Siploma

Your Drink. Just Smarter.

INTERNET OF THINGS: ARCHITECTURES, APPLICATIONS, AND IMPLEMENTATION

Date of submission: April 16, 2025

TEAM MEMBERS:

JANICE UWUJAREN

MAHARSHI PATEL

MISHA HEMANT JOSHI

ROHAN VITTHAL DHAMALE

Team Task-Specific Contributions

Task	Component Weightage	Rohan (%)	Misha (%)	Maharshi (%)	Janice (%)
Hardware Integration	0.25		50	50	
Embedded Code(HW)	0.20		50	50	
Setting up the Cloud components	0.15	100			
Developing UI	0.15				100
Video	0.10	25	25	25	25
Documentation (README files, schematics, etc.)	0.15	25	25	25	25
Per student aggregate contribution		**0.2875**	**0.2875**	**0.2875**	**0.2875**

3. Introduction

In a world driven by personalization and smart living, **Siploma** offers an affordable and intelligent solution for on-demand beverage temperature control. Designed as an **IoT-enabled system**, **Siploma** allows users to heat or cool their drinks with a tap while adapting to ambient conditions for energy-efficient operation.

At its core, the system is built around a **Raspberry Pi**, which uses real-time temperature data from integrated digital sensors to manage a **thermoelectric Peltier module**. The setup connects to **AWS IoT Core**, enabling remote control, live cloud monitoring, and notifications when the beverage reaches the target temperature. Balancing cost-effectiveness and functionality, **Siploma** prioritizes modularity and simplicity, opening doors for future scaling and enhancements in thermal efficiency.

Unlike traditional warmers or high-end smart devices, **Siploma** integrates essential features into a low-cost, modular platform — merging real-time sensing, cloud intelligence, and intuitive control to enhance everyday interactions.

4. Design

The **Siploma system** is divided into two primary domains: **hardware/embedded design** and **cloud-based integration**. This ensures seamless coordination between real-time control mechanisms and smart cloud features.

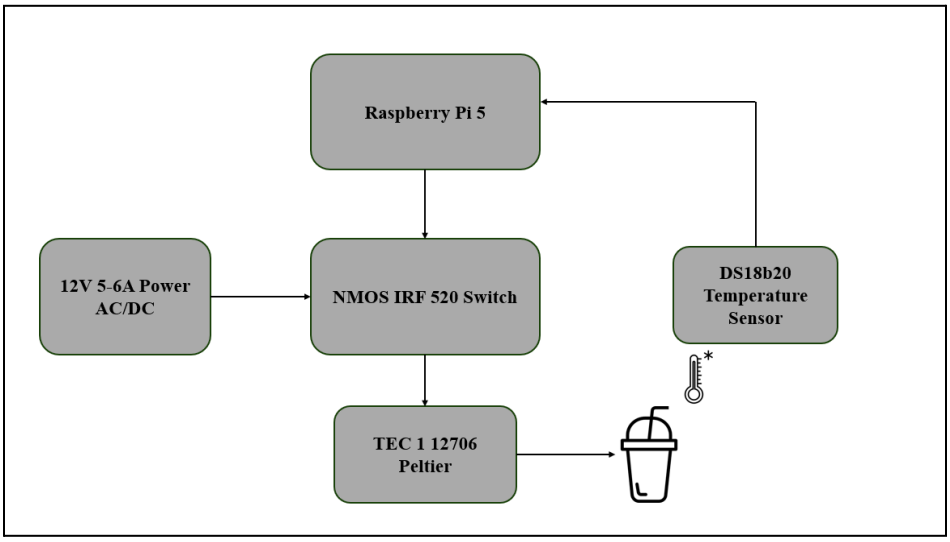
A. Hardware / Embedded Design

Architecture:

The **Siploma hardware** integrates cost-effective, reliable components for intelligent temperature control:

- **TEC1-12706 Peltier Module:** A solid-state thermoelectric cooler that heats or cools depending on current direction, with aluminum heatsinks for efficient thermal management.
- **DS18B20 Temperature Sensor:** A digital sensor thermally coupled to the beverage container, providing accurate, real-time temperature readings (9–12-bit resolution) to the **Raspberry Pi** via a 1-Wire interface.
- **Raspberry Pi 5:** Orchestrates control logic, reads sensor data, compares it with the user-set target, and switches the **Peltier module** on or off accordingly.
- **IRF520 NMOS Transistor Module:** Acts as a low-side electronic switch, controlled by a **Raspberry Pi GPIO signal**, regulating the power to the **Peltier module**.

