MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

# SMART WATER LEVEL MEASURING APPLICATION WITH REAL-TIME FEEDBACK

COMPUTER NETWORKS SEMESTER PROJECT

Authors: **Nagy Timea, Nemes Raluca, Oprea Florin Octavian**

Scientific coordinator: **Ioan Valentin Sita**

**2024**

# Table of contents

# 1 Introduction

## 1.1 Project context

In the context of promoting healthy hydration habits, this project focuses on developing a system that measures the water level in a bottle and provides users with a count of the number of cups consumed. The aim is to encourage individuals to maintain adequate water intake throughout the day. This smart water monitoring system offers a convenient way for users to track their hydration levels, promoting overall well-being.

## 1.2 Operation

Users fill the bottle with water, and the water level sensor continuously monitors the water level. As a person drinks from the bottle, the system updates the water level and increments the cup count accordingly. The Android application provides real-time feedback, allowing users to track their water intake throughout the day. The system is designed to be user-friendly and seamlessly integrate into daily routines.

## 1.3 Objectives

The primary objective of this project is to create a user-friendly system that accurately measures the water level in a bottle and keeps a count of consumed cups. The system aims to raise awareness about the importance of staying hydrated and provides users with a reliable metric for tracking their daily water intake.

Additionally, the project aims to help people reach their fitness objectives by reminding them to drink enough water during the day

Our project helps in promoting hydration awareness: Users receive **real-time feedback** on their water consumption, promoting awareness and encouraging healthy hydration habits.

The cup count feature serves as a motivational tool, incentivizing users to achieve their daily hydration goals.

## 1.4 Specifications

The project utilizes an Arduino Uno microcontroller to interact with the water level sensor and establish a Bluetooth connection with the Android application. The Android application, developed using Android Studio, acts as the user interface, displaying relevant information and allowing users to interact with the system.

The hardware components of the project are an Arduino Uno Microcontroller, a HC-05 Bluetooth module, a HC-SR04 Ultrasonic distance sensor, various connecting wires, and a breadboard.

The Android application displays the current water level, the count of consumed cups, and provides users with a visual representation of their hydration progress. The system ensures ease of use and aims to serve as a motivational tool for individuals to maintain optimal hydration levels.

The system utilizes Bluetooth connectivity for seamless communication between the microcontroller and the Android application.

# 2 Bibliographic research

## 2.1 In the context of IoT

In the context of Internet of Things (IoT) applications, the development of a Smart Water Level Measuring App is an absolute need in todays fitness and health oriented world.

IoT refers to the network of physical devices embedded with sensors, software, and connectivity, allowing them to exchange and collect data. The Smart Water Level Measuring App aligns with the principles of IoT by integrating sensors and connectivity to monitor and analyze water consumption.

While IoT literature commonly explores diverse applications, the specific focus on water monitoring applications reflects a growing interest in leveraging IoT for health-related purposes. Tracking and managing water intake using connected devices is a manifestation of how IoT technology can be applied to promote individual well-being.

This innovative solution highlights the importance of sensor integration, data processing, and real-time connectivity in the development process of todays worlds applications.

## 2.2 In the context of Healthcare

Many people are adopting healthier lifestyles, including fitness routines, balanced diets, and improved sleep patterns. Health apps support individuals in managing and optimizing their lifestyle choices by offering guidance, tracking progress, and providing personalized recommendations. Health apps bring healthcare resources and information directly to users' fingertips, offering a level of convenience that traditional healthcare services may lack. This accessibility encourages individuals to actively engage in monitoring and managing their health.

Health apps contribute to raising awareness about personal well-being and preventive healthcare measures. They empower individuals to take a proactive role in managing their health by providing information, tracking tools, and reminders for healthy behaviors.

# 3 Implementation

## 3.1 Hardware configuration

The circuit diagram which will be implemented using physical components was implemented using Tinkercad and can be seen in Fig. 3.1.1.

