WateringU

Brendan Hernandez Roozen
University of Washington
4522 18th Ave NE
Seattle, WA 98105
(360) 708-7991
brendr4@uw.edu

Leonard Dul
University of Washington
1725 Bloomington Ave
Bremerton, WA, 98312
(516)554-3063
ldul@uw.edu

Sai Jayanth Kalisi
University of Washington
11322 177th Pl NE
Redmond, WA 98052
(425)614-7161
sakalisi@uw.edu

Karlee Wong
University of Washington
6315 28th AVE South
Seattle, WA 98108
(206)883-8855
Karleew@uw.edu

ABSTRACT

WateringU is a hydration tracking device. There are features to help assist the users to set goals and track their hydration. There are two components to WateringU. One is a stationary cup holder device that will scan the beverage the user is drinking. It will check to see if the beverage is water or not water. If it is water, and water was drunk then it will add to how much the user has already drunk. The other component is the app. It allows the user to add information about themselves to calculate the water consumption the user needs. It also features a progress chart and motivations so that the user will be able to meet their hydration goals.

1. INTRODUCTION

Hydration is something that is essential for human beings. Surprisingly, it is one of the most overlooked things in the present day. It has been found by multiple researchers that more than half of the human population is actually dehydrated. Severe dehydration can have major consequences and could be life threatening. Bottom line is that humans need to have water to function properly. To combat this, our team decided to design a product that will allow the user to be reminded to hydrate themselves when they are dehydrated. This paper will describe how our product is unique from similar products and why it will be a better addition to users' lives than those other products. This paper will also go into the technical details about our project, including all of the implementation details of the hardware and software components of both the microcontroller and the phone application. There will be sections to describe and explain all of the features that were included in our final product. Lastly, the paper will have a section about a prototype of the full WateringU product that the team has created and about how we would improve our design.

RELATED WORK

Our work is heavily related to smart water bottles. Water Bottles, and smart devices in general, have been a focus of our understanding. For example, the Hidrate Spark water bottles have excellent water tracking technology. LifeFuels Smart water bottles have the ability to remind you to take water based on your needs. And in EE 475, one of the groups is also doing a similar water intake tracking device.

Our product differs from these by the simple ability to tell whether the intake fluid is water or not. Such an act is extremely important, because without it, one could be tracking, for example, an alcoholic beverage as water intake, when, in reality, consuming such a beverage is considered detrimental to hydration amount.

While our app has the ability to distinguish liquids, currently, as seen in future sections of this report, we were not able to fully flush out this feature. Such an ability will be kept for future iterations of this project.

Additionally, unlike the Hidrate Spark and LifeFuels water bottles, our product is not considered a water bottle. Instead, it is a cup holder, which handles plastic see-through bottles for its use. This can be considered both an upside and a downside. The upside refers to the ability to charge this device repeatedly via a USB cable. The downside is that you will not be able to take this on the go. We did a complete pro-cons assessment of what to choose and ended up deciding that the cup holder idea would be more beneficial to the user.

Finally, we use a different sensor than the others to figure out the water inside. The Hidrate Spark and LifeFuels water bottles both use a combined weight/flow rate detector sensor value to determine how much water remains inside the water bottle. We settled for just the pressure/weight sensor due to time constraints.

2. TECHNICAL DETAILS

2.1 Overview

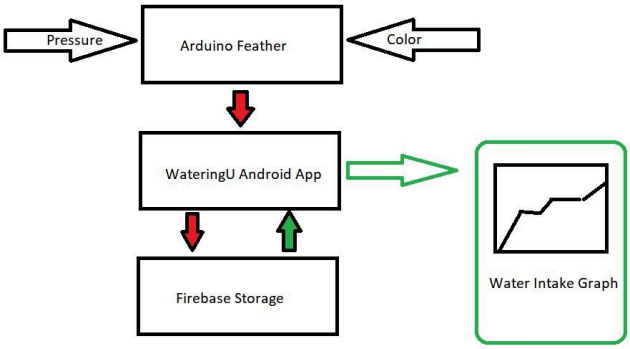


Figure 1 – System Flow

As per Figure 1, the WateringU uses three major components to help track water. From the hardware, the Arduino Feather listens to signals from the pressure and color sensor. These signals are then transmitted to the WateringU android application which further stored them into Firebase. The application then retrieves the data as it needs in order to implement that data in the app's tracking UI (as well as for other application features).

2.2 Theory of Operation

To track hydration, we need the mass of whatever liquid that exists in the water bottle, and the type of liquid that is inside the water bottle. In order to do these, we split the process into three different scopes of operation.

