ECE1528 Term Project Report: IoT-based Smart Water Bottle

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

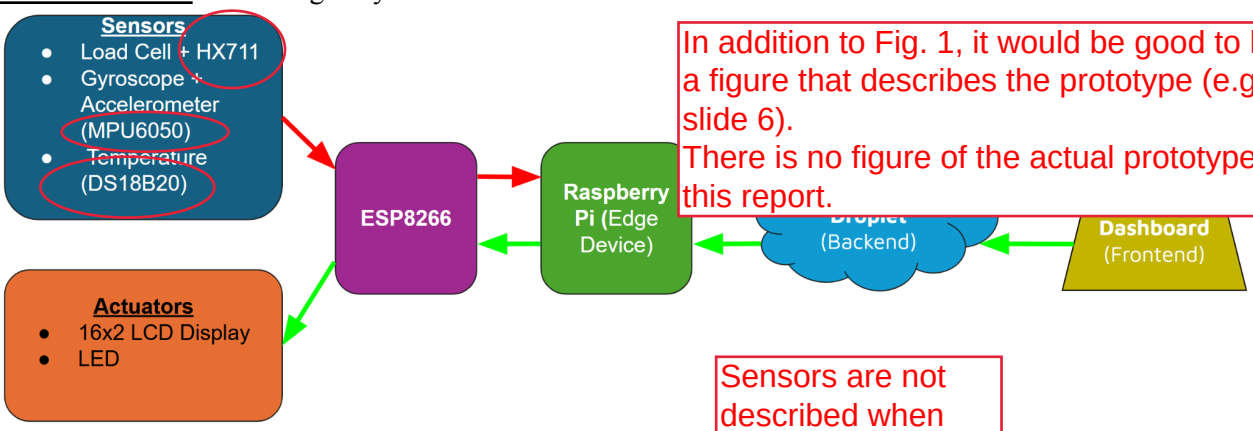
1. Description of the addressed problem and employed approach

Due to the rapid growth of the Internet of Things (IoT), biosensors have accelerated in the domain of eHealth monitoring [1]. At the same time, the modern era demands hectic schedules, where people often put hydration as an afterthought, leading to health deterioration [2]. Considering that two-thirds of the human body is made of water, dehydration is a serious issue [3]. However, most water bottles in the marketplace are basic without any IoT features and even environmentally wasteful (single-use plastic bottles). Combining these realities yields an opportunity to design a smart IoT-based water bottle to improve the health of all individuals.

Our approach to building a smart IoT-based water bottle is to dynamically notify the user via email and red LED whenever the volume of the bottle falls below the user-set threshold. This reminder is critical to ensuring that they keep up with their hydration goals. It offers a web dashboard for the user to adjust the threshold and toggle cleaning mode when the bottle is inactive (not used or being cleaned). It also features a spilling hazard via email notification to minimize the damage of such accidents, making the bottle even smarter. The following section will describe the system in more detail.

2. Description of the system

The structure of our system is shown in Fig. 1. This section starts by describing the IoT device (i.e., water bottle) consisting of an ESP8266 board interfacing with actuators and sensors. Next, we discuss the Raspberry Pi as an edge device that enables bidirectional links between IoT devices and the cloud. Finally, we describe the cloud and the web dashboard. Hereinafter, we use the ***terms bottles, IoT devices and ESP boards*** interchangeably.



In addition to Fig. 1, it would be good to have a figure that describes the prototype (e.g., slide 6).
There is no figure of the actual prototype in this report.

Sensors are not described when first mentioned.

Fig. 1: System structure with red and green arrows show wnstream paths, respectively.

IoT device: The IoT device consists of an ESP board interfacing with sensors and actuators. The sensors attached to the bottle are used to measure its weight, orientation, and temperature. The load cell is responsible for measuring the weight, which is then converted into volume. When the bottle is placed on top of the load cell, the load cell is deformed, creating a strain between the metal bars, as shown in Fig. 2 (left figure) [4]. The load cell requires a spacer and a place to deform properly. An attempt was made to *3D print* the spacer and plate, but it did not work satisfactorily (see Fig. 2, middle figure). Thus, we purchased the plates from Amazon UK. That strain is converted into an electric signal. Since our water bottle is light (weighing less than 5 kg), this electric signal needs to be amplified by HX711 for it to be visible [4]. This amplified signal is unitless, so the ESP8266 runs a separate sketch to convert it into grams. The calibrating sketch involves the user placing an object of known weight (e.g., a caseless phone) and using the formula $Calibration\ Constant = \frac{Measured\ Electric\ Signal}{Known\ Weight\ in\ Grams}$. Once the load cell is calibrated, the density of water can be used to calculate volume:
$Volume\ in\ mL = Weight\ in\ grams \div 1\ gram/mL$. To ensure accurate volume readings, our final

sketch excludes the weight of the bottle container by performing a tare operation in the setup function. However, a limitation with the load cell is that it must be placed upright and still for the weight to be accurate. If the user puts force either by pressing or lifting the bottle, the readings become inaccurate.