When a user sets a target temperature through the **cloud dashboard**, the value is transmitted via **MQTT protocol** to the **Raspberry Pi**. The system activates the **Peltier module**, initiating heating or cooling. When the **DS18B20** detects the temperature within a specified range (typically **±1°C**) of the target, the system deactivates the **MOSFET**, cutting off power. A notification is sent to the user's dashboard, confirming the beverage is

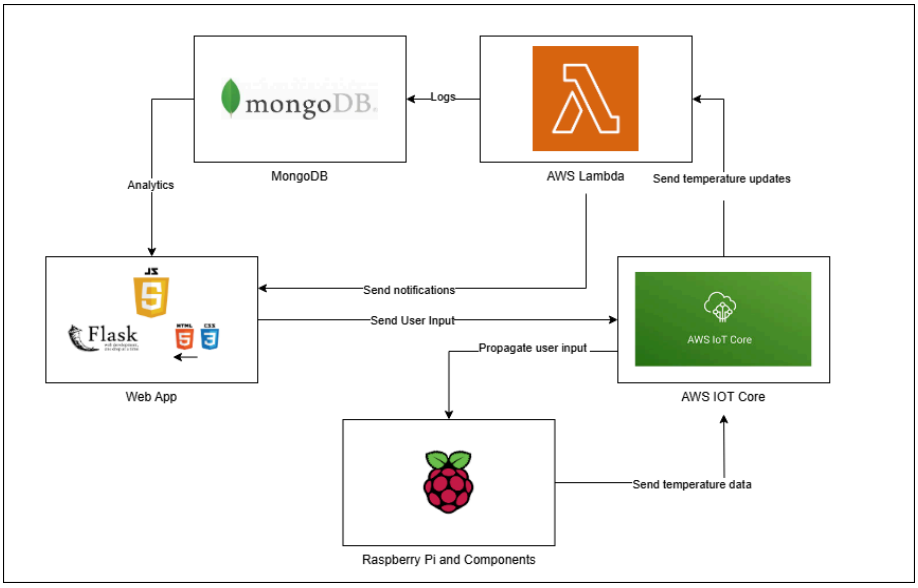


ready. The system automatically monitors and re-engages the control cycle if the temperature drifts.

Key Hardware Specifications:

- **Raspberry Pi 5:** 64-bit quad-core 2.4GHz processor, dual 4K HDMI, 802.11ac Wi-Fi, Bluetooth 5.0, USB 3.0, and a 40-pin GPIO header.
- **IRF520 NMOS Switch:** 100V Vds, 9.2A continuous drain current, low RDS(on), fast switching (30ns rise, 20ns fall).
- **DS18B20 Sensor:** ±0.5°C accuracy (-10°C to +85°C), 1-Wire communication, programmable 9-12 bit resolution.
- **TEC1-12706 Peltier:** 14.4V max, 6.4A, 50W cooling power, **ΔTmax** of 66°C, compact 40mm x 40mm module.

B.



Software System Architecture

The system is composed of the following major blocks:

- **User Interface (Web/Mobile App):** Allows the user to set the target temperature.
- **AWS IoT Core:** Acts as the cloud message broker to route messages securely between the app and the Raspberry Pi.
- **AWS Lambda:** Receives event triggers from AWS IoT, after the user sets a temperature from the web app and also when Raspberry Pi takes an action leading to a message being published. The Lambda function then stores the data to the MongoDB database.
- **MongoDB Database:** Stores temperature readings and timestamps for historical analysis.

4.Implementation

Programming Languages: Python (for Raspberry Pi and Lambda), JavaScript (React for frontend)

Tools Used: AWS IoT Core, AWS Lambda, MongoDB Atlas, HTML, CSS, Python libraries (paho-mqtt, RPi.GPIO, Flask)

Source-code files:
hardware_code/start.py – Raspberry Pi control loop for reading sensors and controlling the Peltier module. It is also responsible for subscribing and publishing messages to the AWS IOT Core MQTT broker. It subscribes to the input topic and controls the Pelier accordingly, along with sending the performance data to the performance topic.

lambda/userInputDataToDatabase/userInputDataToDatabase.py – AWS Lambda function triggered by IoT messages, logs data to MongoDB. The logs are filtered according to the topic and content to decide the collection to be sent to.
dashboard/src/config.py – Contains all the configuration data required to run the web app dashboard.

dashboard/app.py – It is the main function for the Flash application which communicates with the database and the broker. It serves all the API calls from the frontend.

dashboard/static/css/styles.css – Stylesheet for the frontend of the web app.

dashboard/static/js/main.js – Javascript function for API calls and user input on the frontend. It also has the code for rendering analytics using “chart.js”.

dashboard/templates/index.html – The html page for rendering the frontend of the web app.

MongoDB Atlas Configuration:
The database was hosted using MongoDB Atlas. To allow Lambda function access, the IP address of the NAT Gateway (used inside the VPC attached to the Lambda) should be whitelisted in MongoDB's Network Access settings.For our project we stuck

with wide access as static IP address allocation incurs costs. Proper security and role-based access were enforced.