The first includes the color sensor. A color sensor was used to understand what the color of liquid would be. This is important to determine the type of liquid inside the water bottle. Say the water has input values of RGBC – 45, 50, 50, and 45. Color inputs within a certain threshold of these values would be considered values representing water. Conversely, if a liquid is out of the threshold, it is considered not water. This value would be taken into consideration when calculating the output.

Another scope of inquiry is the pressure sensor. The pressure sensor, which is capacitive, takes in a voltage, and another voltage is taken as an output voltage. Using these values, it is possible to calculate what the mass of the pressure on top of the capacitive pressure sensor was.

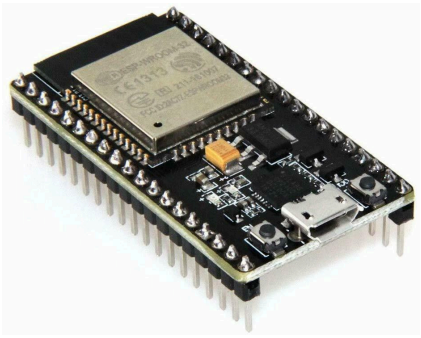
This pressure sensor value, in addition to the color sensor value, would be multiplied to find out all the information we need. The final scope revolves around the use of Bluetooth to transfer information to a cloud-based storage. The inbuilt Bluetooth module of the Feather Huzzah32 is used to transmit information by acting as a server. The android studio made app is acting as a client to store this value, thus completing most of our requirements.

The final requirement is completion of the product. This involves the 3D printing of the case and other holding elements to not only house all of the microcontrollers and wires, but also house the sensors in an organic manner.

2.3 Implementation Details

2.3.1 Hardware

The microcontroller that we selected for our device was the Arduino's ESP 32. The primary reason for the selection of this particular microcontroller was the low-power system, low-cost, and the robust dual-mode Bluetooth. The last feature was critical to this project, since it was required that our device possess bluetooth capability.The microcontroller contained the adequate memory storage that was needed to handle the data we were collecting from the sensors.



Bluetooth Implementation:

The arduino ESP 32 had two distinct bluetooth modes: Classical Bluetooth and BLE (Bluetooth low-energy). To understand why we implemented BLE instead of classical bluetooth, there is an important difference between the two modes. Classical Bluetooth is for continuous data transmission, and BLE is used specifically to transmit small amounts of data for small periods of time. From research, we found that most healthcare applications use BLE, substantiating our decision to implement BLE.

First task was to define the libraries,variables, and the UUID for the service and characteristic.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLE2902.h>
#include <string.h>

/*Bluetooth functionality */
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

BLEServer *pServer;
BLEService *pService;
BLECharacteristic *pCharacteristic;
boolean deviceConnected = true;


class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};
```

To implement BLE, the server and the client had to be created. For our product, the microcontroller acted as the server and the android app acted as the client (which made the implementation of BLE much easier). The server advertises its existence and contains the data that the client will read.This establishes a point-to-point connection.

```
  BLEDevice::init("EE 475 WateringU");
  pServer = BLEDevice::createServer();
  pService = pServer->createService(SERVICE_UUID);

```

The next was to define the characteristic. The characteristic is where the actual data is contained in the hierarchy. It has two attributes: characteristic declaration and characteristic value. We followed the characteristic by defining its properties, or formally known as descriptors, which contains the operations and procedures that can be used with the characteristic. For our application, we used READ, WRITE, and NOTIFY.

```

BLEDevice::init("EE 475 WateringU");
pServer = BLEDevice::createServer();
pService = pServer->createService(SERVICE_UUID);

pCharacteristic = pService->createCharacteristic(
                                       CHARACTERISTIC_UUID,
                                       BLECharacteristic::PROPERTY_READ |
                                       BLECharacteristic::PROPERTY_NOTIFY|
                                       BLECharacteristic::PROPERTY_WRITE
                                      );

```

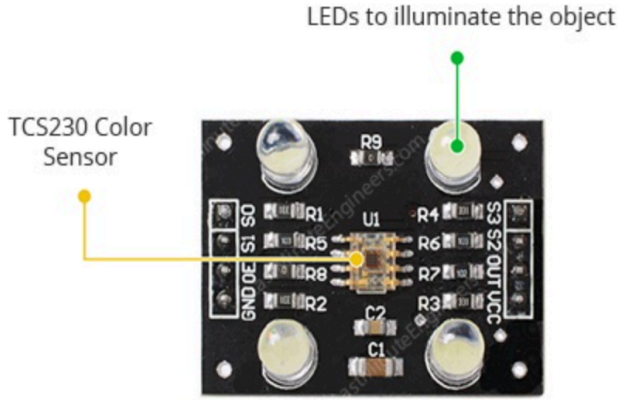
After finishing with setting up the characteristic and server, the next step was to start advertising the server:

```

pCharacteristic->setValue("Hello World");
pService->start();
//BLEAdvertising *pAdvertising = pServer->getAdvertising();  // this still is working for backward compatibility
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pAdvertising->setScanResponse(true);
pAdvertising->setMinPreferred(0x06);  // functions that help with iPhone connections issue
pAdvertising->setMinPreferred(0x12);
BLEDevice::startAdvertising();
Serial.println("Characteristic defined! Now you can read it in your phone!");
```

Sensor Implementation:

For our device, we used two sensors to gather data regarding the mass of water, and data that allowed the device to differentiate between water and non-water liquids.

TCS230 Color Sensor:



The TCS230 Color Sensor was used to obtain data allowed for the classification between water and non-water liquids. The first was to define the appropriate input and output pins:

```
#define s0 15
#define s1 33
#define s2 27
#define s3 12
#define outPin 32

```

The color sensor consists of four distinct colors: red, green, blue, and clear. It is from these readings that allowed us to create the threshold (see scheduler implementation) that defined if the liquid in the bottle is water or not. Pins S2 and S3 are in the input pins- and values for each pin are different for each color (see table below):

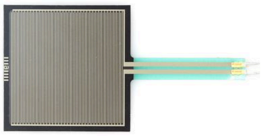
S2	S3	Photodiode type
LOW	LOW	Red
LOW	HIGH	Blue
HIGH	LOW	Clear (No filter)
HIGH	HIGH	Green

Each color has its own separate function that reads the value of each color and returns the value:

```

   //read red component
double red_light () {
   digitalWrite(s2, LOW);
   digitalWrite(s3, LOW);
   red = (double)pulseIn(outPin, LOW);
   return red;
}

   //read green component
double green_light () {
   digitalWrite(s2, HIGH);
   digitalWrite(s3, HIGH);
   green = (double)pulseIn(outPin, LOW);
   return green;
}

  //let's read blue component
double blue_light () {
   digitalWrite(s2, LOW);
   digitalWrite(s3, HIGH);
   blue = (double)pulseIn(outPin, LOW);
   return blue;
}

   //clear
double clear_light () {
   digitalWrite(s2, HIGH);
   digitalWrite(s3, LOW);
   Clear = (double)pulseIn(outPin, LOW);
   return Clear;
}
```

Pressure Sensor: Force-Sensitive Resistor:

To measure the mass of the liquid, we used a force-sensitive resistor as the mechanism to measure the mass of the liquid in the bottle.



The variable declarations and constants needed for the FSR are below:

```
/*Variables and Constants for sensor functionality */
int FSR_pin = A0;    // select the input pin for the potentiometer
double correctionValue = 0.5;
int avg_size = 10; // number of analog readings to average
float R_0 = 10000.0; // known resistor value in [Ohms]
float Vcc = 3.3; // supply voltage
```

Below is the function that extracts the mass of liquid in the bottle. There is a one-to-one conversion ratio between force (measured in g) and mL:

```
// Pressure function
int pressure_function() {
  int fsrADC = analogRead(FSR_pin);
  //Serial.println("input value: " + String(fsrADC));
  // If the FSR has no pressure, the resistance will be
  // near infinite. So the voltage should be near 0.
  if (fsrADC != 0) // If the analog reading is non-zero
  {
    // Use ADC reading to calculate voltage:
    float fsrV = fsrADC * Vcc / 4095.0;
    //Serial.println("input voltage handling " + String(fsrV));
    // Use voltage and static resistor value to
    // calculate FSR resistance:
    float fsrR = R_0 * (Vcc / fsrV - 1.0);

    //Serial.println("Resistance: " + String(fsrR) + " ohms");
    // Guesstimate force based on slopes in figure 3 of
    // FSR datasheet:
    float force;
    float fsrG = 1.0 / fsrR; // Calculate conductance
    // Break parabolic curve down into two linear slopes:
    if (fsrR <= 600)
      force = (fsrG - 0.00075) / 0.00000032639;
    else
      force =  fsrG / 0.000000642857;

    //Serial.println("Force: " + String(force) + " g");
    //Serial.println();

    return force;
  }

  return 0;
}
```

A FSR is essentially a capacitor, so the readings are reported in terms of voltage. Since we are using the ESP 32 microcontroller, the input voltage range is between 0 -3.3 V. analogRead takes these voltage readings and converts them to values ranging from 0-4095.

Scheduler Implementation:

The scheduler implementation was a round-robin style scheduler. The purpose behind this scheduler was twofold: one we were sending two different types of data. The second was due to the simplicity of our design, and so a round-robin scheduler would fit our purpose.