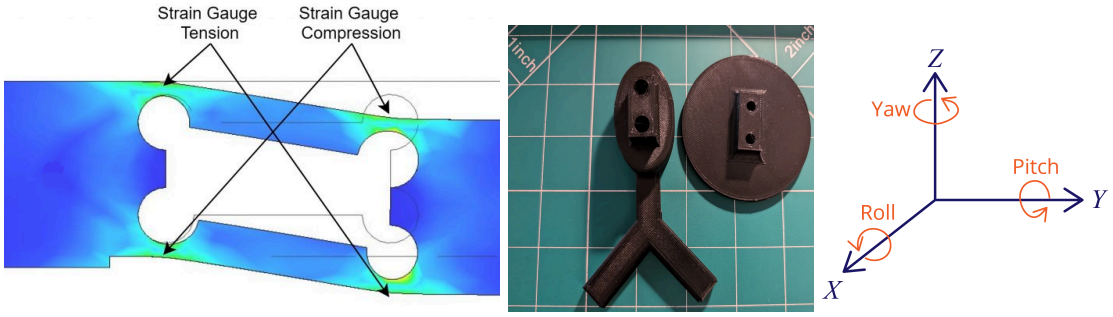


Fig. 2: Left figure: Illustration of the compression that happens when the bottle is placed on top of the load cell.
Middle figure: tentative 3D printing of plates & spacer for the load cell.
Right figure: Illustration of the yaw, pitch, and roll angles that are used to determine the orientation of the bottle.

The MPU6050 is used to measure the orientation or tilt of the bottle to detect potential spilling hazards. The MPU6050 is a 3-axis accelerometer and gyroscope capable of measuring the gravitational acceleration and rotational velocity [5]. The angles (yaw, pitch, and roll) can be calculated using the integration of the angular speed [6]. However, the integration accumulates sensor errors, causing the reading to drift away over time. This can be corrected using gravitational acceleration [6]. Since the MPU6050 detects gravity (9.8 m/s^2) on the Z-axis, the projection of this gravity on the MPU6050 gives another estimation of the angles [6]. Then, the complementary filter uses both values to calculate the final orientation, giving 0.98 weight to the gyroscope data and 0.02 to the accelerometer. Once the angles are calculated, based on Fig. 2 (right figure), a bottle is upside down when the row or pitch is 180° ± 20°. This is when the bottle is flagged for a potential spilling hazard, and an email is sent.

The DS18B20 is a waterproof temperature sensor that measures the temperature of the water. The waterproof feature means that it can be placed inside the bottle to obtain accurate readings [7]. It also operates from -55℃ to +125℃, so the user can pour cold or hot water without damaging the sensor [7].

The 16x2 LCD display from the Lab Kit is used as an actuator. It operates in 3 states, which are determined by the cloud. It displays “Waiting for ActuationCommand” whenever the ESP is disconnected from the edge device. In this state, all sensors are paused until the edge establishes the connection with an actuation command. After the connection is established, there are two modes of operation determined by the cloud. When the cloud turns on cleaning mode, all the sensors are paused, and the LCD displays “Cleaning Mode Readings Paused”. When the cloud turns it off, the LCD will instead display the volume and temperature of the bottle. These states are shown in Fig. 3.

Fig. 3: Illustration of the different LCD states based on actuation commands sent by the cloud.

The final actuator is a red LED from the lab kit. It lights up whenever the volume of the bottle falls below the threshold set by the cloud. The user can set the threshold using the web dashboard. Note that in cleaning mode, the LED will be frozen. So, if the LED was on prior to cleaning mode and the cleaning mode is toggled, it will remain on even if additional water is poured into the bottle.

