Portable Water Tracking Attachment

Final Report

ECE445 Spring 2024

Project #44

Subha Somaskandan, Subhi Sharma, Cindy Su

Professor: Arne Fliflet

TA: Luoyan Li

Abstract

Our project was to build a portable water tracking attachment to a 32oz water bottle that would send reminders based on the weight left in your bottle. We were able to successfully connect to an app and measure the weight to send reminders, as well as make our design portable and compact. For further testing, we hope to make the attachment be able to latch to any size of water bottle, and expand to track drinking of various liquids, such as coffee, or juice.

Table of Contents

1. Introduction

1.1 Problem and Solution

With a busy schedule, it can be difficult to remember to drink enough water during the day. Most water tracking products on the market are not customizable to one's lifestyle or daily habits. The amount of water that everyone needs to drink per day depends on their height, weight, sex, physical activity, and climate. There are a variety of water drinking apps, but these do not actually check if the right amounts of water are being drunk, and they can be ignored without completing the task. Furthermore, there are also water bottles on the market that have markings on them to indicate how much water to drink per hour, but these force one to buy a new bottle entirely. Our solution is a portable box attachment to a 32oz water bottle that measures how much water you are drinking, and connects to a smartphone app to remind you to do so. Within the app, you can input information such as your sex, age, activity level, and your recommended water intake will be calculated, or you can input your intake manually. Every hour, a reminder will be sent out to drink a specific amount of water, and the attachment will check using weight and inertial movement sensors if this is completed. The water drinking data will then be relayed via a microcontroller to the app, where you can see your progress for the day.

1.2 High Level Requirements

1. The application must set water drinking habits and send reminders to drink calculated amounts per hour based on the user's age, sex, activity level, and hours of sleep.

2. If the user finishes the water requirement before the reminder is sent, the application must not send a reminder until the next hour that the user does not meet the water drinking requirement. Similarly, if the user finishes their water requirement for the day, the application must not send any notifications after this point.

3. The application must track and log the user's daily water consumption and present the data in an easy-to-understand chart format for the user to check daily water intake.

1.3 Block Diagram

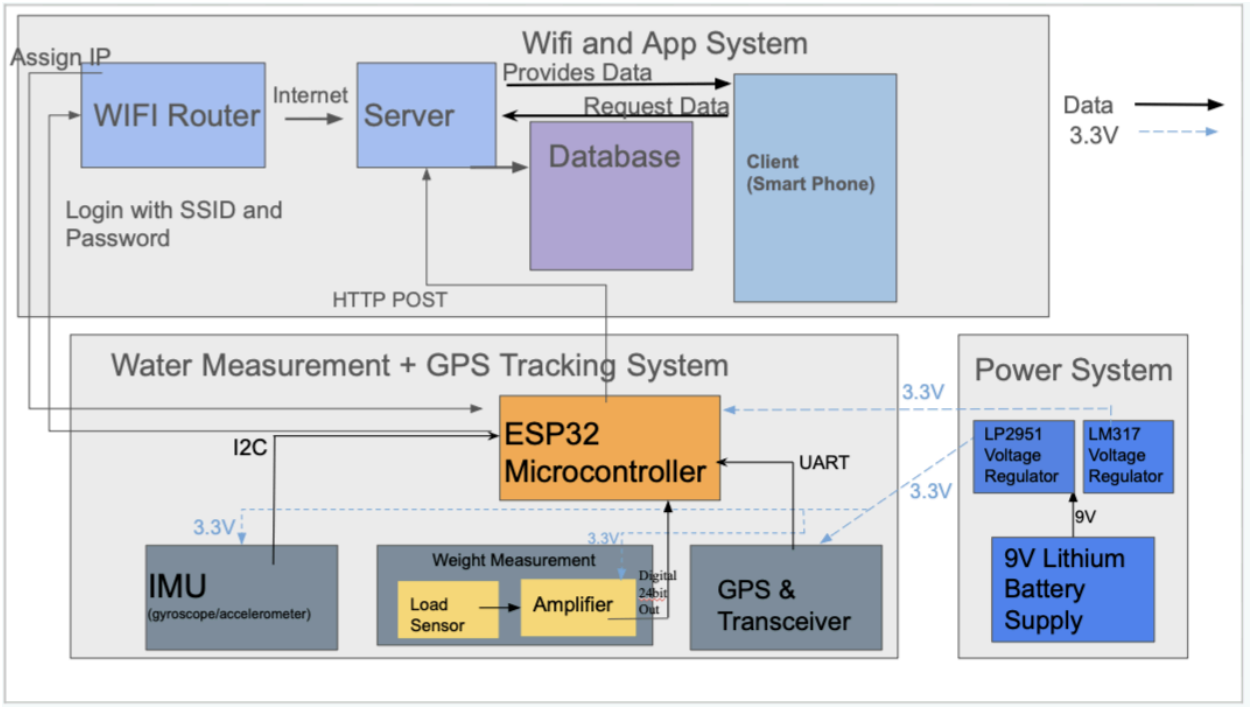

Figure 1: Block Diagram

Our design is made up of 3 subsystems, the Wifi and App Subsystem, the Water Measurement and GPS Tracking Subsystem, and the Power System. The Wifi and App subsystem receives data from the microcontroller through a server and wifi router, and stores the data in a database, which can be accessed from the smartphone app client. The Water Measurement and GPS Tracking system measures the weight of the water bottle when it is placed down. When designing, we omitted the GPS & Transceiver from this subsystem as we did not have a functional printed circuit board, and we could not test this chip. The positioning and weight data is sent to the microcontroller through I2C and digital data, respectively. Finally, the Power System delivers 3.3V through two voltage regulators, which are connected to a 9V Lithium battery supply. We originally had a 3V battery supply being stepped up to 3.3V and 5V, but we quickly realized that this would not work with linear regulators. Also, all of our components can be powered using 3.3V, so we did not need to produce a 5V output. The final design comprises the LM317 chip which delivers 3.3V to the microcontroller, and the LP2951 which delivers 3.3V to the IMU as well as the Amplifier in the Water Measurement subsystem.