```
void loop() {
  if (deviceConnected) {
      digitalWrite(led_pin, LOW);
      // pressure sensor
      pressure_value = float(pressure_function());
      int mass = (int) pressure_value;


      //pCharacteristic->setValue(mass);
      //pCharacteristic->notify();

      delay(20);
      digitalWrite(led_pin, HIGH);
      // color sensor
       red = float (red_light());
       blue = float (blue_light());
       green = float (green_light());
       Clear = float (clear_light());

       Serial.print("R: " + String(red) + "G: " + String(green) + "B: " + String(blue) + "C: " + String(Clear) + " Pressure: ");

      digitalWrite(led_pin, HIGH);

      int water;
      if (red >100 && red < 150 && green > 100 && green < 190 && blue > 70 && blue < 160 && Clear > 25 && Clear < 50) {
        is_water = true;
        water = (int) is_water;
      } else is_water = false;

      delay(50);
      double waterColour;
      if(is_water) waterColour = 1;
      else waterColour = -1;
      int returnValue = pressure_value*waterColour/correctionValue;
      Serial.println(returnValue);
      pCharacteristic->setValue(returnValue);
      pCharacteristic->notify();
      //String x = "the liquid is water: " + String(int(is_water))  + " and the amount is" + String(pressure_value);

      digitalWrite(led_pin, LOW);
  }
  delay(3000);
```

Data will only be collected and transmitted if deviceConnected is true, otherwise the device will continue to sleep. Then the pressure function will be called, and data from the FSR will be collected. Then data is collected from the color sensor to, where the if statement determines whether or not the liquid is water or not- which is determined by the thresholds from the calibration of the sensor. Then the following logic combines the data and sends it to the app.

Water Bottle Detachment:

The water bottle detachment was done using AutoDesk Fusion 360 for this project. When creating the detachment we decided that the way we would be able to fit the microcontroller, light sensor, pressure sensor, and battery was to create different levels of platforms that could house different parts of the WateringU project. To achieve this the design shown in Fig 2. demonstrates the two sets of guide rails along the inner wall of the main body. Additional features include a port on the rear face of the model that allows for a USB-C charging wire to connect to the Arduino. Because of this charging port the Arduino would sit at the bottom of the inner container, along with the connection to its battery.

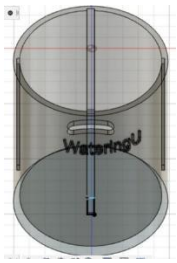


Figure 2 – Main body detachment

Subsequently a plate was created to be introduced into the guides. The plate had extrusions that would allow it to slide down to a position of 25.5 mm above the bottom of the WateringU detachment. This height was chosen to accommodate the total height of the Arduino on a breadboard and the wire connections that needed to be made on the breadboard to the other sensors. A hole was introduced so that these wires would be able to pass through to the next level upwards. On this plate the pressure sensor was seated in the center.

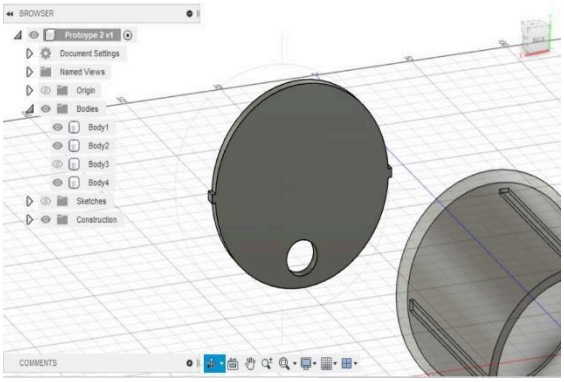


Figure 3 – Pressure Plate

To activate the pressure sensor a housing unit for the color sensor was created that needed to rest on the pressure sensor. For this two different segments were created, the main housing unit and the cover unit that allowed for an acrylic viewing window. Figures 4 and 5 demonstrate the bottom and top of the main housing unit respectively. The main housing unit includes extrusions for falling into the guide rails, as well as a hole for the color sensor connections. The bottom of the housing unit has a support pillar that rests on the pressure sensor by gravity, while the color sensor itself rests on the top platform.

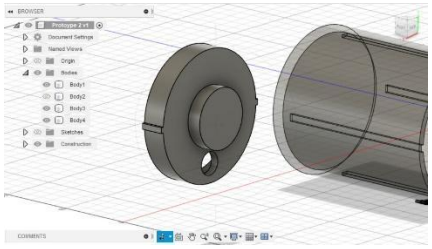


Figure 4 – Housing unit Bottom

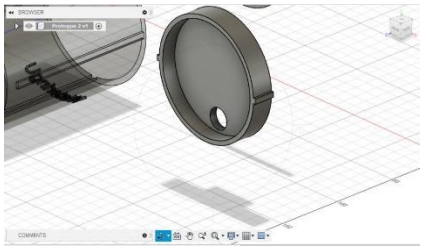


Figure 5 – Housing Unit Top

