Cloud Application Development Continual Assessment 2024

Microcontroller Driven Room Temperature Monitoring and Alarm System with AWS Integration

Deirdre Lee T00126583

**Abstract**

The purpose of the work, methods used for research , main conclusions reached

*Keywords: Microcontroller, Monitoring, AWS, Serverless, API, Alarm, Notification*

# **Introduction**

The rationale for undertaking the work

* Explanation of significance and decision-making process.
* Clarification of assignment or self-selection.

Description of report

* Clear statement of focus and purpose.
* Identification of the main question or problem.

Background info

* Overview of existing research and key findings.
* Identification of relevant themes and timeliness.

Approach to responding to the brief

* Description of methodology or approach.
* Explanation of how research addresses the question.

Method of enquiry

* Brief outline of research methodology.
* Explanation of data collection approach.

# **Literature Review**

Material search process

* Explanation of methodologies used to find relevant publications.
* Identification of any notable trends observed during the search.

Group text into themes

* Subheadings categorizing publications based on thematic relevance.
* Clear organization to highlight connections between studies.

Theme sections

* Critical summaries of individual works within each theme.
* Emphasis on relevance to the current research.
* Assessment of strengths, weaknesses, and contributions of each piece.

Conclusion

* Discussion of how the literature review informs the current research.
* Identification of areas to build upon based on the review.
* Recognition of gaps in existing literature and plans to address them.



# **Methodology**

Architecture of application – draw graph

### Develop Sensor Data Collection

I had limited experience working with micro:bit microcontrollers before beginning this project, using the online editor at <https://python.microbit.org/v/3>. However, for this project, I decided to download the Mu Editor to work on the program locally, as suggested in the book ‘*Beginning BBC micro:bit: A Practical Introduction to micro:bit Development’* (Seneviratne, 2018). I created a new file within the application and wrote a basic script to check if the temperature exceeded set thresholds, printing a warning across the microcontroller screen as well as the current temperature, shown in 4.10.1.1.

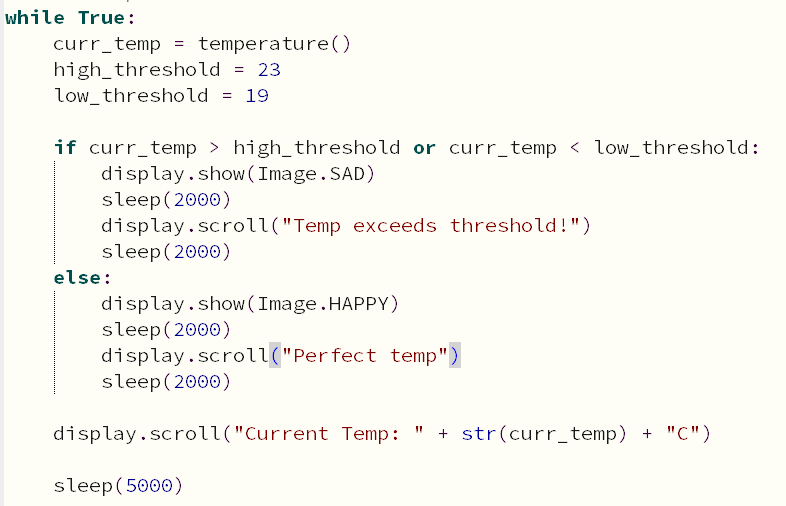


Fig 4.10.1.1 Initial code

I used the flash option of Mu Editor to copy the code onto the microcontroller to observe the program in action and was pleased with the results.

### Transmit Readings to Laptop Device

Exploring Bluetooth as a communication option was considered. However, upon investigation, it was determined to not be viable for the current scope of the project. This decision was influenced by the need to obtain an intermediary device to facilitate communication over Bluetooth Low Energy (BLE) via a ‘bridge layer’. (Corbellini et al., 2017)

The micro:bit, being the primary sensor device, lacks the native capability to transmit data via BLE. While it possesses built-in radio capabilities, it operates on a custom protocol incompatible with BLE. To enable BLE communication, an intermediary device is essential.