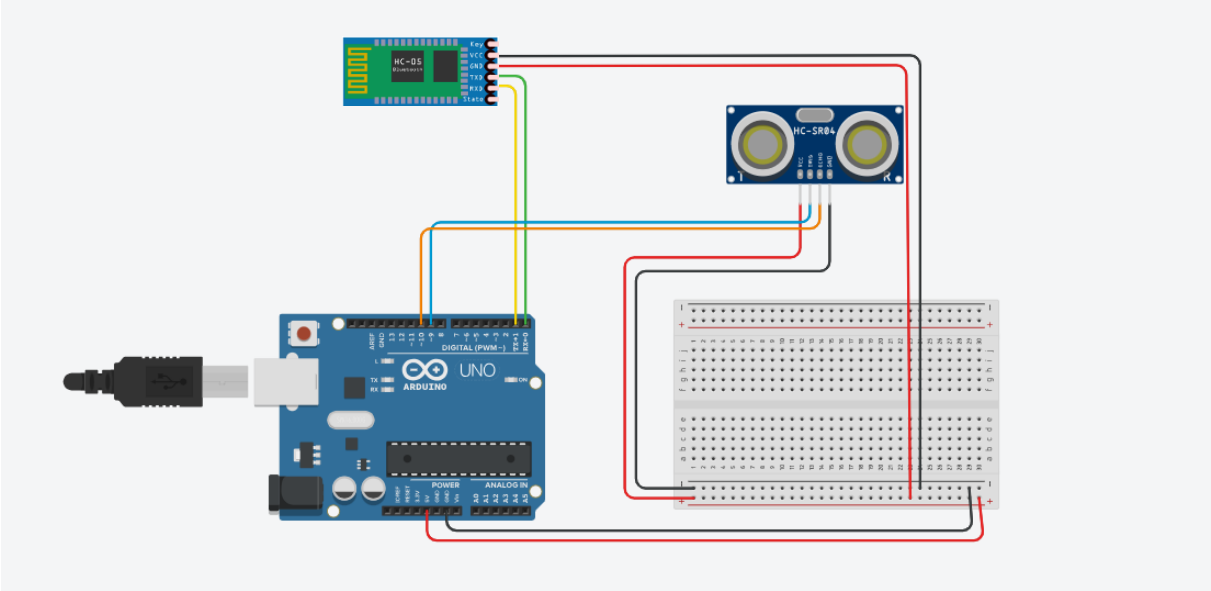



*Fig. 3.1.1. Smart Water Level Measuring – circuit and wiring diagram*

The components used in the hardware configuration can be seen in fig. 3.1.2 – 3.1.5:

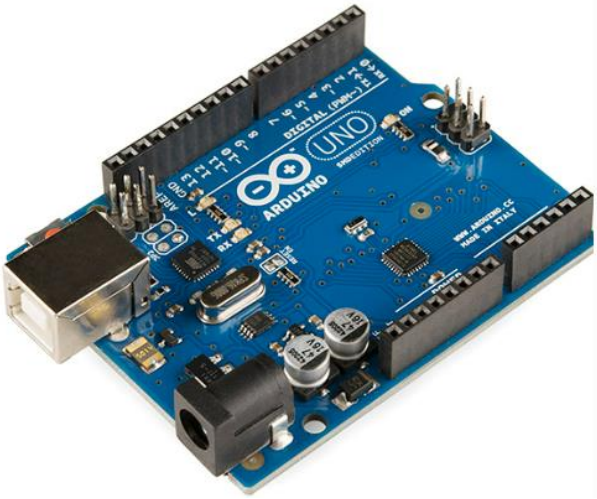



*Fig. 3.1.2. Arduino Uno*


*Fig. 3.1.3. Ultrasonic distance sensor*


*Fig. 3.1.4. Bluetooth module HC-05*

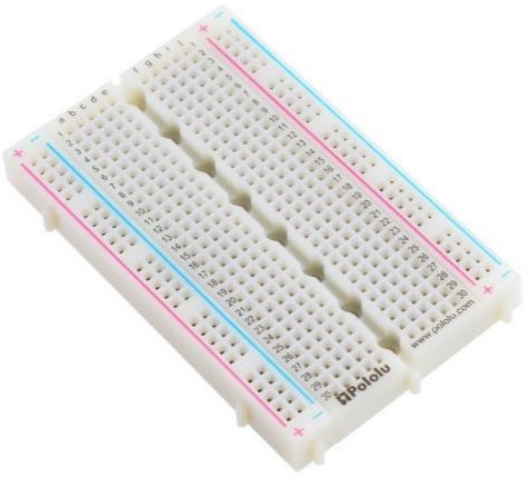
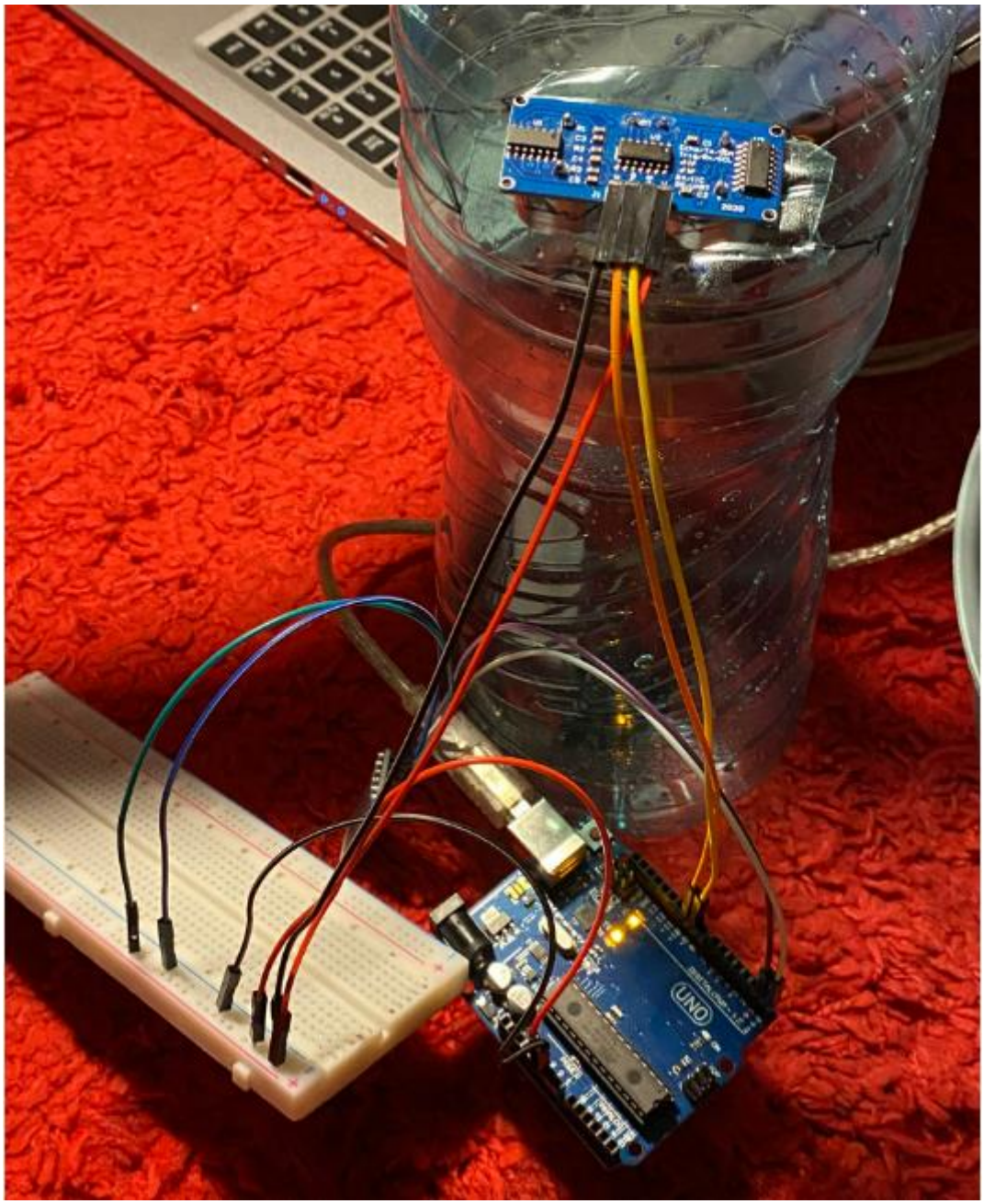



*Fig. 3.1.5. Breadboard*



## 3.2 Arduino IDE + Bluetooth Connection

The development environment used for for writing the first part of the code in this project is Arduino IDE. The ultrasonic sensor measures distance by sending a sound wave (ultrasonic pulse) and measuring the time it takes for the pulse to bounce back after hitting an object. The Arduino calculates the distance based on the time taken for the round trip of the pulse.