Figure 6 shows the cover unit with the viewing window. The viewing window was made to fit the dimensions of the color sensor (36 x 20.6 mm). An acrylic view window of the same dimensions was manufactured and given a height to be set flush with the inner base of the main housing unit platform.

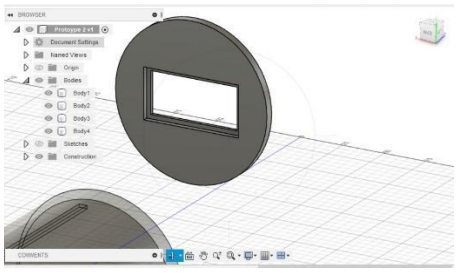


Figure 6 – Housing Unit Viewing Window

2.3.2 Software

Android Studio Application

For the application, the team decided on the use of Android Studio as a platform. The goal of the application was to develop a multi-activity app that permitted Bluetooth connection with the main objectives of tracking water intake on a hourly, daily, and monthly scale, and to update this on the graph based on the determination of it being only water. The architecture of the app was broken down into four activities : Bluetooth Connection page, Home Page, Settings Page, Graph page, and User Profile page (As seen in Fig.2). In addition to the BLE (Bluetooth), Firebase was introduced as the main storage space for the application. By creating a Firebase project on the website, the applicable JSON file was available to be added to the project folder and allow for data transfer to and from the android application.

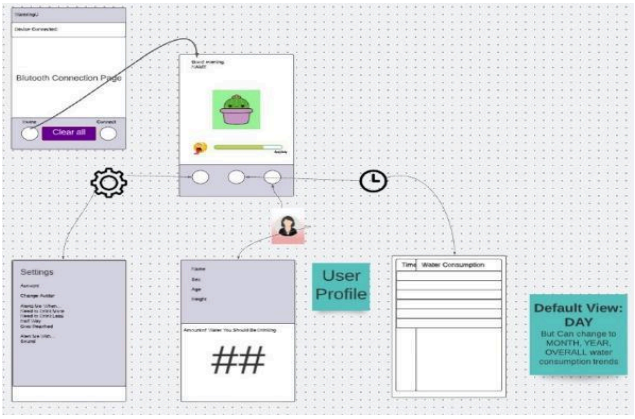


Figure 2 – App Architecture

Bluetooth Connection Page:

Upon starting the application this was designated to be the main page that the user first encounters as dictated by the AndoirdMainifest.XML file for the project. Three widgets were made available for the UI with the first being a homepage that would connect the user directly to the home page. The connect button would perform a scan for any available devices in the vicinity and allow for an android fragment to display with a list of available devices to select from and subscribe for BLE connection. The last feature on this page was the “Clear all” button which was introduced to clear all data that had been stored from the Arduino and stored in Firebase as well as the index of the stored data. This was introduced so that users could start tracking their water usage again if they chose to do so.

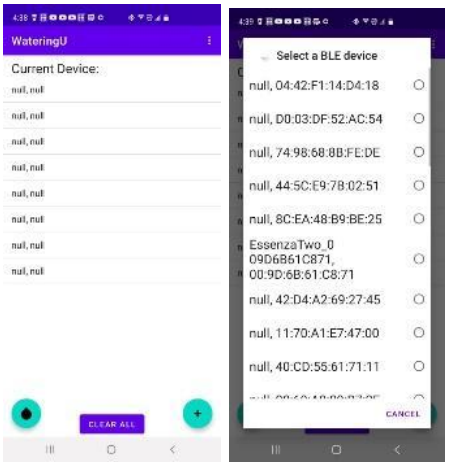


Figure 3 – Bluetooth Connection page

Home page:

The home page was designed to have four main features, with three of those being navigation widgets to the additional activities available in the application. On the bottom bar from left to right: Settings page, Graph Page, and User Profile Page. Each of these has its own onClickListener to navigate to its respective activity. The fourth main feature included a bar graph that would scale in real time by accessing data stored in Firebase from User Profile and displaying a percentage of water drank with respect to the total amount of water needed for the day. This data was also used to display a TextEdit that included the amount of water in mL that was required for the remainder of the day. Additional minor features included an Avatar image as well as incorporating the Users name which was retrieved from Firebase.

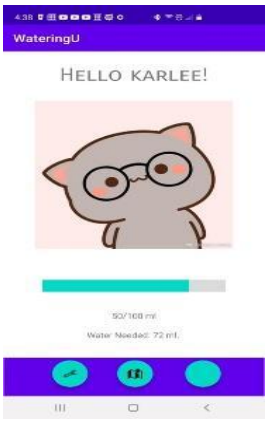


Figure 4 – Home page

Settings Page:

The settings page was designed to apply the current user's name from data in Firebase. It also incorporated two drop-down menus, the first called "Avatars" that has a drop-down menu for choosing between one of three avatars and updating them across the application. The second drop down, "Alarm", was for choosing between one of four alarm settings: Need to drink more, Need to drink less, Halfway, and Goal reached. Upon selecting one of these options a Toast will display a notification on the screen saying that this alarm setting has been chosen. A "submit" button is incorporated to make these changes available. Lastly there is a "Home button" made available to return to the Home page activity.

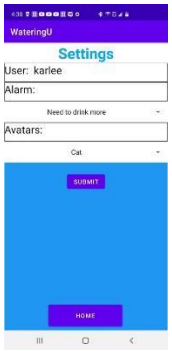


Figure 5 – Settings Page

Graph Page:

The graph page incorporates a linear graph that is intended to plot applicable data from Firebase as functions of water intake increase on x-axis intervals of days. In order to perform these actions, the following code was implemented –