One feasible option involves using a Raspberry Pi, which inherently supports Bluetooth Low Energy (BLE). Acting as the bridge, the Raspberry Pi would receive data from the micro:bit via its radio module, process it, and transmit it over BLE to the receiving device.

Given these considerations, the decision was made to rely on an alternative approach that involves using a USB cable to transmit data directly to a laptop. In this setup, the micro:bit code would need to be updated to log readings at specified intervals and trigger events when threshold exceedances occur. (microbit, 2022)

The temperature logs were stored in the memory on the micro:bit. To ensure continuous logging, I developed a Python script that imported the pyserial library to establish communication with the micro:bit via the serial port. (Liechti, 2020) This script read the temperature data from the micro:bit and transferred it to a file on the laptop for storage and further analysis. I was able to access the micro:bit through serial port COM3. (microbit help & support, 2021) Initially, I tested this using PuTTY and then proceeded to write a Python script to access the port. For the information to be read, I updated the micro:bit code to print the readings that were being logged. This printed data was then received through the program. The initial code is shown in Fig 4.8.2.1 and sample output in Fig 4.8.2.2.

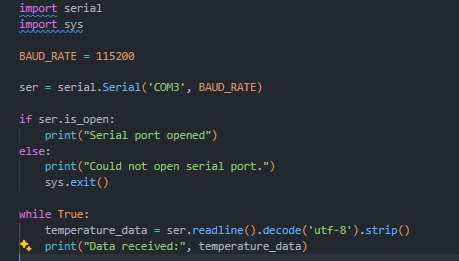


Fig 4.8.2.1 Initial receiver Code

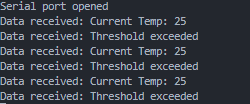


Fig 4.8.2.2 Sample output

### Set up AWS Identity & Access Management (IAM)

As I was creating the services needed I ensured the necessary permissions were assigned by creating policies and assigning them to the required roles. Throughout the description of steps taken going forward, I include to which services this was implemented.

### Create AWS Lambda Function

After completing the previous steps of creating the initial microcontroller program and setting up communication with the receiving laptop it was necessary to begin the cloud application process.

My initial plan for the cloud implementation had included utilising IoT Core to manage communication with the IoT device. However, since the device is not internet-enabled and needs to communicate through the Python receiver program, as a result of the research and exploration in [4.10.2 Transmit Readings to Laptop Device,](#_Transmit_Readings_to) this service is not suitable for the proof of concept version.

The receiver program I created will need to communicate with the cloud and to do this I created a Lambda function to receive the data through a HTTP endpoint. This also necessitated the creation of an API Gateway service.

I created the following basic Lambda function, shown in Fig 4.8.9.1, to receive a HTTP request. It receives data and prints it, as well as returning a status OK code.

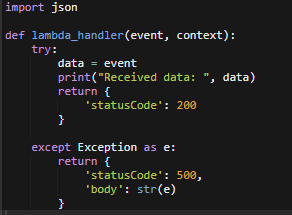


Fig 4.8.9.1 HttpRequestHandlerLambda

Once testing is complete the function will be updated as necessary to further the application.

### Set up API Gateway

For the lambda function created in [4.10.9 Create AWS Lambda Function](#_Create_AWS_Lambda), it was necessary to set up an API Gateway service and add it as a trigger to the Lambda function. This ensures that when data is sent to the endpoint, the Lambda function will be triggered. I created a REST API gateway with a POST endpoint and added it as a trigger to the Lambda function. Upon testing, I received a successful response as shown in Fig 4.8.10.1.

