IoT-Based Smart Water Bottle

Presented by:

Maaz Qureshi (qures419)
Joao P. P. G. Marques (pintoga2)
Charles Hsieh (hsiehts4)

Introduction

Context:

- Recently, **health-based** monitoring systems have drawn attention due to their benefits to the overall quality of life.

- These devices have been enabled by the advances in the **IoT technologies**.

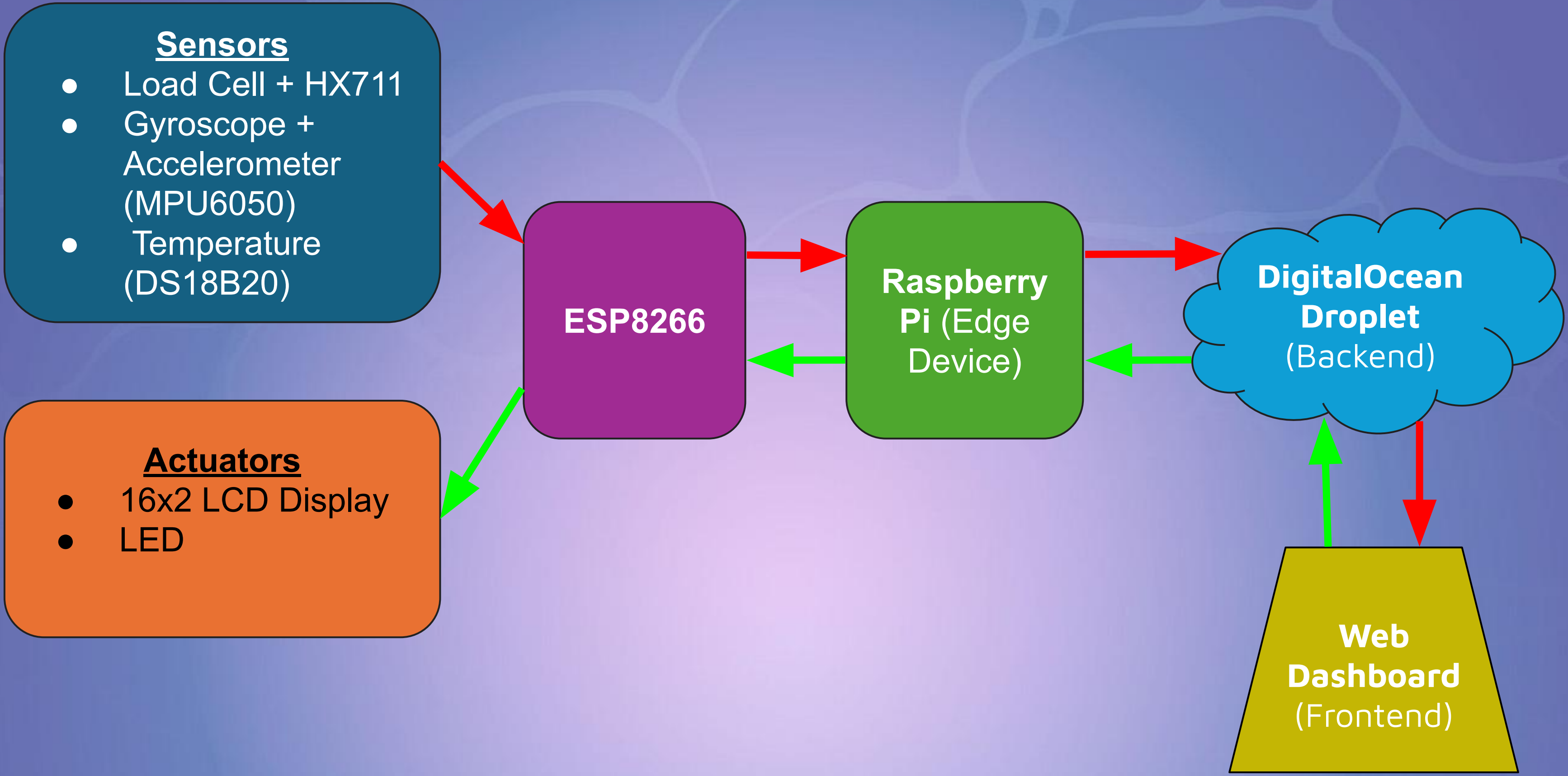
- A critical health component is proper **hydration**.