The following part is focused on the main program (for the central Arduino board).

```
// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;
```

Variables trigPin and echoPin are defined to specify the pins connected to the trigger and echo pins of the ultrasonic sensor.

```
// defines variables
long duration;
int distance;
```

Variable duration holds the time taken for the ultrasonic pulse to travel and return and distance stores the calculated distance based on the duration.

```
void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(38400);
}
```

The command pinMode is used to set trigPin as an output and echoPin as an input.

Serial.begin initializes serial communication for Bluetooth communication with baud rate 38400. Bluetooth communication is set up in this part.

```
void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;
  Serial.print("Distance: ");
  Serial.println(distance);
  delay(1000);
}
```

The loop performs the following steps:

- Sends a short pulse to the ultrasonic sensor (trigPin).
- Measures the time for the pulse to return (echoPin).
- Calculates the distance using the speed of sound (assumed at 0.034 cm/µs).
- Prints the distance to the Serial Monitor.
- Adds a delay of 1000 milliseconds (1 second).

## 3.3 Android Studio Application Development

Android Studio is a specialized software development environment designed to empower developers in creating applications for Android devices. Developed by Google, it serves as the primary Integrated Development Environment (IDE) for Android app development.

Our project utilized one of Android Studio's project templates, the Basic Activity. It provided a basic template and a pre-configured structure including essential files and components for easier project setup. In Android Studio, we wrote our app's code using Java.

The main fragment is WaterReminder.java. Here we initialised the Bluetooth communication:

```
    // Bluetooth variables
    private BluetoothAdapter bluetoothAdapter = null;
    private BluetoothSocket bluetoothSocket = null;
    private InputStream inputStream = null;
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB"); // SPP UUID
    private static final String DEVICE_ADDRESS = "98:D3:31:F6:8C:F6";
    private static final int REQUEST_BLUETOOTH_PERMISSION = 1; // You can use
any integer value
    private boolean isBluetoothConnected = false;

    // Handler to update UI with received Bluetooth data
    private final Handler handler = new Handler(new Handler.Callback() {
        @Override
        public boolean handleMessage(Message msg) {
            // Handle the received data here
            String receivedData = (String) msg.obj;
            updateUI(receivedData);
            return true;
        }
    });
```

In this code, the `DEVICE_ADDRESS = "98:D3:31:F6:8C:F6"` is the address of our Bluetooth module.

The setup of the variables visible on the interface is done by a bindidng method as follows:

```
        binding.idTextTotalCups.setText(String.valueOf(totalCups));
        binding.idTextTotalCups.setInputType(InputType.TYPE_NULL);
        binding.idTextTotalCups.setFocusable(false);
        binding.idTextTotalCups.setFocusableInTouchMode(false);
```

After initialisation, the verification of the bluetooth connection is needed:

```
// Bluetooth setup
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
        if (bluetoothAdapter == null) {
            Toast.makeText(requireContext(), "Device does not support
Bluetooth", Toast.LENGTH_SHORT).show();
            return;
        }

        // Check if Bluetooth permission is granted
        if (ContextCompat.checkSelfPermission(requireContext(),
Manifest.permission.BLUETOOTH_CONNECT) !=
                PackageManager.PERMISSION_GRANTED) {
            // Request Bluetooth permission if not granted
            requestPermissions(new
String[]{Manifest.permission.BLUETOOTH_CONNECT},
REQUEST_BLUETOOTH_PERMISSION);
            return;
        }

        // Replace DEVICE_ADDRESS with your Arduino's Bluetooth address
        BluetoothDevice device =
bluetoothAdapter.getRemoteDevice(DEVICE_ADDRESS);

        try {
            bluetoothSocket =
device.createRfcommSocketToServiceRecord(MY_UUID);
            bluetoothSocket.connect();
            inputStream = bluetoothSocket.getInputStream();
            isBluetoothConnected = true;
            readData();
        } catch (IOException e) {
            e.printStackTrace();
            Toast.makeText(requireContext(), "Bluetooth connection failed",
Toast.LENGTH_SHORT).show();
        }
```