Edge: We use a Raspberry Pi as the edge device. The edge plays a key role in the system by enabling efficient and reliable bidirectional data flows between the cloud and possibly multiple IoT devices. The edge communicates with the cloud via HTTP methods, while the communication with the IoT devices

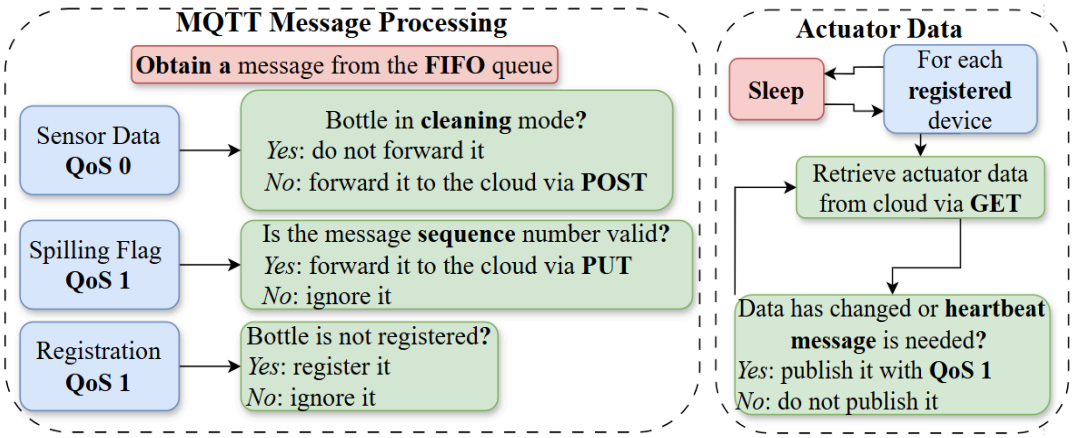
Motivation for using different QoS levels in MQTT is not clear.

occurs through the MQTT protocol [8-10] and WiFi. For that, the Raspberry Pi runs an MQTT broker and client. The exchanged messages are JSON-formatted.

The edge operation relies on a thread-based paradigm to achieve scalability and responsiveness. The edge only processes messages from the IoT devices that are registered, meaning that the first edge-IoT exchange must be initiated by the ESP board with a registration message. We describe in the following the operation of each thread, including how bottles are registered:

- **MQTT Subscriber:** In this thread, the incoming messages from the MQTT broker are received and put in a first-in-first-out thread-safe queue, similar to [11]. The messages come from the topics the edge has subscribed to, which are described subsequently.
- **IoT-Sourced Message Processor:** This thread processes the messages that were put in the queue by the MQTT subscriber thread. The messages are processed depending on their topics. There are three topics, and each of them uses a different Quality of Service (QoS) [12]:
  - **QoS 1 - Registration**: the edge registers the bottle.
  - **QoS 0 - Sensor Data** (temperature and volume): the edge sends the data to the cloud via POST methods.
  - **QoS 1 - Spilling Flag**: the edge forwards the data to the cloud via PUT methods. These messages also carry the sequence number of the message for the bottle, ensuring the orderly processing of the messages at the edge. The spilling flag processing out of order could disrupt the system operation since the edge could send to the cloud a false spilling hazard.

  We set the QoS of the spilling flag and registration topics to one, ensuring that the messages are delivered at least once. In this QoS level, acknowledgment (ACK) packets are used. In contrast, we set the QoS level of the sensor data topic to zero, which does not use any ACK packets and does not provide any guarantees [12]. We chose to set the lowest QoS level for the sensor data topic to improve the system energy efficiency since sensor readings are usually exchanged more often compared to the other topics. Fig. 4 (left figure) shows the operation for this thread in more detail. Also, observe that we use POST methods for the sensor data topic, whereas we use PUT methods for the spilling flag topic. This is because, for the sensor data topic, we want accumulated readings in the cloud, as opposed to updating the existing data for the spilling flag message.
- **Cloud-Sourced Message Processor**: This thread retrieves the actuator data for each registered bottle from the cloud through GET methods periodically, and publishes it with QoS 1 to a topic specific to the bottle. The actuator data is only published under two conditions: i) the data has changed compared to the previous retrieval; ii) the data has not changed, but the edge must send a heartbeat message to the bottle to let it know that the connection is alive. These conditions are important to avoid sending messages unnecessarily while ensuring the ESP knows about the state connection. Fig. 4 (right figure) describes the operation for this thread.



good figure

Fig. 4: Left figure: edge operation for the MQTT received message processing for each subscribed topic. Right figure: operation for the actuator data transmission for each registered bottle. Figs. done with [13].