Main objective: design a smart water bottle that monitors key metrics, notifies the user on spilling risks and can be easily accessed by the user.

High-level System Features

- An **IoT**–based smart water bottle for tracking volume and temperature.
- Low volume and spilling hazard **warning emails**.
- Edge device for interfacing with **multiple** bottles and cloud.
- Fault **tolerant** (adapting to node failures and link instability) and **distributed** (supports deployments at multiple sites).
- Integration between a backend on a **commercial** cloud and a frontend dashboard.

System Structure



Sensors

- Load Cell + HX711
- Gyroscope + Accelerometer (MPU6050)
- Temperature (DS18B20)

ESP8266

Raspberry Pi (Edge Device)

DigitalOcean Droplet (Backend)

Actuators

- 16x2 LCD Display
- LED

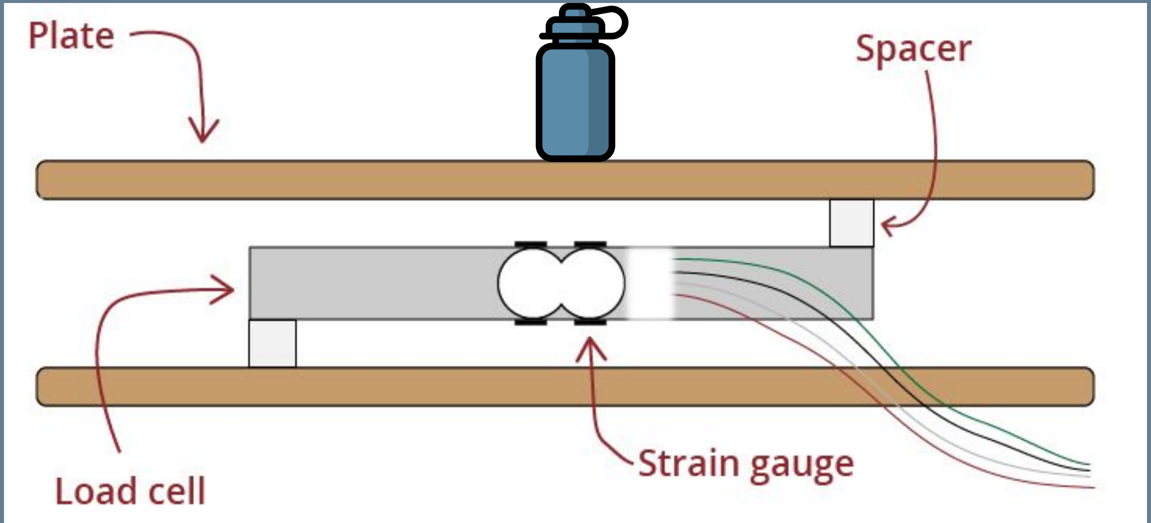
Web Dashboard (Frontend)

01

IoT Devices

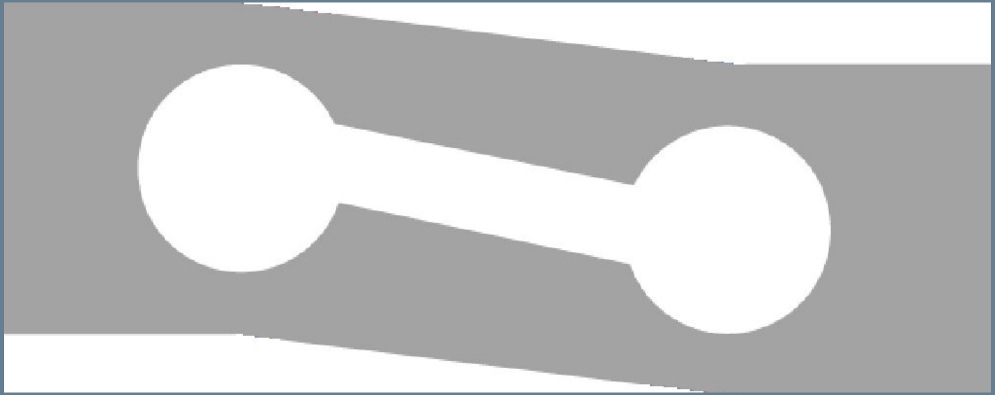
Sensors - Load Cell + HX711 (1/2)

- Load cell is used to measure weight [1]
- Weight is ***<u>converted</u>*** to volume using density of water → 1 gm/mL