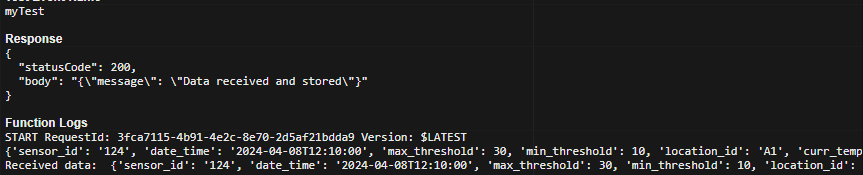


Fig 4.8.10.1 Lambda test response

### Set up Amazon Simple Notification Service (SNS)

Before creating the next Lambda function, which will send an email notification in the case of threshold exceedance using SNS, I opted to first create the SNS topic and subscription. I created a topic called TempThresholdAlarm which by default can only be published and received by the topic owner. This can be updated but I felt this was sufficient for a POC. I then proceeded to create a subscription, using an email protocol, and defining my college email to be the subscriber. (Amazon AWS, 2024)A subscription confirmation email was sent to my email address, to which I was able to subscribe, the confirmation is shown in Fig 4.8.11.1.

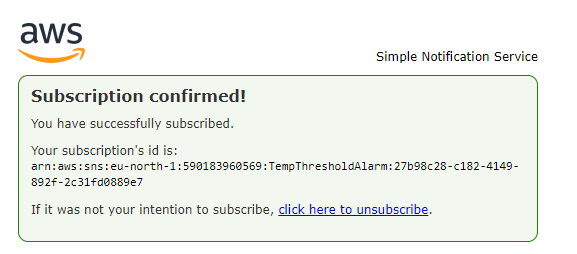


Fig 4.8.11.1 Subscription confirmation

The next step was to create a Lambda function which will be called from the previous Lambda if the is\_exceeded variable is set to ‘True’. After creating the function I added a new IAM policy to allow the Lambda function permission to publish SNS. (AWS Knowledge Center, 2024) I updated the HttpRequestHandlerLambda function to invoke the ThresholdExceededAlarm function, passing the current temperature, location, and threshold variables. These variables are then included in the updated message sent to the subscriber. This was achieved by using Amazon SNS to trigger the second Lambda function. (Amazon AWS, 2024) I created a new topic called ‘’NotifyAlarmLambda’ and set up a subscription with a Lambda protocol, which trigger the ThresholdExceededAlarm function from the HttpRequestHandlerLambda endpoint. I then updated the receiving function, ThresholdExceededAlarm, to parse the incoming message and perform the SNS actions Additionally, I ensured that both Lambda functions had the necessary IAM roles to perform the required actions. The HttpRequestHandlerLambda function, when invoked, publishes to the SNS topic NotifyAlarmLambda, providing the necessary variables curr\_temp, location, and threshold. The ThresholdExceededAlarm, subscribed to the NotifyAlarmLambda topic, parses the incoming message details and incorporates them into a message published to the TempThresholdAlarm topic. This message is distributed to all subscribers, which currently only includes my college email, as part of this proof of concept. The resulting notification email is shown in Fig 4.8.11.3.

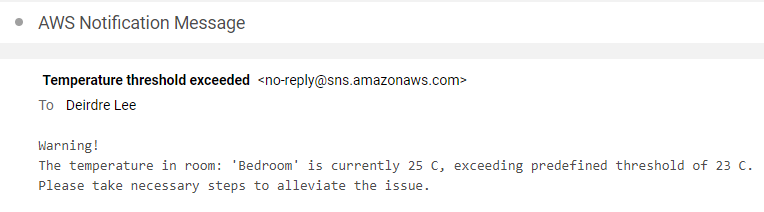


Fig 4.8.11.3 Notification email for temp exceeded warning

Overall, the SNS topics and subscriptions were easy to manage and updateable to include multiple subscribers as the application grows.

### Configure Amazon DynamoDB

#### Database Plan for Storing Sensor Readings

DynamoDB will be used for storing the temperature readings. DynamoDB is a NoSQL database that consists of a key-value data store. I chose DynamoDB due to its ability to handle large quantities of data and scale efficiently. It is also suitable for real-time processing, which will be crucial for this application. Users will need to be able to check historical data, and there will be continuous write operations as the temperature data is stored.

Currently, the data being collected includes the temperature reading in degrees Celsius to the nearest whole number. It also records when the threshold is exceeded, as well as holding the value of the upper and lower temperature thresholds, which are values intended to be changed by the user.