2. Design

2.1 Design Procedure

For the power subsystem, we designed with a 9V battery going to two linear voltage regulators, the LM317 and the LP2951, both outputting 3.3V. We chose a 9V battery as this could be provided by just one D battery cell, which makes our design a bit more portable and accessible. We chose the LM317 to power the microcontroller due to its high current limit, as well as its variable output voltage. The LM317 is rated for 1.5A, which is about double the expected consumption of the microcontroller. The LM317 also has an external resistor divider to set its output voltage, so we could test with various resistances and use a formula provided by the datasheet to calculate the output voltage. Finally, we decided to power our sensors with the LP2951 since we wanted to ensure the microcontroller had ample leeway for its current, as well as making our testing easier since we could pinpoint errors from the separate chips much better.

For the water measurement and GPS tracking system, we originally had the ESP32 microcontroller, a load cell and HX711 amplifier, the MPU6050 IMU, and a GPS chip and its accompanying antenna. We chose the ESP32 microcontroller since it was highly compatible with the Arduino interface, where the majority of our testing and code writing would occur. Furthermore, its wifi capability was important to our database and app connectivity portion of our project. Our load cell had a capacity of 5kg, which was much greater than the maximum weight we would be measuring. We chose to opt for a higher weight capacity as this also means the sensor is more sensitive to changes in weight, and this is vital to the scope of our project. Furthermore, the HX711 was needed to convert the analog weight data output of the load sensor into digital logic so this was readable for the microcontroller. As for our IMU, we chose a basic model that could display position and acceleration in three different directions. We essentially only needed to track the tilt of the water bottle to detect when it was being picked up, so the MPU6050 would meet our requirements. Finally, the initial design included a GPS Transceiver module as an additional perk to the main functionality. However, during the design and building phases, we were unable to get a working PCB by the required deadline, and this chip is a surface mount. Thus, we were unable to test this feature and omitted it from the project's main scope.

For the Wifi and App subsystem, we fetch data from the ESP32, and go through a database to store it in the user's login in the app. We chose to go through a database due to the fact that we needed to store data frequently for our design. Furthermore, we tested with wifi since most phones do have wifi connectivity, but for further testing, we could test with LTE networks, as this is more accessible overall. Furthermore, logging in on the app through SSID provides security for one's data, as well as the ability to have multiple accounts.

2.2 Design Details

2.2.1 Power Subsystem

The power subsystem consisted of a 9V battery input, and the LM317 and the LP2951 voltage regulators, which both output 3.3V. Table 1 shows the current and voltage operating points for the devices that will be powered by the subsystem. The LM317 will solely power the microcontroller, and the LP2951 will power the MPU-6050 as well as the HX711 amplifier.

Component/Device	Voltage & Current Requirements
ESP32-WROOM-32 Microcontroller	Input Voltage: +3.3V Max Current: 1A
MPU-6050 3 Axis Gyroscope + Accelerometer	Input Voltage: +2.375V - +3.46V Operating Current: 3.9mA
HX711 Load Cell Amplifier	Input Voltage: +2.7V - +5.5V Operating Current: <1.5mA

Table 1: Power Requirements for All Components/Devices

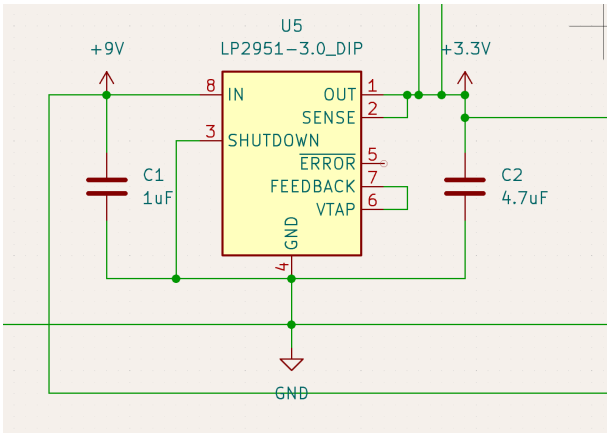

Figure 2: Linear Regulator LP2951 Schematic

For the LP2951 voltage regulator, the output is fixed at 3.3V, and was routed to the IMU and the HX711 amplifier. The input capacitance of 1uF and the output capacitance of 4.7uF were provided by the datasheet, as these values overall reduce ripple in the voltage. Internally, to produce a 3.3V fixed output, the feedback and vtap pins are tied together, output and sense are tied together, and shutdown is connected to ground.