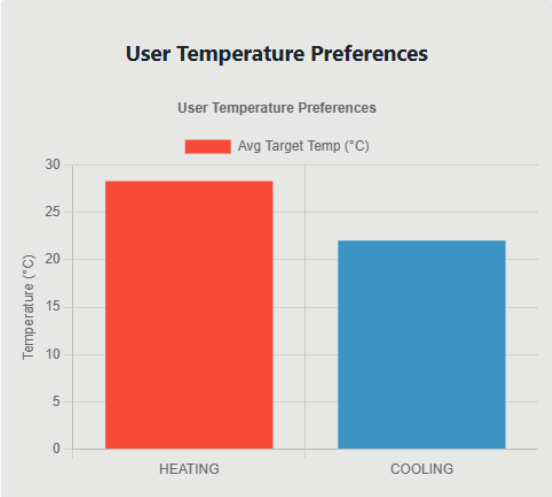
AWS Certificates and Policies: AWS IoT Core was configured with X.509 certificates for mutual TLS authentication. IAM roles and policies were created with the principle of least privilege. Policies were scoped by both action and resource, allowing access only to required MQTT topics and specific AWS services like IoT Core. Devices authenticated via certificate-based access, client ID, and role-based permissions controlled publishing, subscribing, and connecting actions.

Flow

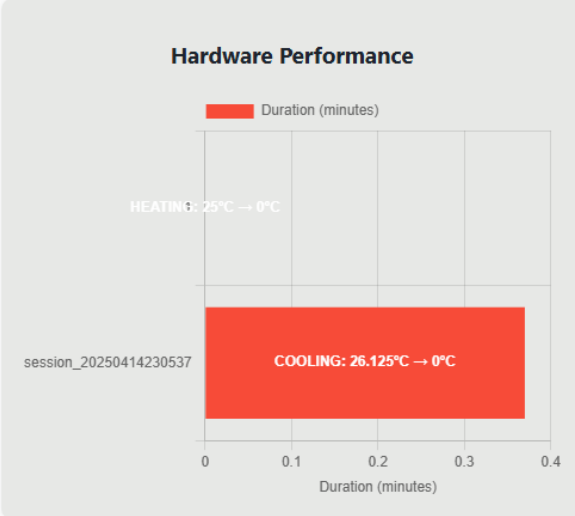
The overall flow of the system begins with the user interacting through a web or mobile application. The user inputs the desired target temperature and submits it, which is then published to an MQTT topic via AWS IoT Core (the broker). This message is received by both the Raspberry Pi and an AWS Lambda function. The Raspberry Pi, upon receiving the command, activates the heating or cooling mechanism accordingly using a Peltier module and continuously monitors the temperature via sensors. As the temperature changes, performance data—such as current temperature and time taken to reach a target—is sent back to the MQTT broker. In parallel, the AWS Lambda function processes incoming messages for logging and analytics. It extracts relevant data from each message and stores it into a MongoDB Atlas database. This data includes timestamps, user preferences, and performance metrics. The frontend application periodically fetches this data from the database to generate visual analytics for the user. The dashboards display trends such as preferred temperature ranges, user input history, and the system's efficiency in reaching desired temperatures.

5. Results and Discussions

The system includes a dashboard that provides insightful analytics based on user interaction and hardware performance. It visualizes user preferences[6.1] by displaying commonly selected hot and cold temperatures. A time-series graph shows the history of user inputs[6.3], helping track usage patterns over time.

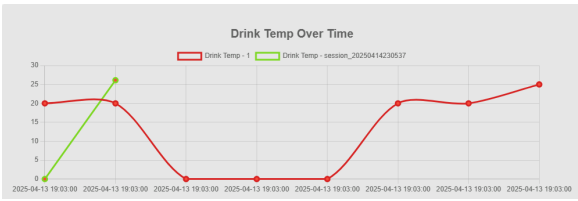


6.1 User Preferences Graph



6.2 Hardware Performance

Another key feature is a performance chart[6.2] that illustrates the efficiency of the heating and cooling process—highlighting the time taken to reach different target temperatures from initial values. These visualizations offer a comprehensive understanding of both user behavior and system responsiveness.



6.3 Drink Temperature Over Time

6. Related Works and References

a. "Getting started with AWS IoT Core", AWS Documentation. https://docs.aws.amazon.com/iot/latest/developerguide/
b. MongoDB Atlas Documentation. https://www.mongodb.com/cloud/atlas
c. Peltier Module TEC1-12706. https://peltiermodules.com/peltier.datasheet/TEC1-12706.pdf
d. DS18B20 Temperature Sensor Overview, https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
e. "Getting started with AWS IoT Core", AWS Documentation. https://docs.aws.amazon.com/iot/latest/developerguide/
f. "AWS IoT Policies and Authorization". https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html
g. "AWS IoT Core MQTT Protocol". https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html
h. "AWS Lambda Developer Guide". https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
i. "Managing Devices with AWS IoT Things". https://docs.aws.amazon.com/iot/latest/developerguide/thing-registry.html