Additionally, the schema needs to include the following information: Sensor ID, a unique identifier for each sensor device; Location, details about the location of each sensor device (such as room name); Time of Reading, a timestamp indicating when each temperature reading was recorded; and Exceeded Threshold, a flag indicating whether the temperature reading exceeded the predefined upper or lower thresholds. There is only one sensor device being used for this proof of concept application, but as part of the non-functional requirement of scalability, the schema needs to reflect the fact that multiple sensors may be present.

Considering all this, I propose the following schema.

|  |  |
| --- | --- |
| TableName | MicroSensor |
| SensorID | String (PK - Partition Key) |
| Location | String |
| TimeDate | String (PK - Sort Key) |
| Temperature | Number |
| Upper Threshold | Number |
| Lower Threshold | Number |
| Exceeded Threshold | Boolean |

I decided to use a composite primary key for the table. For the partition key, I chose the sensor ID. This decision was made on the need for scalability, as I expect to deploy multiple sensors beyond the proof of concept phase. By distributing data across multiple partitions based on sensor ID, DynamoDB can efficiently handle an increase in data or workload. (Amazon AWS, 2024)

I chose the timestamp as the sort key. This choice aligns with DynamoDB's design principles and helps with efficient querying, especially for accessing historical data. (Amazon AWS, 2024) By organising data based on the event time, the table supports optimised data retrieval and storage.

I updated the microcontroller code to pass the required variables to the receiver program. Despite spending a lot of time researching and testing, I was unable to pass the time data from the microcontroller. Instead, a variable is initialised on the receiver side, which is then stored when the other variables are stored.

As part of the debugging process, the receiver creates a text file to log the data being received. A snippet of the updated receiver code, including the current data storage implementation, is shown in Fig 4.8.12.1.1.

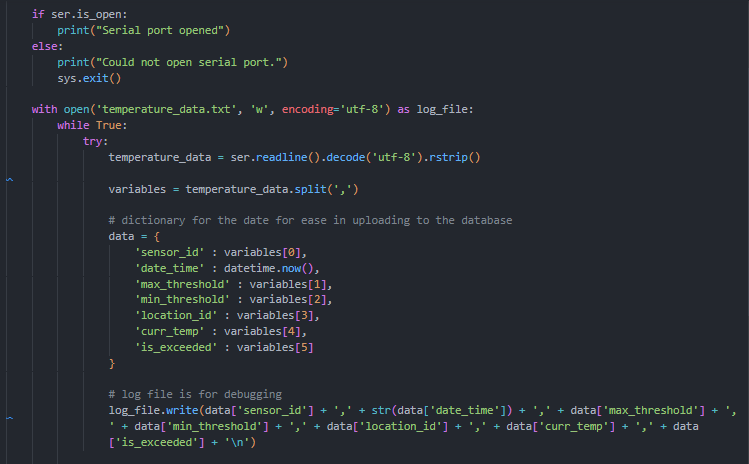


Fig 4.8.12.1.1 Receiver Code

The log output is shown Fig 4.8.12.1.2.

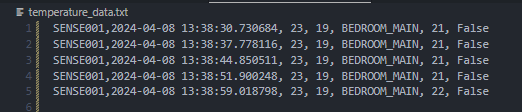


Fig 4.8.12.1.2 Log output

Finally, the updated microcontroller code is shown in Fig 4.8.12.1.3.