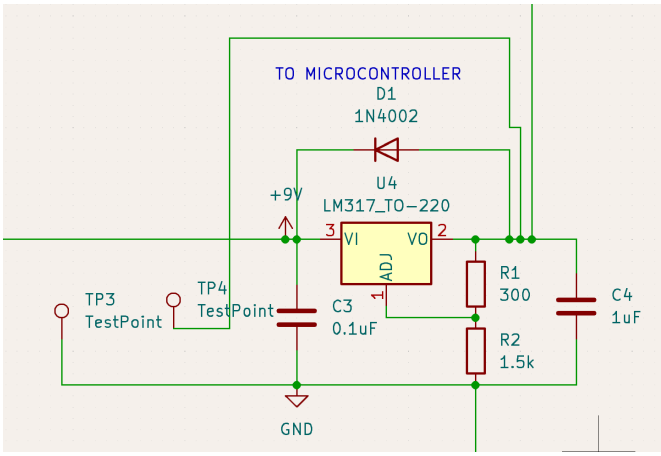

Figure 3: Linear Regulator LM317 Schematic

For the LM317, an input of 9V is given, and the output voltage is variable depending on the values of R1 and R2, as shown in Figure 3. The input and output capacitors have values of 0.1uF and 1uF, respectively, and these were provided by the datasheet to produce more stable voltages. There is also a diode connected from output to input, to provide a path for current to flow in case the difference between the input and output voltage threshold drops too low. This was also recommended by the datasheet. While there was no set formula given by the datasheet due to the variety of applications, we loosely used the formula below to reason how to get the desired output voltage of 3.3V.

$$VO = VREF\ (1+R2/R1)+(IADJ \times R2)$$

From the formula, it can be seen that increasing R2 would also increase the output voltage. With this in mind, we tested originally with values of R1 = 2k, and R2 = 1k, and got about 3.28V. However, when we used these values in conjunction with the microcontroller, we realized that the microcontroller needed a bit more voltage, since some portion of it would get lost with use. Thus, this led us to overshoot on the output of the LM317 slightly, and to go to about 4.1V, which is still safe for our microcontroller. This led us to values of R1 = 300 and R2 = 1.5k.

2.2.2 Weight Measurement Subsystem

This system contains an inertial measurement unit (IMU), transceiver, and a load sensor with an accompanying amplifier which all communicate with the ESP32 microcontroller. This system measures the weight of the water bottle through the load sensor and amplifier, specifically when it is placed down, which is indicated by the IMU. This data is sent to the microcontroller through a I2C protocol for the IMU, and UART protocol for the load cell and amplifier. This system gets 3.3V from the power subsystem as input to send to the load sensor, the amplifier, and the IMU.

2.2.2.1 Inertial Measurement Unit (MPU-6050 Module 3-Axis Gyroscope and Accelerometer)

Input: Position of the chip, which translates to the position of the water bottle.
Output: SDA and SCL pins to I/O of the ESP32

This IMU contains a triple axis gyroscope and accelerometer which have digital outputs. The unit will be connected to the microcontroller and power supply.

The IMU is necessary because it will detect when someone has picked up their bottle based on the angular position of the chip. This data will be sent to the microcontroller using I2C protocol so that it can be used to indicate when the bottle has been placed back down to initiate the weight sensor. The range of the gyroscope is user-programmable from 200, 500, 1000, and 2000 degrees/second. The unit has a 3.9 mA operating current when the available 6 degrees of motion are all enabled.

Pin	Name	Connection
13	VDD	To Power Supply Voltage and Digital I/O Supply Voltage
18	GND	To Power Supply Ground
23	SCL	I2C Serial Clock to Microcontroller SCL
24	SDA	I2C Serial Data to Microcontroller SDA

Table 2: IMU Pin Connections

2.2.2.2 Weight Measurement Using a Load Cell and HX711 Amplifier

Amplifier Inputs: Load Cell's channels S- and S+
Amplifier Outputs: SCK and DT to the ESP32

The HX711 is necessary for the load cell because it ensures that the output voltage of the strain gauge is proportional to the excitation voltage. This helps to make the final voltage produced by the load cells accurate and free from noise. This unit is a 24 bit analog-to-digital converter meant for weighing sensors.

The ESP32 microcontroller has a HX711 library in order to effectively calibrate the load cell, set any offsets, and obtain any weight data. Commands within the ESP32 IDE such as read() and get_value(number of readings) obtain the raw readings and the average of the last defined number of readings minus the tare weight, respectively.

The load cell is 12.7mm x 12.7mm x 80mm and will be mounted on a small platform. The range of weight it can measure is from 0-5kg, and the rated output is 1.0±0.15 mV/V. It has a recommended excitation voltage of 3.3V. The input resistance is 1066±20Ω and the output resistance is 1000±20Ω.

Pin	Name	Connections
3	AVDD	To Power Supply of 2.6-5.5V and E+ of the Load Cell
5	AGND	To Power Supply Ground
7	INA-	To Channel S- of the Load Cell
8	INA+	To Channel S+ of the Load Cell
9	INB-	To Power Supply Ground
10	INB+	To Power Supply Ground

Table 3: Pin Assignments of HX711 Amplifier

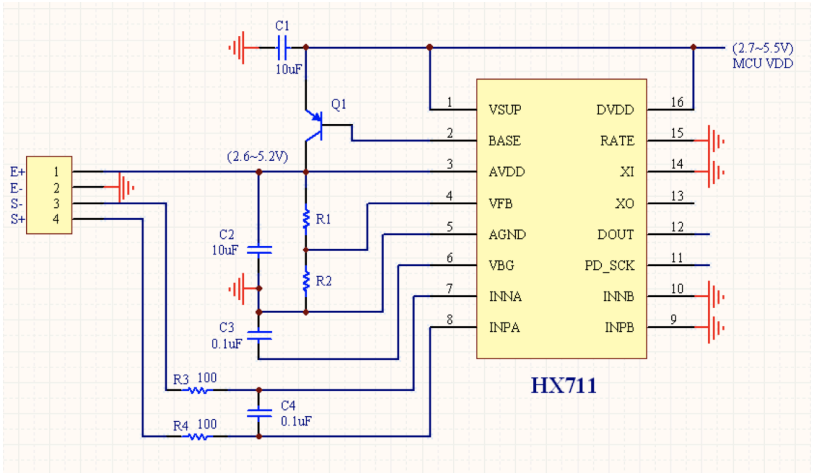

Figure 4: HX711 Module connected to Load Cell

In order to connect the load cell to the amplifier and microcontroller:

- Connect the red wire (E+) to the E+/GND of the ESP32 and HX711 modules

- Connect the black wire (E-) to GND for the HX711 amplifier and GPIO 16 pin of the ESP32 module
- Connect the white wire (S-) to the INA- pin for the HX711 amplifier and GPIO 4 pin of the ESP32 module
- Connect the green wire (S+) to INA+ of the HX711 amplifier and the 3.3V pin of the ESP module.

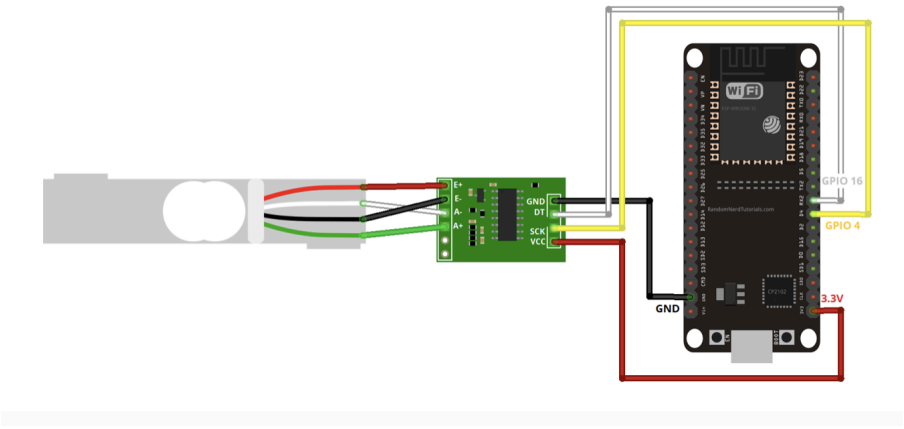

Figure 5: Load cell, HX711 and ESP32 Configuration

2.2.3 Wifi and App Subsystem

When the sensor detects a change in the weight of the water bottle, it updates the measured weight in real-time. This updated weight value is then transmitted to the microcontroller (ESP32) via a digital signal, The microcontroller processes this information and sends it to a remote database through a WiFi connection. And the real-time updates are pushed to a Postgresql database hosted on a cloud server.

2.2.3.1 Wifi-Module (ESP32 WROOM 32)

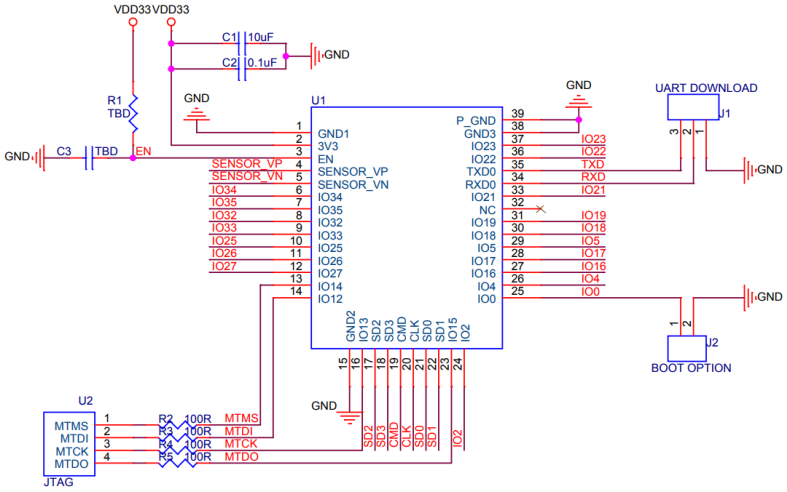

Figure 6: Peripheral Schematics

Pin number	Name	Usage
35	UART TXD0	Data transmission, will be connected to the TX pin of the USB to Serial port convertor
34	UART RXD0	Data reception, will be connected to the RX pin of the USB to Serial port convertor
25	GPIO0	Determine if code is run from memory or loaded via UART
24	GPIO2	Determine if code is run from memory or loaded via UART
3	EN	When Programming the esp32, we will pull EN pin low briefly to trigger reset
1, 15, 39	GND	Ground
3	3V3	3.3V power

Table 4: ESP32 Pin Diagram

Configuration

To program the ESP32 via the UART terminal and ensure it reads new code instead of executing preloaded code from its memory, it is necessary to manipulate specific GPIO pins to induce the correct boot mode. Specifically, GPIO0 must be pulled low during the reset to enter the bootloader mode. After the new code has been transmitted to the ESP32 via UART, a subsequent reset is initiated. For the ESP32 to boot from its updated internal memory and execute the new code, GPIO0 should be set high before this reset. This sequence ensures that the ESP32 exits the bootloader mode and starts running the newly uploaded program from its flash memory upon reboot.

Booting Mode			
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0
GPIO2	Pull-down	Don't-care	0

Figure 7: Strapping Pins table from esp32 datasheet

To implement the bootloader on esp32. The microcontroller will be connected with a USB/UART bridge, reset button, and boot/flash button on a breakout board. Whenever the Arduino IDE uploads the code, the microcontroller will change to download boot mode by pressing both buttons, which will write the flash memory and programme the board.