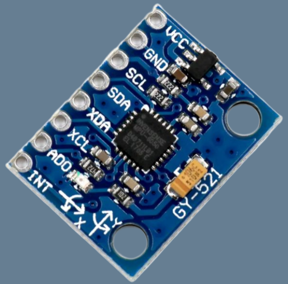
Sensors - Load Cell + HX711 (2/2)

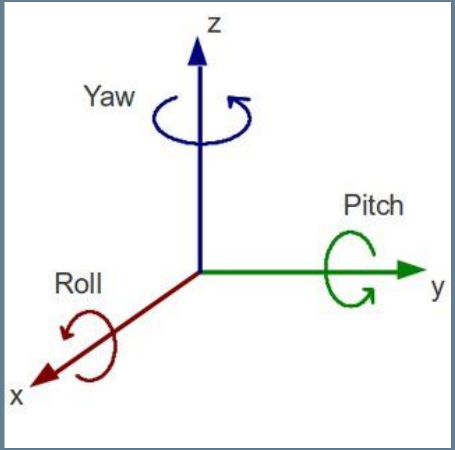
- Load cell deforms when bottle is placed on the plate [1].
- Electric signal from load cell is amplified by HX711.
- ***Calibration is performed*** at ESP8266 to determine the mapping between electric signal to weight (e.g., 215 000 → 500 grams).



Sensors - MPU6050

- 3-axis accelerometer + gyroscope [2].
- Code uses ***Complementary Filter*** to convert readings to angles/tilt (yaw, pitch, & roll) [3].
- Angles can be used to detect potential **water bottle spilling** (when it is upside down).



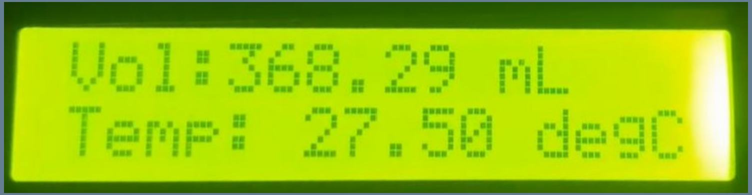


Sensors - DS18B20

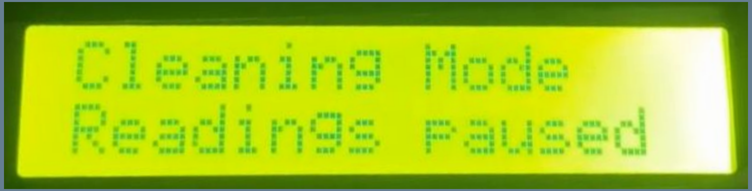
- **Waterproof** temperature sensor.
- Can operate from -55°C to +125°C (accommodate ***both*** hot & cold drinks) [4].

Actuators (1/2)

- Actuators act on **data stored in the cloud**.
- LCD is used to display sensor readings. It displays a special message when **cloud sets** cleaning mode to true.



```
Vol:368.29 mL
Temp: 27.50 degC
```

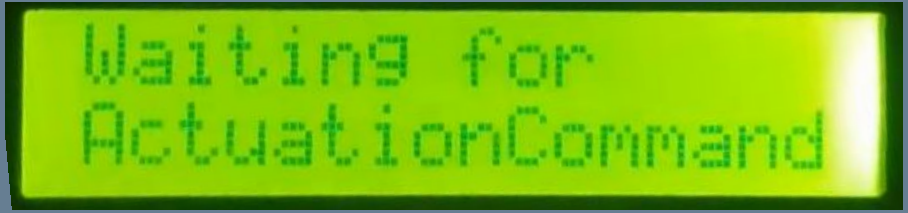


```
Cleaning Mode
Readings paused
```

Actuators (2/2)

- Lights up Red LED if volume fall below threshold set by the **cloud**:

- *Dynamically* **pauses** readings if connection to edge is lost. It continues **probing** the connection while displaying:



```
Waiting for
ActuationCommand
```