If the connection is OK, we read the data:

```
private void readData() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                byte[] buffer = new byte[1024];
                int bytes;

                while (true) {
                    try {
                        bytes = inputStream.read(buffer);
                        String receivedData = new String(buffer, 0, bytes);
                        handler.obtainMessage(0, receivedData).sendToTarget();
```

```
                    } catch (IOException e) {
                        break;
                    }
                }
            }
        }).start();
    }
```

And then Update the User Interface and calculate the appropriate percentages:

```
private void updateUI(String receivedData) {
        requireActivity().runOnUiThread(new Runnable() {
            @Override
            public void run() {
                //max 25 - pahar gol
                //min 2 - pahar plin
                try {
                    String cleanedData = receivedData.trim();
                    int dataValue = Integer.parseInt(cleanedData);
                    // Map the range [2, 25] to [100%, 0%]
                    double percentage = 100 - (dataValue/20.0f) * 100;

                    if(percentage < 0)
                        percentage = 0;
                    if(percentage < 10 && !empty)
                    {
                        System.out.println("Increase cup consumption");
                        empty = true;
                        totCups++;
                    }
                    if(percentage >= 85)
                        empty = false;
                    binding.currentlyInCup.setText((int)percentage + "%
water");
                    System.out.println(dataValue + " " + percentage + "%");
                    binding.idTextTotalCups.setText(String.valueOf(totCups));
                } catch (NumberFormatException e) {
                    binding.currentlyInCup.setText("Invalid data");
                }
            }
        });
    }
```

It trims and converts the data to an integer, calculates a percentage, and adjusts UI components accordingly. Conditions handle special cases, and exceptions are caught for invalid data, displaying an appropriate message. It counts the number of cups consumed. One cup is considered consumed if the bottle is first empty, and then refilled.

The code can be downloaded from GitHub:

[NTimea302/SmartWaterLevelMeasuring (github.com)]

# 4 User's manual

## 4.1 Installing the IDE and setting up the hardware

Arduino IDE can be downloaded from https://www.arduino.cc/en/software. The user chooses the version corresponding to their operating system and installs it based on the instructions in the installation wizard. After this step is finished, the code file provided in the project can be opened as a new sketch inside the IDE.

The hardware configuration can be seen in fig. 3.1.1 and in fig. 3.1.10. The user will also need some cables to connect the components and an Ethernet cable to connect the board to the Internet.

After all the parts are interconnected, the user will have to compile the program and upload it to the board by pressing the Verify and Upload buttons in the IDE (Fig. 4.1.1.). At this point, the application should start and the initial message can be seen in the serial monitor.

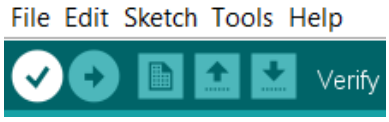



*Fig. 4.1.1. Arduino IDE menu*

## 4.2 Installing AndroidStudio

To get started with Android Studio, download it from Download Android Studio & App Tools - Android Developers. Choose the version that matches your operating system and follow the installation instructions provided. Once installed, open the project's code files within Android Studio to begin development. As mentioned above, the application's code can be downloaded from GitHub: NTimea302/SmartWaterLevelMeasuring (github.com).

After opening the project files in Android Studio and writing the appropriate code in Arduino IDE, the user needs to set up ptoper bluetooth communication. After the wiring is done, the address of the bluetooth module can be found either written on it, or opening device details on bluetooth on any android phone. After the connection is made, both codes need to be running. The AndroidStudio Application can re run using the top right build and run button as seen in Fig 4.2.1:


*Fig. 4.2.1. AndroidStudio run*

To run your Android app on a device, follow these steps:

- Ensure you have an Android device (phone or tablet) and a data cable to connect it to your computer.

- For Linux or Windows, check the additional steps in the "Run Apps on a Hardware Device" documentation, including installing USB drivers if required.
- Enable USB Debugging on your Android device:
- On Android 4.2 and higher, go to Settings > About phone, tap Build number seven times, return to Settings, and enable Developer options.
- In Developer options, enable USB Debugging.
- Connect your device to your computer using a USB cable. If prompted, allow USB debugging on your device.
- In Android Studio, click the Run icon in the toolbar. The Select Deployment Target dialog appears, showing available emulators and connected devices.
- Select your device and click OK. Android Studio installs and runs the app on your connected device.