```
Date date = new Date();
int dateINT = date.getMonth();
String finalMonth = monthName(dateINT);
int finalDay = totalDays(dateINT);
```

The first thing to consider in determining the graph was how it would place the appropriate number of days on the x-axis. To begin a date variable is created which is used to extract the month in INT format and stored in the dateInt variable. Creating a method called "totalDays", datInt was passed to return an int that would be used for x-axis scaling of our graph.

```
private int totalDays(int dateNumber){
    int returnINT;
    if (dateNumber == 0 ||dateNumber == 2 || dateNumber == 4 || dateNumber ==
6 ||
        dateNumber == 7 ||dateNumber == 9 || dateNumber ==11){
        returnINT = 31;
    }
    else if (dateNumber == 3 || dateNumber == 5 ||
            dateNumber == 8 ||dateNumber == 10){
        returnINT = 30;
    } else {
        returnINT = 28;
    }
    return returnINT;
}
```

To access the data sent from the Arduino specific calls were made to get the data path of the values we required, in this case "inputs" -> "Data Reading". The ValueEventListener allowed specific access to the information at the child of "Data Reading". IDnum was used to get that child's ID number and the "value" was a variable used to retrieve the specific data in that child representative of a change in pressure and positive if it was water. Converting value to an int produced our "yNumber" which would be used as our y-point on our graph in conjunction with the child's IDnum as the x-point. Because data was stored as positive or negative based on the color sensor readings (positive for water), we were only concerned with the positives for graphing. To pass on any negative values the "dayDiff" variable was created and used to keep track of the correct day value (x-axis), if the data from firebase was passed on. The graph was appended to include

IDNum as the x-point, yNumber as the y-point, and include finalDay as the variable x-axis length.

```
ref =
FirebaseDatabase.getInstance().getReference().child("inputs").child("Data
Reading");
ref.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        int dayDiff = 0;
        for(DataSnapshot snap: snapshot.getChildren()){
            int IDnum = Integer.valueOf(snap.getKey());
            String value = String.valueOf(snap.child("value").getValue());
            int yNumber = Integer.valueOf(value);
            if (yNumber < 0){
                dayDiff++;
                continue;
            } else if ((dayDiff != 0) && (yNumber>0)){
                IDnum = IDnum - dayDiff;
                series.appendData(new DataPoint(IDnum, yNumber), true,
finalDay);
                dayDiff = 0;
            } else {
                series.appendData(new DataPoint(IDnum, yNumber), true,
finalDay);
            }
        }
    }
    @Override
    public void onCancelled(@NonNull DatabaseError error) {
    }
});
```

User Profile Page:

The User Profile page was created to allow the user to input data specific to them to include: Name, sex, height, and weight. These are text widgets that all for user input from a keyboard and are updated to Firebase by clicking the update button below. Each of the fields is stored together in the Firebase app and are used by other activities, specifically the Home page which shows the progress bar and text displaying the amount of water still required for the day.

Figure 6 – User Profile Page

Upon pressing the update button, the java class for the User Profile UI would perform the following calculation –