Another key feature of the edge is to be fault-tolerant. It monitors the incoming messages for each registered bottle to ensure the connection is still alive. If messages from a registered bottle are not received within an interval, the edge assumes node failures or link disruptions have occurred. In this case, the edge deregisters the bottle. The ESP also has a similar feature, and after a period of time, it will start sending registration messages again so that the connection can be reestablished. This feature is crucial to prevent sending messages unnecessarily and improve the system's energy efficiency and robustness.

Cloud: The cloud component uses a *commercial* cloud: DigitalOcean [14]. The backend is programmed in Node.js Express, which interfaces with a PostgresSQL database to store the data persistently [15-16]. The backend consists of GET, POST, PUT, and DELETE routes for sensor readings, volume thresholds, cleaning mode, and spilling hazards. POST is used for sensor readings for multiple data points, whilst PUT is used to update single parameters such as threshold, cleaning mode and spilling hazards. These components are packaged through Docker Compose, which packages multiple containers (Node.js Express and PostgreSQL), and runs on a DigitalOcean Droplet (akin to a virtual machine) (see Fig. 5) [15-17]. The PostgreSQL database resides in a persistent volume attached to the Droplet to ensure data is not lost between container restarts or server failures [14], [16]. The server IP is exposed and can be interfaced by the frontend and the edge. The frontend web dashboard is written in React.js, which interfaces with the backend to display sensor readings for bottles, along with actuation features for allowing users to configure volume threshold and toggle the cleaning mode (see Fig. 6) [18]. The backend also interfaces with SendGrid to send users alert email notifications for when the volume of the bottle falls below the user-set threshold or a spilling hazard is detected (see Fig. 7) [19]. The backend can easily be extended to support more bottles through an editable environment variable.

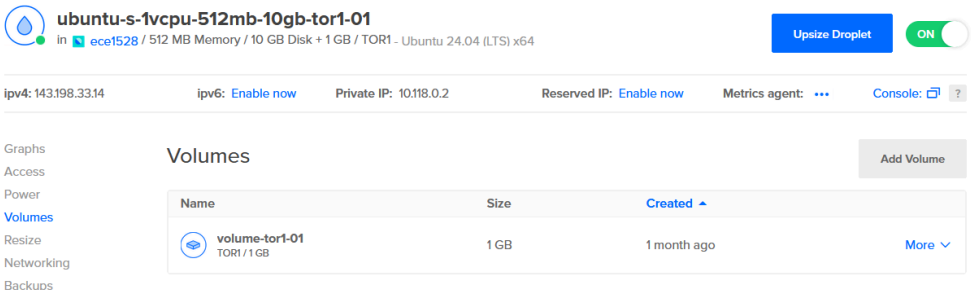


Fig. 5: Backend deployed on commercial cloud - DigitalOcean Droplet (virtual machine).

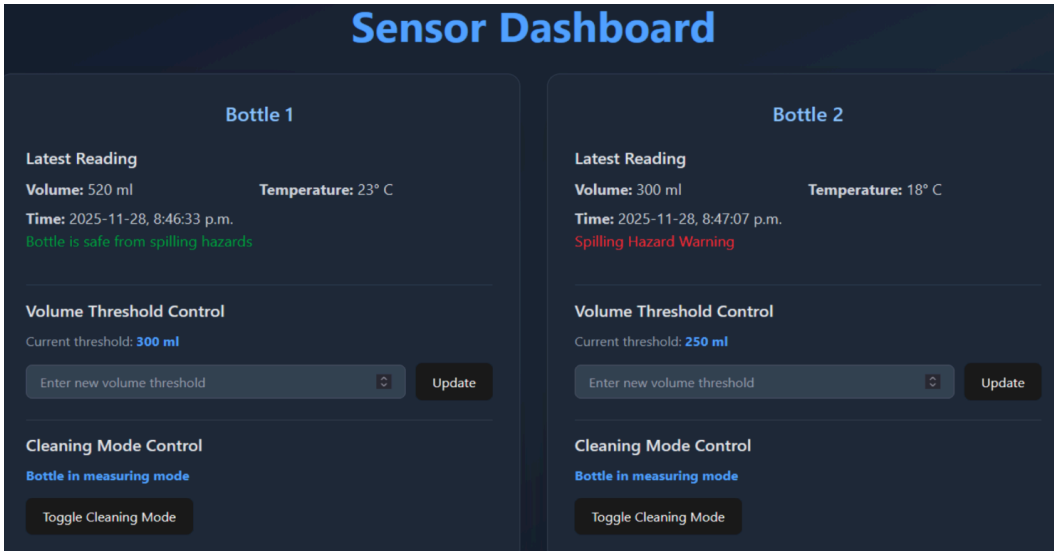


Fig. 6: Web dashboard with sensor readings, volume and cleaning mode control for two bottles.