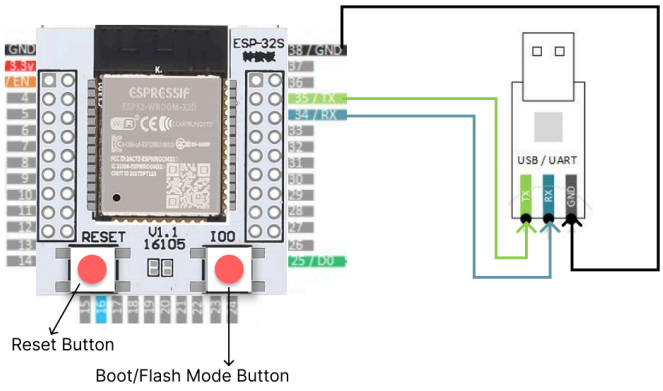

Figure 8: Connecting esp32 with the USB/UART bridge

Wi-Fi Configuration

To set up Wi-Fi connectivity on the ESP32, the Arduino IDE will be used to program and deploy the necessary configuration script. This script includes the SSID, password, and the server's URL for the Wi-Fi network that the ESP32 is intended to connect to. After the script is successfully transferred to the

ESP32, it establishes a connection to the designated Wi-Fi network with the specified credentials and sends POST requests to transmit data to the database.

2.2.3.2 Front-End Application

When opening the app, the user will see a homepage with a water bottle at the center, representing the current water level within your actual water bottle. The homepage also has a button that navigates to the statistics page. This page shows the analysis of the user's water intake and displays the intake data in a chart presented through weekly, monthly, and yearly statistics, allowing users to track and manage their hydration habits. Another feature accessible from the home page is the location page. This page provides the functionality of showing the real time location of the user's water bottle to ensure that it can always be found easily.

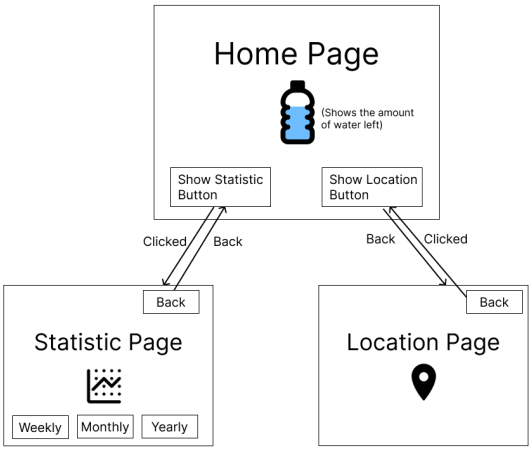

Figure 9: Front-End Application flowchart

Every 50 minutes, the app will check to ensure the user's water intake aligns with their pre-set hydration goals. If the user has not consumed the desired amount of water within this timeframe, the app will send out a notification to the user as a reminder, encouraging the user to drink water and ensure they maintain high hydration levels throughout the day.

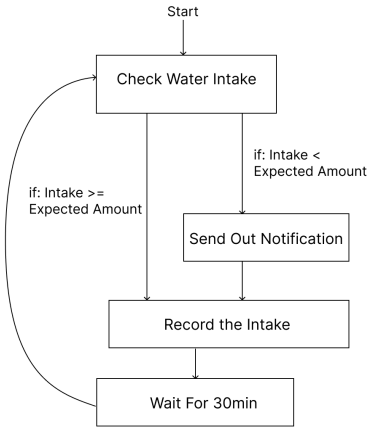

Figure 10: App Notification System flowchart

2.2.3.3 Back-End Application

The cloud database is going to be hosted on Amazon Relational Database Service and connected to the pgAdmin server. This database will be responsible for receiving the data from the microcontroller and sending them to the mobile app.

There will be three tables in the database. The first one will be the 'user_data' table, designed to include each user's personal information, such as username, gender, age, weight, physical activity, recommended hourly water intake, and password. The second table will be the 'water_intake' table, designed to contain columns for intake id (the unique id for each water intake), username, intake timestamp, and water amount in milliliters. Lastly, the 'notification' table will contain a flag for each user so that the app will send out the notification when the flag for the corresponding user becomes 1.

	username [PK] character varying (5	gender integer	age integer	weight integer	physical_activity integer	password character varying	daily_water_intake integer
1	user1	0	22	54	2	123	100
2	user2	1	25	60	1	123	100
3	user3	0	18	50	3	123	100

Figure 11: User Data Table

	intake_id [PK] integer	username character varying (50)	intake_timestamp timestamp without time zone	water_amount_ml integer
1	1	user1	2024-03-27 08:30:00	250
2	2	user2	2024-03-27 09:45:00	300
3	3	user3	2024-03-27 10:15:00	200

Figure 12: Water Intake Table

	username character varying (255)	flag integer
1	user2	0
2	user3	0
3	user1	0

Figure 13: Notification Table

When a user first logs into the app, it will prompt them to input their personal information, which will be sent to the database and added as a new row in the 'user_data' table. Whenever a user drinks water from the water bottle, a change in weight will be detected by our device, and the microcontroller will send this change to the database, adding a new row to the 'water_intake' table.