```
String SexHelper  =
regSex.getEditableText().toString().toLow
erCase(Locale.ROOT);
int WeightMath  =
Integer.parseInt(regWeight.getEditableTe
xt().toString());
if (SexHelper == "male"){
    WaterNeed = (int) (WeightMath*0.75);
} else{
    WaterNeed = (int) (WeightMath*0.60);
}
```

This would retrieve the weight value inputted by the user through the UI as well as the gender in a String from the saved Firebase User Profile data. Based on the gender, the weight is manipulated to produce the amount of water (weight(lbs) x 0.75/0.60 (oz) (males/females)). This “WaterNeed” variable was used for the previously mentioned progress bar and text of water required for the day.

Firebase Architecture:

The Firebase Architecture was designed first to store the UserProfile data to include: height, name, sex, weight, and waterNeeded based on previous calculations. Additionally, as indicated by Figure 7, data was stored from the Arduino and tracked by numerically incrementing indexes under inputs > Data Reading and was additionally given a timestamp. The data used for tracking purposes fell under “newTest” and was a combination of pressure/color sensor readings.

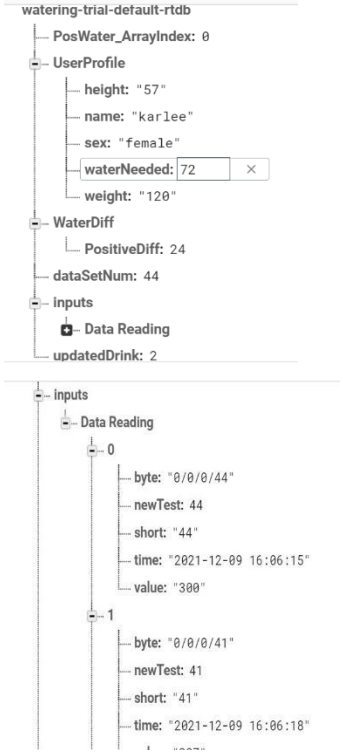


Figure 7 – Firebase Architecture

To convert this data from the Arduino into Java we needed to convert the 2-byte int from the Arduino into a signed 4-byte int the application could use. This was done by bit masking and shifting each byte received to the application by the appropriate amount, thus creating a positive or negative representation of the pressure sensor reading.

```
int leggo = ((data[0] & 0xff) |
        ((data[1] & 0xff) << 8)|
        ((data[2] & 0xff) << 16)|
        ((data[3] & 0xff) << 24));
database.child("inputs").child(datRead).c
hild(datReadID).child("newTest").setValue
(leggo);
```

In order to avoid the application overwriting previous data upon starting up again after being closed out, we needed to include a way to index the data and keep it at the last index of the last data that had been stored in Firebase. Using "dataSetNum" as the index keeper in Firebase we added code to increment a static variable "placeholder" after each new Data Reading was updated.

```
placeHolder++;
database.child("dataSetNum").setValue(pl
aceHolder);
```

After "dataSetNumber" had been updated by the placeholder we designed the Bluetooth to pull the existing placeholder value from Firebase and update it in the Main activity so that on the next Data Reading input from the sensors the values would be stored in a new child.

```
public void onStartConnect() {
    ref =
FirebaseDatabase.getInstance().getRefere
nce().child("dataSetNum");
    ref.addValueEventListener(new
ValueEventListener() {
        @Override
        public void onDataChange(@NonNull
DataSnapshot snapshot) {
            String newPlaceHolder =
String.valueOf(snapshot.getValue());
            placeHolder =
Integer.valueOf(newPlaceHolder);
        }
        @Override
        public void onCancelled(@NonNull
DatabaseError error) {