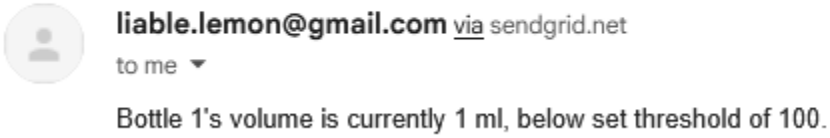


Fig. 7: Alert email sent to user when the current bottle volume falls below user-set threshold.

3. System Evaluation

The system was evaluated primarily through qualitative means. The ESP and edge device were placed within 5 meters of each other, and were connected via WiFi. The ESP was powered by a computer through the USB port. The ESP-edge communication started with the ESP sending registration messages every 0.5 seconds. After registration, the edge sent actuator data from the cloud to the ESP every 10 seconds. If the actuator data did not change between retrievals, a heartbeat message was sent from the edge to the ESP every 20 seconds. Once the ESP received the first actuator data, it started transmitting sensor data (temperature and volume), as well as the spilling flag every 0.5 second. Regarding the fault tolerance feature of the system, if the edge did not receive a message from a registered bottle for 60 seconds, it assumed the connection was lost and deregistered it. If no actuator data was received on the ESP for more than 1 minute, it assumed the connection was lost and started sending registration messages again. We simulated connection disruptions from each side by terminating the corresponding scripts (i.e. unplugging the ESP to simulate a connection termination). After restarting either the edge or the ESP script, depending on the simulated scenario, the system resumed its regular operation. We also verified that the spilling flag messages were successfully accompanied by sequence numbers to ensure orderly processing. Experiments were also carried out with two simultaneous bottles connected to the edge to verify the system's scalability and responsiveness. Overall, the system performed as it was designed to when evaluated qualitatively; data was transmitted successfully both upstream and downstream. These experimental results are illustrated in the project demo video. Future evaluations will include quantitative experiments, such as measuring packet delivery ratios across different environments and system settings (e.g., three or more bottles).

4. Related Works

From the literature, there are numerous attempts to build an IoT-based smart water bottle [1-3], [20-23]. These projects used many different sensors such as water float sensor, HC-SR04 to measure water level height, and photoplethysmogram for physiological monitoring [1], [20], [21]. Outside of the realm of sensors, others have adopted Bluetooth, Wi-Fi, GSM modem, and SMTP as mechanisms to transfer data to an application [3], [20], [22]. These projects have also implemented mechanisms to personalize water level intake, like considering weather conditions, DHT11's humidity levels, historical data, and fuzzy logic [2], [21], [22]. Projects also often featured a phone application to view the results more easily. These highlight potential features we could consider to improve the system.

5. Conclusion

~~In conclusion,~~ we have designed and implemented a smart water bottle system that can be easily used by users through a straightforward web dashboard and provides users with key information, such as water temperature and volume, and spilling risk. We discuss in the following additional potential use cases and future improvements for the system. Some use cases could include functioning as a pet water feeder, deploying at a restaurant for waiters to receive notifications to refill customers' bottles, and distilleries or perfumeries being notified of potential liquid loss during transportation. For each of these applications, modifications to the system are needed. Regarding future improvements, using batteries to power the bottles is crucial to offer the users mobility. Nevertheless, this change comes with some challenges. As the devices become energy-constrained, it is paramount to consider energy-saving sleep cycles in the ESP boards, and less frequent sending of sensor readings. Furthermore, the entire system could be shifted to a printed circuit board (PCB) for a more compact design. Also, Bluetooth can be incorporated as a wireless alternative to WiFi. With respect to the cloud, its deployment can also be improved through the use of WebSockets for instantaneous updates, along with water consumption tracking over time across multiple bottles for a more detailed user experience.

Some good references

References

[1] E. Jovanov, V. R. Nallathimmareddygari, and J. E. Pryor, “SmartStuff: A case study of a smart water bottle,” *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug. 2016, doi: https://doi.org/10.1109/embc.2016.7592170.

[2] A. Nirbhavane, R. Kushwaha, G. Singal, A. Tomar, V. Pal, and A. Gupta, “HydroPro: an IoT-Based Smart Water Bottle,” *2017 IEEE Region 10 Symposium (TENSYMP)*, pp. 1–6, Sep. 2024, doi: https://doi.org/10.1109/tensymp61132.2024.10752312.