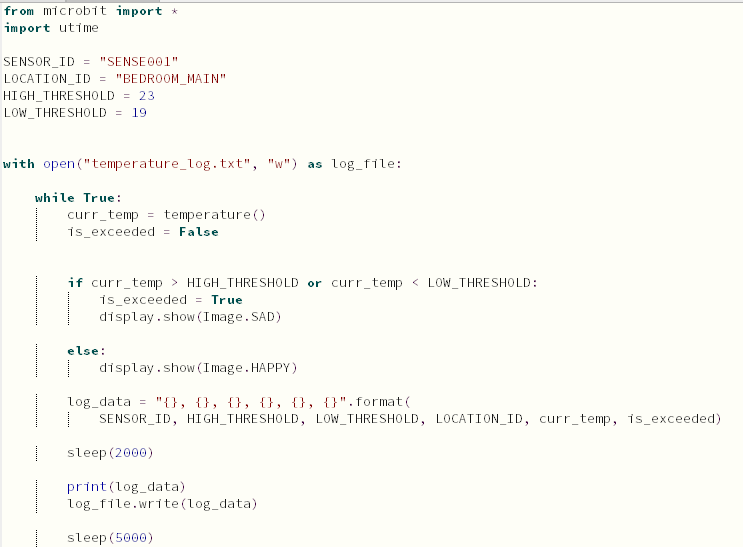


Fig 4.8.12.1.3 Microcontroller code

#### Set up Storage Solution with DynamoDB

The next step in the project was to setup a DynamoDB table to store the data received by the Lambda function via the API Gateway. I created a table named MicroSensor, setting SensorID as the partition key and DateTime as the sort key. Then I created an IAM policy called DynamoDBReadWrite to allow permissions to the lambda function to read and write to the table. After creating the policy, I attached it to the HttpRequestHandlerLambda role.

Once the IAM configuration was completed, I updated the Lambda function to write to the DynamoDB table as shown in Fig 4.10.12.2.1. (Amazon AWS, 2024)

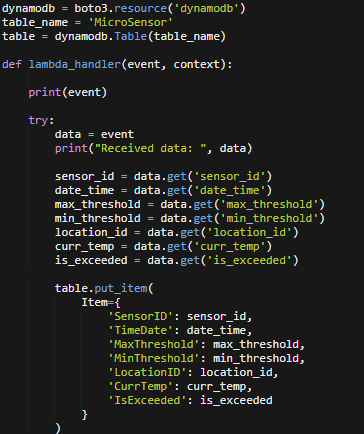


Fig 4.10.12.2.1 HttpRequestHandlerLambda

Upon testing, it was observed that the data is being processed and stored correctly using the Lambda test function. The test data stored in DynamoDB can be seen in Fig 4.10.12.2.2.

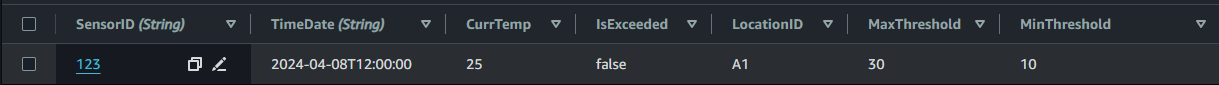


Fig 4.10.12.2.2 Test data

### Set Up Amazon S3

{ automatically done from amplify }

### Set up AWS Amplify

Created role to allow amplify backend dev to access aws resources

Connected to github repository

Followed instructions on link…

Used CloudFormation via Amplify to log and view errors

Having issues with permission, review and try gain

# 

# **Results/Data/Findings**

# **Discussion**

# **Conclusion**

# **References**

Amazon AWS, 2024. *Best practices for designing and using partition keys effectively.* [Online]   
Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html  
[Accessed 5 April 2024].

Amazon AWS, 2024. *Best practices for using sort keys to organize data.* [Online]   
Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-sort-keys.html  
[Accessed 5 April 2024].

Amazon AWS, 2024. *Fanout to Lambda functions.* [Online]   
Available at: https://docs.aws.amazon.com/sns/latest/dg/sns-lambda-as-subscriber.html  
[Accessed 9 April 2024].

Amazon AWS, 2024. *Getting started with Amazon SNS.* [Online]   
Available at: https://docs.aws.amazon.com/sns/latest/dg/sns-getting-started.html  
[Accessed 9 April 2024].

Amazon AWS, 2024. *Tutorial: Build a CRUD API with Lambda and DynamoDB.* [Online]   
Available at: https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html  
[Accessed 8 April 2024].

Anon., n.d. [Online].

AWS Knowledge Center, 2024. *How can I publish a message to an Amazon SNS topic using a Lambda function?.* [Online]   
Available at: https://repost.aws/knowledge-center/sns-topic-lambda  
[Accessed 9 April 2024].