Every 15 minutes, the app will check to ensure the user's water intake aligns with their pre-set hydration goals. This is achieved by making a fetch request to the database for the 'water_intake' table. The API call will request the total amount of water intake for the corresponding user on that day, and the app will verify if the user has consumed the desired amount of water. This verification involves making another call to the 'user_data' table to obtain the recommended hourly water intake. If this value exceeds the actual hourly water intake the user has had, the app will send out a notification to the user as a reminder, encouraging them to drink water and maintain high hydration levels throughout the day.

This is how the backend interacts with each pages of the front end:

1. Main Page: This is the application's home screen, featuring an image of a water bottle at the center that dynamically updates to reflect the current volume of water the user has consumed on that day. This update is achieved by fetching the 'water_intake' table from the database and calculating the total amount of water consumed by the corresponding user on that day.
2. Statistics Page: This page provides users with data on their hydration habits, displayed through a curved area graph. It shows the daily, weekly, and monthly water intake of users. This information is also retrieved by fetching the 'water_intake' table from the database and calculating the total amount of water consumed by the corresponding user over the selected time period—daily, weekly, or monthly—based on the graph the user chooses to view.

3. Location Page: This page offers the functionality of displaying the real-time location of the user's water bottle. This is accomplished by fetching the 'location_data' table from the database for the corresponding user and integrating this data with the Google Maps API to display the map and the marker indicating the location.
4. Settings Page: This page allows users to update their personal information, such as username, gender, age, weight, and physical activity. When a user submits the changes, they are updated in the 'user_data' table in the database.

3. Verification

3.1 Power Subsystem

The main testing for the power subsystem involved ensuring the output voltages from our linear voltage regulators were the desired values, and that they could be used in conjunction with all the devices, and confirming these requirements with Table 1 in the Appendix. For the requirement for the 9V battery input, we measured the voltage using a multimeter, and found the voltage to be steady around 9.4V, which was plenty and accounted for drainage throughout use. We repeated the same procedure with the voltage regulators. For the LP2951, output voltage was very steady around 3.328V, and both the IMU and HX711 amplifier were room temperature to the touch, which means their currents were well within the limit of the LP2951.

For the LM317 output voltage, it could be varied by a set of external resistors, in a resistor divider configuration. Originally, we wanted 3.3V, so we tested various values and measured the output pin with respect to ground using a multimeter. Our trials and outcomes are shown in Table 5 below.

Input Voltage (V)	R1 (**Ω**)	R2 (**Ω**)	Output Voltage (V)
9	2.7k	1k	3.987
9	3k	1k	4.234
9	2.7k	1k	3.9
9	3k	1k	4.260
9	4.7k	1k	5.404
9	2.2k	1k	3.614
9	2k	1k	3.44

Table 5: LM317 input/output voltage relationship for resistor combinations and input voltage of 9V

As shown in the last trial, R1 = 2k and R1 = 1k produced 3.44V, but when testing with this on the same breadboard as all of our other components, it seemed the output voltage would dip frequently. As this was powering the microcontroller, our code would not run, as the microcontroller was not getting enough voltage. After this, we tested with other combinations, but landed on R1 = 300 and R2 = 1.5k, and this was producing a value hovering steadily around 4.1V, which meets our requirements. After testing with the microcontroller, our code ran normally and our design worked fully.

3.2 Weight Measurement Subsystem

The testing for the IMU was done according to the R&V table for section 2.2.2.1 as listed in the appendix. Sample code from the Arduino IDE was used to obtain basic readings for the x, y, and z axes. These readings were then used in the main code to trigger the weight sensor to start if the z acceleration indicated to be flat against a surface. Figure 14 shows the data log collected by this program, and we omitted any data that is not the z acceleration for the scope of this project. The z acceleration in the figure proves to be close to the acceleration of gravity indicating that the IMU is flat, which aligns with our goals.

```
16:20:45.280 -> Acceleration X: -0.17, Y: -0.46, Z: 9.06 m/s^2
16:20:45.280 -> Rotation X: -0.06, Y: -0.02, Z: 0.01 rad/s
16:20:45.280 -> Temperature: 28.27 degC
16:20:45.280 ->
16:20:45.768 -> Acceleration X: -0.18, Y: -0.46, Z: 9.04 m/s^2
16:20:45.768 -> Rotation X: -0.06, Y: -0.02, Z: 0.01 rad/s
16:20:45.768 -> Temperature: 28.25 degC
```

Figure 14: Data Log for the MPU6050

Commands such as `sensors_event_t(a, g, t)`, and `empmpu.getEvent(&a, &g, &temp);` were used to get this data from the chip which uses I2C protocol.

The test log for the HX711 amplifier and load cell displays the calibration data when an empty water bottle is placed on top of the scale. The code accounts for any object on top of the scale when calibrating, and then it only measures the weight inside or on top of that object. After the line where the scale is set up, we tested further by placing different weights on top of the water bottle to ensure that the calibration was correct, as per the R&V table for section 2.2.2.2 in the appendix. This test program in Figure 15 gives an updated weight after one second. This code was implemented with the IMU's location, where a condition set the scale to be triggered if the IMU was flat.

```
11:21:28.312 -> HX711 Demo
11:21:28.312 -> Initializing the scale
11:21:28.312 -> Before setting up the scale:
11:21:28.312 -> read:           208310
11:21:28.312 -> read average:           208326
11:21:30.150 -> get value:              208334.00
11:21:30.625 -> get units:              208309.0
11:21:32.047 -> After setting up the scale:
11:21:32.047 -> read:           208341
11:21:32.148 -> read average:           208334
11:21:34.048 -> get value:              -14.00
11:21:34.511 -> get units:              0.1
11:21:34.981 -> Readings:
11:21:34.981 -> one reading:    -0.0    | average:      0.08485
11:21:51.026 -> one reading:    0.3     | average:      0.26549
11:22:07.402 -> one reading:    0.3     | average:      0.31750
11:22:23.774 -> one reading:    354.2   | average:      1107.15076
11:22:40.140 -> one reading:    869.7   | average:      869.50690
11:22:56.487 -> one reading:    10.4    | average:      10.32960
```

