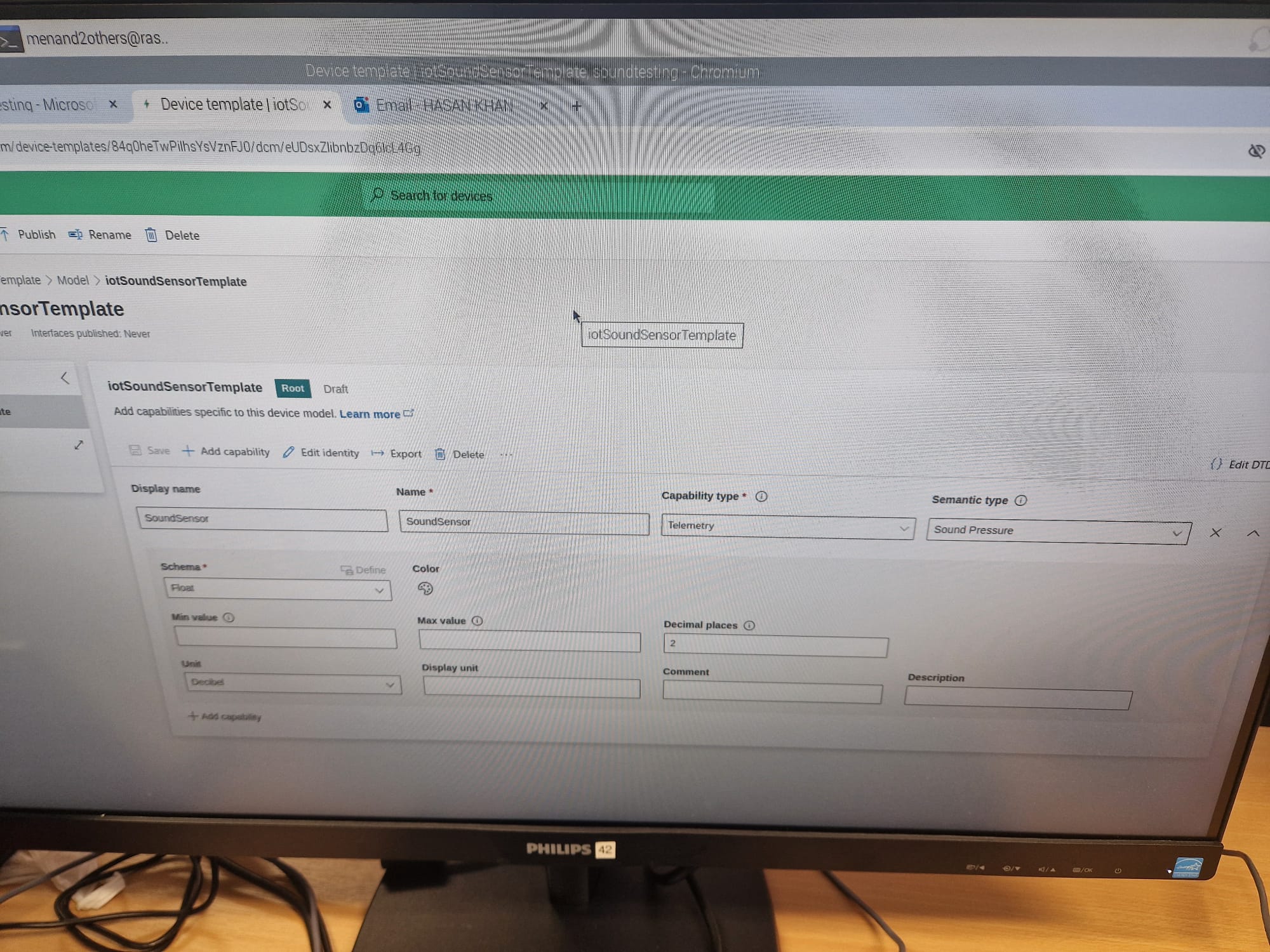


Figure 17: iotSoundSensorTemplate

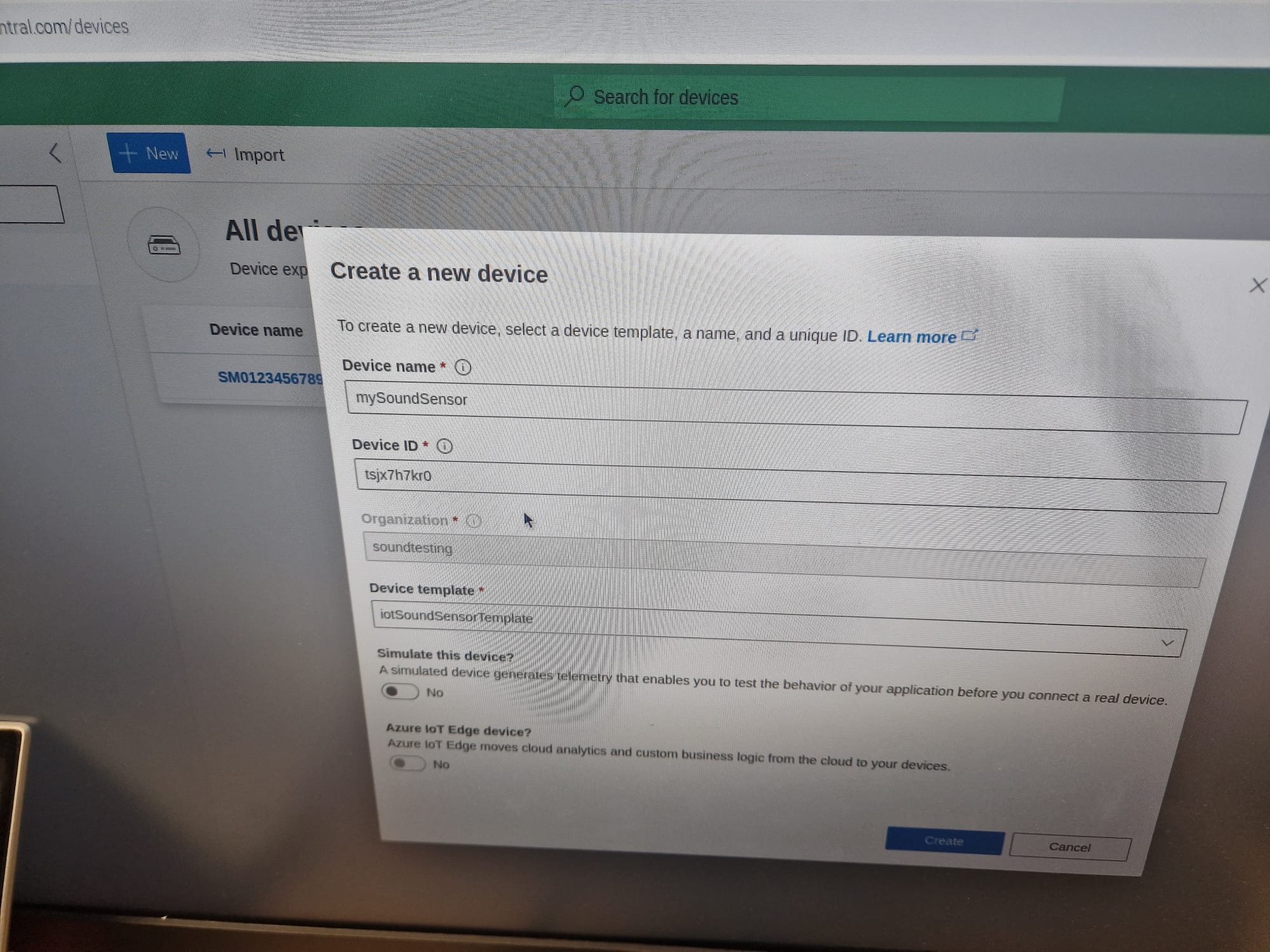


Figure 18: Creation of mySoundSensor Device