Golovnia, S., 2022. *15 NON-FUNCTIONAL REQUIREMENTS IN SOFTWARE ENGINEERING [2024].* [Online]   
Available at: https://www.sfappworks.com/blogs/non-functional-requirements-in-software-engineering  
[Accessed 1 April 2024].

Liechti, C., 2020. *pyserial.* [Online]   
Available at: https://pyserial.readthedocs.io/en/latest/shortintro.html#opening-serial-ports  
[Accessed 3 April 2024].

microbit help & support, 2021. *Outputing serial data from the micro:bit to a computer.* [Online]   
Available at: https://support.microbit.org/support/solutions/articles/19000022103-outputing-serial-data-from-the-micro-bit-to-a-computer#:~:text=You%20can%20send%20and%20receive,and%20Makecode%20for%20micro%3Abit.&text=In%20Python%2C%20the%20  
[Accessed 4 April 2024].

microbit, 2022. *Serial.* [Online]   
Available at: https://makecode.microbit.org/device/serial  
[Accessed 4 April 2024].

Nuclino, 2024. *A Guide to Functional Requirements.* [Online]   
Available at: https://www.nuclino.com/articles/functional-requirements  
[Accessed 1 April 2024].

Seneviratne, P., 2018. *Beginning BBC micro:bit: A Practical Introduction to micro:bit Development.* 1st ed. Sri Lanka: Apress.

# References

Amazon AWS, 2024. *Best practices for designing and using partition keys effectively.* [Online]   
Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html  
[Accessed 5 April 2024].

Amazon AWS, 2024. *Best practices for using sort keys to organize data.* [Online]   
Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-sort-keys.html  
[Accessed 5 April 2024].

Amazon AWS, 2024. *Fanout to Lambda functions.* [Online]   
Available at: https://docs.aws.amazon.com/sns/latest/dg/sns-lambda-as-subscriber.html  
[Accessed 9 April 2024].

Amazon AWS, 2024. *Getting started with Amazon SNS.* [Online]   
Available at: https://docs.aws.amazon.com/sns/latest/dg/sns-getting-started.html  
[Accessed 9 April 2024].

Amazon AWS, 2024. *Tutorial: Build a CRUD API with Lambda and DynamoDB.* [Online]   
Available at: https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html  
[Accessed 8 April 2024].

Anon., n.d. [Online].

AWS Knowledge Center, 2024. *How can I publish a message to an Amazon SNS topic using a Lambda function?.* [Online]   
Available at: https://repost.aws/knowledge-center/sns-topic-lambda  
[Accessed 9 April 2024].

Golovnia, S., 2022. *15 NON-FUNCTIONAL REQUIREMENTS IN SOFTWARE ENGINEERING [2024].* [Online]   
Available at: https://www.sfappworks.com/blogs/non-functional-requirements-in-software-engineering  
[Accessed 1 April 2024].

Liechti, C., 2020. *pyserial.* [Online]   
Available at: https://pyserial.readthedocs.io/en/latest/shortintro.html#opening-serial-ports  
[Accessed 3 April 2024].

microbit help & support, 2021. *Outputing serial data from the micro:bit to a computer.* [Online]   
Available at: https://support.microbit.org/support/solutions/articles/19000022103-outputing-serial-data-from-the-micro-bit-to-a-computer#:~:text=You%20can%20send%20and%20receive,and%20Makecode%20for%20micro%3Abit.&text=In%20Python%2C%20the%20  
[Accessed 4 April 2024].

microbit, 2022. *Serial.* [Online]   
Available at: https://makecode.microbit.org/device/serial  
[Accessed 4 April 2024].

Nuclino, 2024. *A Guide to Functional Requirements.* [Online]   
Available at: https://www.nuclino.com/articles/functional-requirements  
[Accessed 1 April 2024].

Seneviratne, P., 2018. *Beginning BBC micro:bit: A Practical Introduction to micro:bit Development.* 1st ed. Sri Lanka: Apress.