Figure 15: HX711 Test Log

3.3 Wifi and App Subsystem

The esp32 is able to connect to wifi and successfully push the correct data(including username, the time of the updates, and water intake amount) to the database. The app is also able to fetch the correct data using the defined url setted up in the server.


```
[
    {
        "intake_id": 1,
        "username": "user1",
        "intake_timestamp": "2024-03-27T13:30:00.000Z",
        "water_amount_ml": 250
    },
    {
        "intake_id": 2,
        "username": "user2",
        "intake_timestamp": "2024-03-27T14:45:00.000Z",
        "water_amount_ml": 300
    },
    {
        "intake_id": 3,
        "username": "user3",
        "intake_timestamp": "2024-03-27T15:15:00.000Z",
        "water_amount_ml": 200
    },
    {
        "intake_id": 4,
        "username": "user1",
        "intake_timestamp": "1970-01-01T06:00:03.000Z",
        "water_amount_ml": 4095
    },
```


Figure 16: Data pushing to database from ESP32

4. Cost and Schedule

The total component and part cost we needed to purchase is listed below in Table 6, excluding shipping and taxes. For the smaller components like capacitors, resistors, and voltage regulators, we were able to utilize the self service inventory, with no cost to us. With shipping and taxes included, our final total for components and parts is $111.98. For labor costs, we are assuming a salary of $35/hour for each teammate, and estimating 200 hours each, this comes out to $7,000 per person. Having 3 people, the total labor cost for the team comes out to 3 x &7,000 = $21,000. Furthermore, we will also be utilizing the machine shop, with a quote of 10 hours, assuming a salary of $20 per hour. Thus, our project comes out to a grand total of $21,311.98.

Component/Parts Description and Part Number	Manufacturer	Vendor	Quantity	Extended Price
ESP32-WROOM-32 Microcontroller	Espressif	Amazon	Pack of 3	$12.99
USB to Serial port CP2102	HiLetgo	Amazon	1	$7.49
YZC-133 Load Cell Weight Sensor	Geekstory	Amazon	Pack of 2	$10.99
MPU-6050 3 Axis Gyroscope + Accelerometer	DIANN	Amazon	Pack of 5	$12.99
HX711 Load Cell Amplifier	Stemedu	Amazon	Pack of 5	$9.49
L70RE-M37 Navigation GPS Transceiver Module	Quectel	DigiKey	2	$18.60
W3011A RF ANT 1.575GHZ	Pulse Electronics	DigiKey	5	$8.45
32oz Water Bottle	N/A	Amazon	1	$12.99
9V Battery	Generic Brand	ECE Supply Shop	1	$1.50
TOTAL	—		—	**$95.49**

Table 6: Total component quantity and cost

5. Conclusions

This project aims to track water consumption, and be a customizable tool to help the user stay more hydrated. The overall design has been implemented through a breadboard, and all of the high level requirements have been met through this design.

We successfully have been able to trigger the weight sensor via the IMU to read the weight once an hour. User input has been established via the app subsystem, which is incorporated into tracking the amount of water that needs to be drunk per hour. If this requirement is not met, there is a reminder sent through the app. Our design is fully functional on a breadboard, however we were unable to get a printed circuit board working in time.

If time allotted, we would have done more testing with caveats related to refilling the water bottle, and tracking water consumption if weight increases throughout the hour. Due to us having a limited amount of memory on the microcontroller and limited time, assumptions were made to make the demo more efficient. We would like to improve upon our design in the future by making the device more portable and flexible to fitting all sizes of bottles, not just 32 oz. Additionally, we have considered using other liquids, such as coffee or juice, for testing and calibrating the load cell.

According to the IEEE Code of Ethics, Section I, Point VI, maintaining technical competence and undertaking tasks only after clear limitations have been outlined as mentioned, could result in potential similarities. However, our design does have clear differences as it is portable and removable from the bottle. This device can be connected to any 32 oz bottle, rather than being built into one. We understand the ethical standard of not infringing on any existing patents and we will not be repeating any existing processes already done by HidrateSpark.

Furthermore, according to Section II, Point VII in the IEEE Code of Ethics, discrimination based on age, gender, sex, ethnicity is not tolerated and innovation should reflect this as well. With our product, we do have to make calculations based on metrics such as these, so we have to be careful to not assume anything or be offensive. One idea we have to combat this is to allow an option for the user to set their water intake manually, as this may be what works the best for them. We can also put forth our calculations as recommendations or suggestions rather than facts, as everybody will be different.

This project aims to be environmentally friendly due to it theoretically being customizable for all sizes of bottles. The user does not have to buy many different bottles based on how much water they drink. Additionally, many items like this on the market are not sending reminders and physically tracking water consumption, so many users do not see benefits with those products.