02

Edge

Edge: Overview

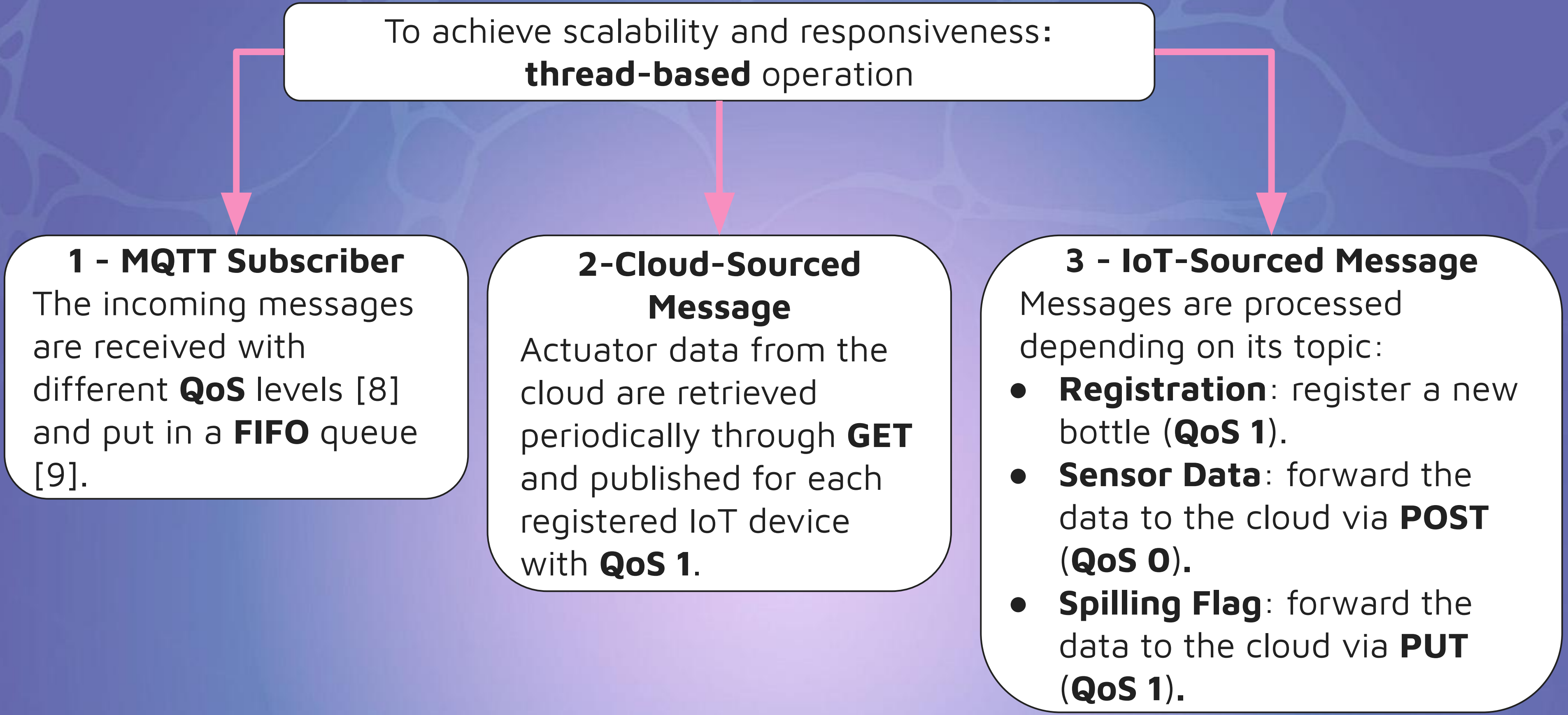
Objective：enable reliable and efficient communications between multiple IoT devices and the cloud through the MQTT protocol [5,6,7].

- Edge must support multiple IoT devices (scalable) in a responsive manner.
- Edge-IoT communication must be fault tolerant.

Challenges:

- How the edge can operate multiple crucial procedures simultaneously?
- How to deal with the inherent instability of the wireless medium?

Edge: Key Operations



To achieve scalability and responsiveness: **thread-based** operation

1 - MQTT Subscriber

The incoming messages are received with different **QoS** levels [8] and put in a **FIFO** queue [9].

2-Cloud-Sourced Message

Actuator data from the cloud are retrieved periodically through **GET** and published for each registered IoT device with **QoS 1**.

3 - IoT-Sourced Message

Messages are processed depending on its topic:

- **Registration**: register a new bottle (**QoS 1**).
- **Sensor Data**: forward the data to the cloud via **POST** (**QoS 0**).
- **Spilling Flag**: forward the data to the cloud via **PUT** (**QoS 1**).