        }
    });
```

3. EVALUATION AND RESULTS

For the evaluation the team decided to analyze three specific components that comprised the majority of the WateringU project: the server (Arduino), the receiver (Android App), and that physical detachment.

Arduino:

By the end of production the team had successfully run trials to verify that both the color sensor and the pressure sensor had been calibrated and could establish a bluetooth connection to the Android Application. Data was transmitted as intended to the application, which meant that the Arduino had no need to schedule data transmissions between individual color and pressure readings (scheduling the signal transmission frequency did still occur). In addition the calibration of the sensors had been verified by checking Serial Monitor readings as well as data stored within Firebase. For the majority of the project timeline the Arduino was successful in its purpose and was deemed reliable. Last minute calibrations affected the data output from the Arduino and affected pressure sensor readings, thus making Android App see an almost constant pressure despite obvious changes in the water weight applied.

Android App:

This element proved to be the most difficult to work throughout the project. The ultimate product succeeded in establishing three key features initially. Bluetooth connection to the Ardunio was verified, data had been successfully stored to Firebase (storage/retrieval were available), and we had been able to create a plot of the data as a function of time. The water graph was one of the biggest highlights of the app and though it initially functioned, the team realized that graphing the data alone would not lead to accurate water tracking. To mitigate this the team changed the existing code to retrieve the raw stored data from the sensors and store it again based on positive values (indicating water). Additionally this new storage would not specifically store the applicable raw data, but needed a single data point per unit time (in our case this was days). In the process of attempting to create and store a new data point from various stored data sets, the team lost the graphing functionality and could not produce the deliverable by the final presentation date. Secondary shortcomings also included the alarm features that were meant to be operable. The team spent the majority of the time attempting to fix the graph functionality and lost sight of assigning the alarm drop down menu options their respective alarm set points.

Detachment:

The detachment worked averagely well for the majority of its components, however this was only true when none of the components had been wired through the holes in the plates and housing unit. Several concerns that did arise with the final project were that first, the entire detachment may have had too wide a diameter. While the initial goal was to create a portable unit, due to the size of the detachment we had to adjust the model to have an intended purpose of being a stationary unit. The issue that affected the final product the most was not taking into consideration that way the wires would kink after they pass through a hole. This made it more difficult to keep the sensors stationary, especially the color sensor which had a vertical range of motion. The kink also caused the plate it passed through to have a tendency to enter the main body at an angle (creating friction against the guide rails). The final issue we had with the detachment was that the charging port had been sized too large. While this presented no specific issues during testing, we felt that it left too much access to the Arduino. Though we did not predict water could get it from outside the device, the possibility existed through the charging port, so this is an issue that would need more consideration in the future.

4. DISCUSSION

Overall, our product performed up to most of the functionality that was outlined in our Project Proposal (before our demonstration session). Our product was able to take in readings from both sensors, sent them to Firebase for storage, and then to the app. We got plausible readings from the sensors, and the

bluetooth connection was able to send them accordingly at their times in our scheduler. Looking to the future, there is major room for considerable upgrades and improvements to the overall product.

The first step in our next step is to redesign the container. Our prototype was designed for parts to slide into place, but that revealed an unexpected problem. Our sensors were not put into place, and hence they slid around while we used our device. Our current design consisted of the color sensor sitting on top of the pressure sensor, and that caused the physical device to be unstable, too. Figuring out how to properly place both sensors to have a stable device would be an important first step- where the sensors will be permanently in place.

The next step is to finish the functionality of our app. We were unable to get the graph that tracked how much water the user consumed over a given day- we ran into several bugs in the code and we were unable to debug. Figuring out the flaws in the code for the graph is a vital component of the app that we did not work properly.

Once our product is augmented to improve its functionality, the next step is to add additional functionality to the device-specifically in terms of the app. One major addition we want to add is the ability of the app to notify you to drink water if you are not consuming an adequate amount throughout the day- notifications would be a major addition. Additionally, the app should have the ability to track both water and non-water substances.

4.1 Subsections

5. CONCLUSION

Our goal, when starting this project, was to create a device to support the water intake a human should need. While our prototype had some drawbacks to what we initially set out to do, it is paramount to know that our project could be considered a success, in that our device is functioning as a minimum viable product. This means that there is a lot to be done before the product is considered competitive in the market.

There is still so much that can be done, and such thoughts will be kept in consideration for the next iteration of prototyping and product development in this project.

We know that our app and product combination is not competitive in the market, yet for our time constraints, we believe that we did the best that we could do, and hope that those who take up a project regarding the hydration of human beings will be able to use our experience as a building block to success.

6. REFERENCES

[1] Interfacing TCS230/TCS3200 Color Sensor with Arduino. https://lastminuteengineers.com/tcs230-tcs3200-color-sensor-arduino-tutorial/

[2] Force Sensitive Resistor Hookup Guide. https://learn.sparkfun.com/tutorials/force-sensitive-resistor-hookup-guide?_ga=2.203566551.658500376.1638726360-584771399.1638726360#example-arduino-sketch

[3] Getting Started with ESP32 Bluetooth Low Energy (BLE) on Arduino.https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/.

[4] Interfacing Force Sensing (FSR) with https://lastminuteengineers.com/fsr-arduino-tutorial/.

[5] Connect Bluetooth Devices

https://developer.android.com/guide/topics/connectivity/bluetooth/connect-bluetooth-devices

[6] Sketching and extruding on a curved surface

https://forums.autodesk.com/t5/fusion-360-design-validate/sketching-and-extruding-on-curved-surface/td-p/6290863

[7] Installation and Setup on Android https://firebase.google.com/docs/database/android/start

[8] Read and Write Data on Android Studio

https://firebase.google.com/docs/database/android/read-and-write

.