Citations

[1] “NMEA Parser Example.” *GitHub*, github.com/espressif/esp-idf/blob/master/examples/peripherals/uart/nmea0183_parser/README.md. Accessed May 2024

[2] Dave, et al. “ESP32 with Load Cell and HX711 Amplifier (Digital Scale).” *Random Nerd Tutorials*, 23 Apr. 2022, randomnerdtutorials.com/esp32-load-cell-hx711/#hx711-amplifier. AAccessed May 2024

[3] PulseElectronics, productfinder.pulseelectronics.com/api/open/part-attachments/datasheet/W3011A. Accessed May 2024

[4] LM317 3 Terminal Adjustable Voltage Regulator Datasheet, Texas Instruments, https://www.ti.com/lit/ds/symlink/lm317.pdf Accessed May 2024

[5] L70 Hardware Design - Rs Components, docs.rs-online.com/74ea/0900766b8147dbe2.pdf. Accessed May 2024

[6] LP295x Adjustable Micropower Low-Dropout Voltage Regulators Datasheet, Texas Instruments, https://www.ti.com/lit/ds/symlink/lp2952-n.pdf?ts=1708657346374&ref_url=https%253A%252F%252Fwww.google.com%252F. Accessed May 2024

[7] How to Convert a Load Cell Reading into Total Weight, Hunker, https://www.hunker.com/13408598/how-to-convert-a-load-cell-reading-into-total-weight. Accessed May 2024

[8] How to connect ESP32 for programming, https://www.lab4iot.com/2019/07/14/tutorial-on-how-to-program-the-esp32-wroom-32-or-esp32f/ Jul 14, 2019. Accessed May 2024

[9] ESP32-wroom-32 Datasheet, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf. Accessed May 2024

[10] IEEE Code of Ethics, IEEE, https://www.ieee.org/about/corporate/governance/p7-8.html. Accessed May 2024

Appendix

Requirements	Verification
Section 2.2.1 Lithium Battery Unit must supply 9V +/- 5% to the Voltage Regulators when active	1. Use a multimeter to connect across the battery unit's positive and negative leads, and measure the voltage and note any fluctuations. Ensure the steady value is within 5% of 5V.
Section 2.2.1 The voltage regulators LM317 must output 4.1V within +/- 5% with current up to 1.5A, and the LP2951 must output of 3.3V within +/- 5%, current up to 100mA	1. Probe OUT and GND on the LP2951 and the LM317 with the positive and negative leads (respectively) of a differential probe that can be connected to an oscilloscope. 2. Once displayed on the oscilloscope, measure average voltage and ensure it is within 5% of desired output voltage. If necessary, measure maximums and minimums to this same tolerance as well. 3. Monitor the devices and ensure they do not maintain a very high temperature, as this is a good indicator of current load.

Table 1: R&V Table for Power Subsystem

Requirements	Verification
Section 2.2.2.1 The IMU (Inertial Measurement Unit) sensor data, specifically x, y, and z angular velocity and acceleration, must be able to be read on the microcontroller IDE.	1. Program ESP32 Board with Arduino IDE, and download the MPU6050 library. 2. Ensure pin connections to IMU and ESP32 are correct as outlined in 2.2.2.1. 3. In the code, initialize an object of the MPU6050 library, and follow the documentation to set ranges for the accelerometer and gyroscope. 4. Using the MPU6050 documentation, write code to get the values for x, y, and z acceleration and angular velocity, and upload code. 5. Move and rotate the water bottle that the IMU is attached to in all three directions, and ensure the display readings correspond accordingly. 6. Document changes in the z-acceleration only. If the value is close to the acceleration of gravity (9/8m/s^2), it is correct.
Section 2.2.2.2	1. Program ESP32 Board with Arduino IDE, and download the HX711 library.

The Load Cell and Amplifier Combination must be able to detect changes in weight as little as 2.6oz[1] +/- 5%, as the user drinks water, and displays it on the microcontroller IDE	2. Ensure pin connections to the load cell and amplifier within themselves as well as the amplifier to the microcontroller are correct based on 2.2.2.2. 3. Following the documentation for the HX711 Library, calibrate the sensor with the water bottle, which has a known weight of 0.590kg. 4. After calibration, fill up the water bottle to full capacity with water. 5. Using the HX711 Library, write code to get and display the weight of the water bottle, and verify it is heavier due to the added weight of the water. 6. Drink or pour out amounts close to 2.6oz and place the water bottle back down flat, and read the weight again, and ensure it is within 5% of 2.6oz subtracted from the original weight of the water bottle. Repeat for bigger amounts of water to verify.

Table 2: R&V Table for Weight Measurement Subsystem

Requirement	Verification
Section 2.2.3 A laptop must be able to transmit code to the ESP32 through a USB Type-C connector to UART	Connect the ESP32 to a LED and to the laptop using a USB Type-C cable and upload a led blinking test code. Confirm the successful code transmission by observing the expected behavior of the blinking LED.
Section 2.2.3 The ESP32 must successfully connect to a specific Wi-Fi network using credentials provided in the Arduino IDE code, which includes the SSID, password, and server's URL .	After uploading the Wi-Fi configuration code to the ESP32, verify the connection by checking the ESP32's serial output for a confirmation message indicating successful connection to the Wi-Fi network
Section 2.2.3 Once connected to the Wi-Fi network, the ESP32 must handle POST requests to send data to an online database.	Execute a test script that triggers the ESP32 to send mock data to the online database using POST request. Verify the data by checking the entries. Measure the response time over multiple transmissions to ensure reliability and accuracy.

Table 3: R&V Table for WiFi and App Subsystem

[1] This quantity is derived from the minimum amount of water the user can manually set, which is 48oz, spread over 24 hours minus the minimum hours of sleep a user can input, which is 6 hours. This equates to 2.6oz per hour the user is awake. Note that the calculated water intake in a day based on user input of age, sex, and activity will be in the range of 60-80oz.