[3] R. Ravindran, R. Ravindran and T. Anjali, "HydrationCheck: An IOT based smart water bottle," *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2022, pp. 1-5, doi: 10.1109/ICCCNT54827.2022.9984544.

[4] S. Santos. “ESP8266 NodeMCU with Load Cell and HX711 Amplifier (Digital Scale).” Random Nerd Tutorials. Accessed: Dec. 09, 2025. [Online]. Available: https://randomnerdtutorials.com/esp8266-load-cell-hx711/

[5] S. Santos. “ESP8266 NodeMCU MPU-6050 Accelerometer and Gyroscope (Arduino).” Random Nerd Tutorials. Accessed: Dec. 09, 2025. [Online]. Available: https://randomnerdtutorials.com/esp8266-nodemcu-mpu-6050-accelerometer-gyroscope-arduino/

[6] R. J. Fetick, “MPU6050_light,” *GitHub*, Mar. 09, 2022. Accessed: Dec. 09, 2025. [Online]. Available: https://github.com/rfetick/MPU6050_light

[7] R. Santos. “ESP32 DS18B20 Temperature Sensor with Arduino IDE (Single, Multiple, Web Server).” Random Nerd Tutorials. Accessed: Dec. 09, 2025. [Online]. Available: https://randomnerdtutorials.com/esp32-ds18b20-temperature-arduino-ide/

[8] R. Light, “MQTT Python client library.” Python Package Index. Accessed: Dec. 09, 2025. [Online]. Available: https://pypi.org/project/paho-mqtt/

[9] 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.

[10] J. Gaehwiler. “MQTT library for Arduino.” Arduino Documentation. Accessed: Dec. 09, 2025. [Online]. Available: https://docs.arduino.cc/libraries/mqtt/

[11] S. Cope. “Receiving Messages with the Paho MQTT Python Client.” Steve’s Internet Guide. Accessed: Dec. 09, 2025. [Online]. Available: http://www.steves-internet-guide.com/receiving-messages-mqtt-python-client/

[12] HiveMQ Team. “What is MQTT Quality of Service (QoS) 0,1, & 2? – MQTT Essentials: Part 6.” HiveMQ. Accessed: Dec. 09, 2025. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

[13] draw.io. Accessed: Dec. 09, 2025. [Online]. Available: https://app.diagrams.net/

[14] “AI-powered unified Agentic Cloud Infrastructure.” DigitalOcean. Accessed: Dec. 09, 2025. [Online]. Available: https://www.digitalocean.com/

[15] “Node.js web application framework.” Express. Accessed: Dec. 09, 2025. [Online]. Available: https://expressjs.com/

[16] P. G. D. Group. PostgreSQL. Accessed: Dec. 09, 2025. [Online]. Available: https://www.postgresql.org/

[17] “Docker compose.” Docker Documentation. Accessed: Dec. 09, 2025. [Online]. Available: https://docs.docker.com/compose/

[18] “React.” React Blog RSS. Accessed: Dec. 09, 2025. [Online]. Available: https://react.dev/

[19] “SendGrid email API.” SendGrid. Accessed: Dec. 09, 2025. [Online]. Available: https://sendgrid.com/en-us

[20] P. B. Pankajavalli, R. Saikumar, and R. Maheswaran, “Hydration Reminding Smart bottle: IoT Experimentation,” *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, Apr. 2017, doi: https://doi.org/10.1109/ipact.2017.8245105.

[21] A. E. Wijanarko, M. Abdurohman and A. G. Putrada, "A Fuzzy Logic Based Internet of Things (IoT) for Smart Water Bottle," *2019 5th International Conference on Computing Engineering and Design (ICCED)*, Singapore, 2019, pp. 1-6, doi: 10.1109/ICCED46541.2019.9161100.

[22] V. Poddar, V. Barnwal, and D. Saisanthiya, “Aqua-Trail: IOT Based Smart Water Bottle,” *2024 International Conference on Trends in Quantum Computing and Emerging Business Technologies*, pp. 1–6, Mar. 2024, doi: https://doi.org/10.1109/tqcebt59414.2024.10545297.

[23] N. E. Lee, T. H. Lee, D. H. Seo, and S. Y. Kim, “A Smart Water Bottle for New Seniors: Internet of Things (IoT) and Health Care Services,” *International Journal of Bio-Science and Bio-Technology*, vol. 7, no. 4, pp. 305–314, Aug. 2015, doi: https://doi.org/10.14257/ijbsbt.2015.7.4.30.