Edge: Additional Features

- **Sequence numbers** ensure **orderly** processing of messages.
- **Fault Tolerance** where the edge monitors incoming messages for each registered bottle. If no message is received within a period, the bottle is deregistered.

03

Cloud

Backend

- Deployed on a **commercial** cloud - **DigitalOcean.**

- Written in Node.js Express with API routes (GET, POST, PUT) for sensor readings, volume thresholds, cleaning mode, and spilling hazards.

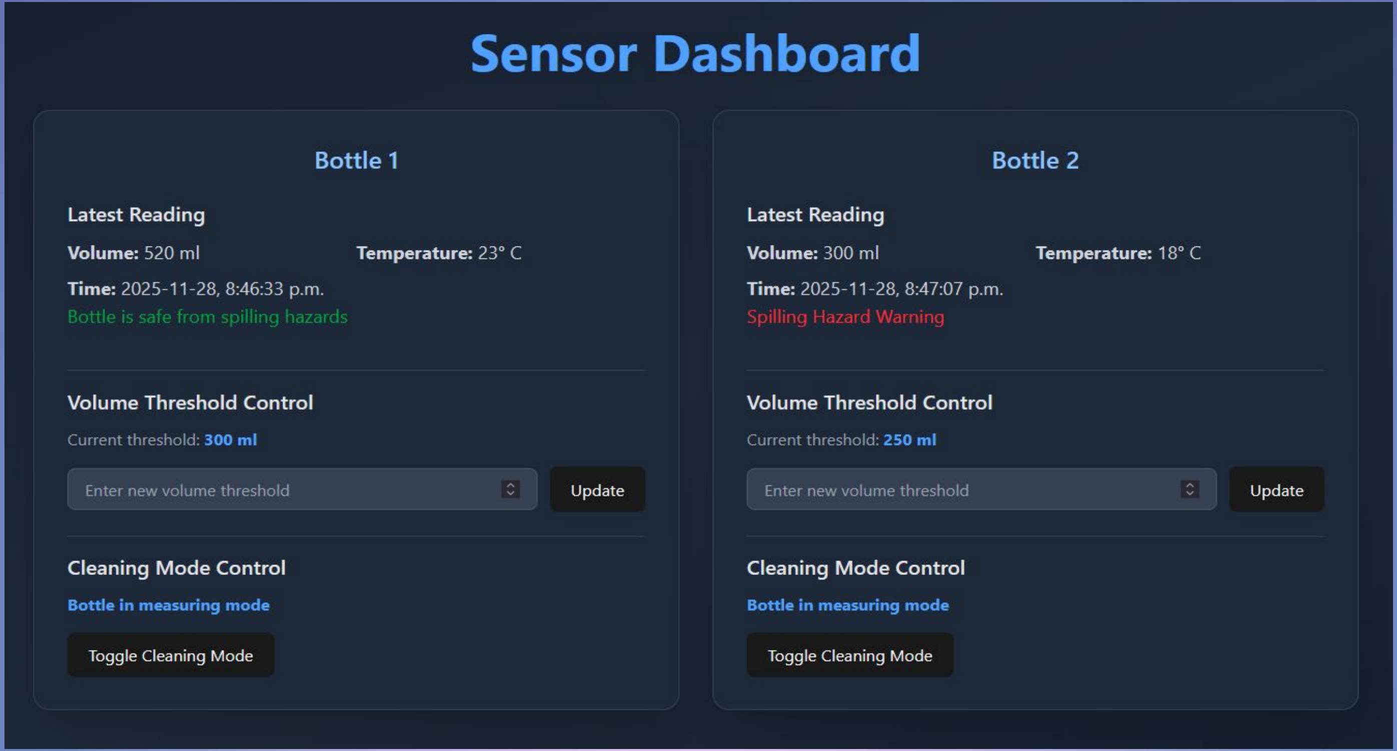
- Multi-container architecture using Docker → packages the API and a PostgreSQL database.

- Database is mounted on a volume to ensure persistent storage on container restarts and server failures.

UI to Display Cloud Data

- Written in React and interfaces with deployed backend’s public IP.
- Displays latest sensor readings and spilling hazard warnings.
- Allows for toggling of cleaning mode and changing of volume threshold.

User Interface



Sensor Dashboard

Bottle 1

Latest Reading

Volume: 520 ml **Temperature:** 23° C

Time: 2025-11-28, 8:46:33 p.m.