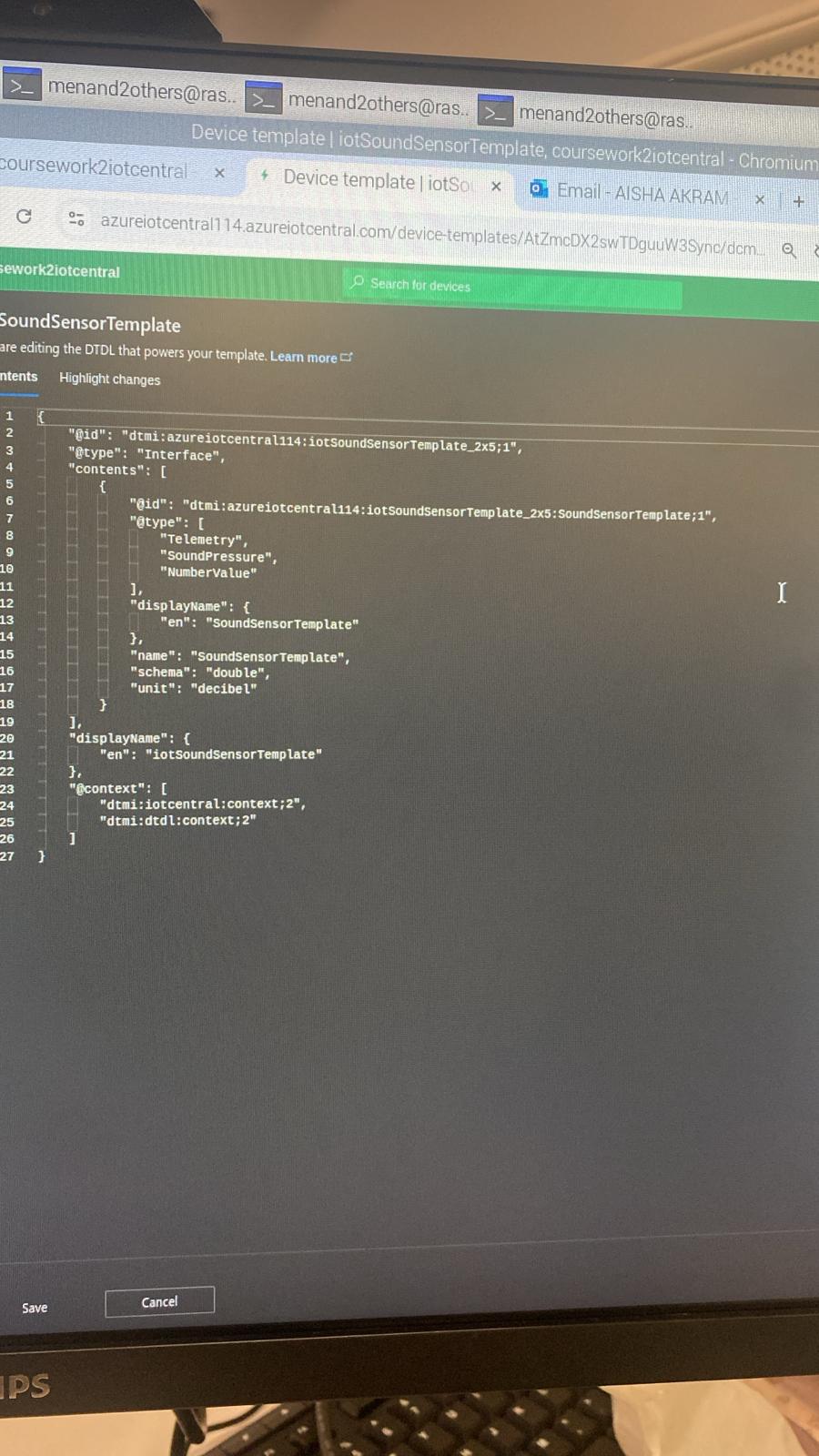


Figure 19: DTDL Info of iotSoundSensorTemplate

## Device Connection Groups

A computer screen with a white box on it

AI-generated content may be incorrect.

Figure 20: Information of Device Connection Groups

# Appendix C: Azure Validation Tools