Bottle is safe from spilling hazards

Volume Threshold Control

Current threshold: **300 ml**

Cleaning Mode Control

Bottle in measuring mode

Bottle 2

Latest Reading

Volume: 300 ml **Temperature:** 18° C

Time: 2025-11-28, 8:47:07 p.m.

Spilling Hazard Warning

Volume Threshold Control

Current threshold: **250 ml**

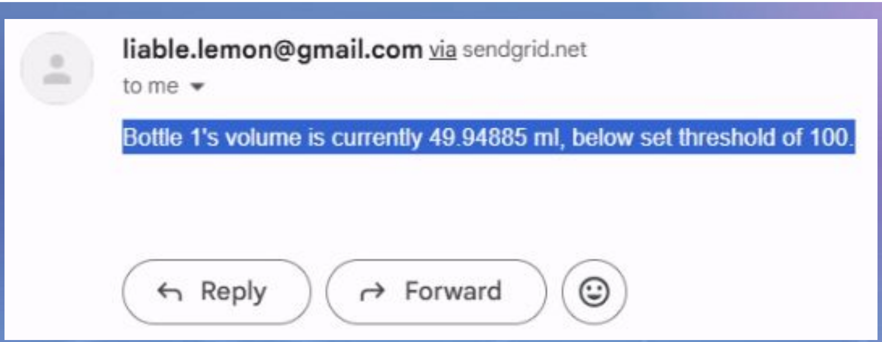
Cleaning Mode Control

Bottle in measuring mode

Email Alerts

Backend interfaces with **SendGrid** API to send alert emails. Email sent when:

- Volume is below the user set threshold
- Risk of spilling



liable.lemon@gmail.com via sendgrid.net

to me

Bottle 1's volume is currently 49.94885 ml, below set threshold of 100.

Conclusion

IoT–based smart water **bottle** for tracking volume and temperature.

The system was designed focusing on:

- **Robustness and Energy Efficiency**: QoS-specific communication as needed;
- **Scalability/Responsiveness**: support for multiple devices and prompt reaction;
- **Fault Tolerance**: consideration of instabilities of nodes and communication;
- **User-friendliness**: straightforward user interface.

Future Works:

- WebSockets for instantaneous frontend updates;
- Sleep/standby mode for ESP for energy saving;
- Water consumption tracking over time across multiple bottles.

References

[1] S. Santos, "ESP8266 NodeMCU with Load Cell and HX711 Amplifier (Digital Scale)," *Random Nerd Tutorials*, Apr. 22, 2022. https://randomnerdtutorials.com/esp8266-load-cell-hx711/ (accessed Dec. 01, 2025).

[2] S. Santos, "ESP8266 NodeMCU MPU-6050 Accelerometer and Gyroscope," *Random Nerd Tutorials*, Jan. 13, 2021. https://randomnerdtutorials.com/esp8266-nodemcu-mpu-6050-accelerometer-gyroscope-arduino/ (accessed Dec. 01, 2025).

[3] R. Fetick, "MPU6050_light," *GitHub*, Jul. 28, 2025. https://github.com/rfetick/MPU6050_light (accessed Dec. 01, 2025).

[4] R. Santos, "ESP8266 DS18B20 Sensor," *Random Nerd Tutorials*, Jul. 16, 2019. https://randomnerdtutorials.com/esp8266-ds18b20-temperature-sensor-web-server-with-arduino-ide/ (accessed Dec. 01, 2025).

[5] R. Light, "MQTT Python client library," Apr. 2024. https://pypi.org/project/paho-mqtt/ (accessed on Dec. 01, 2025).

[6] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," *The Journal of Open Source Software*, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265

[7] J. Gaehwiler, "MQTT library for Arduino," Oct. 2024. https://docs.arduino.cc/libraries/mqtt/ (accessed on Dec. 01, 2025).

[8] HiveMQ Team, "What is MQTT Quality of Service (QoS) 0,1, & 2? – MQTT Essentials: Part 6," May 2025. https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ (accessed on Dec. 01, 2025).

[9] Steve's Internet Guide, "Receiving Messages with the Paho MQTT Python Client," Jun. 2021. http://www.steves-internet-guide.com/receiving-messages-mqtt-python-client/ (accessed on Dec. 03, 2025).

THANKS!

Do you have any questions?

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon** and infographics & images by **Freepik**