After both codes are running and proper bluetooth communication is set up, the interface will look like Fig 4.2.2 where the current water level is measured and the total cups consumed.

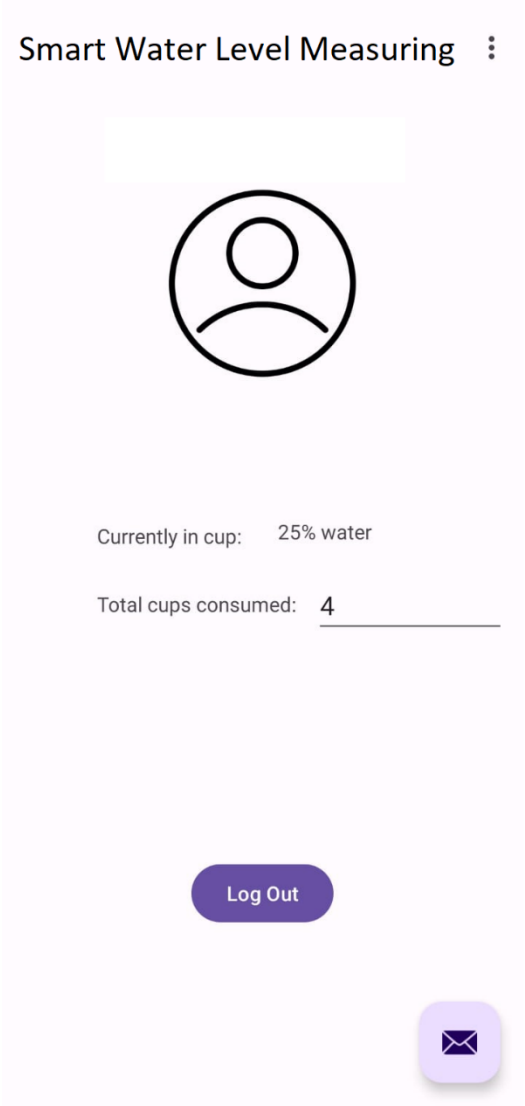



*Fig. 4.2.2. Mobile Application*

When the application runs correctly, the console in Android Studio should look like the on in Fig 4.2.3 where the measured distance and the according water level is printed.

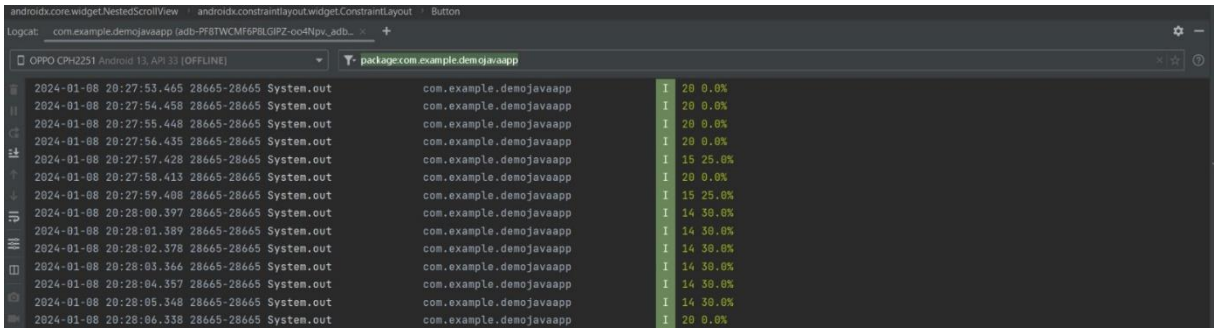



*Fig. 4.2.2. Android Studio Console*

# 5 Conclusions

## 5.1 Results

Having the hardware components listed in section 3.1 and following the diagram in Fig.3.1.1, with the help of the instructions in the User Manual in Chapter 4, the application can be implemented with the code from section 3.2. and tested by anyone.

## 5.2 Possibilities of improvement/further development

Some possibilities of further improvement of this project might include:

- Login to Personal Account
- Smart Reminders: Implementation of smart reminders to prompt users to drink water at regular intervals.
- Customizable Goals: Allow users to set personalized hydration goals based on individual needs and preferences.
- Personalised goals based on users weight, current fitness level, etc

# 6 Bibliography

[1] Build Your First Android App in Java

[2] Arduino Hc 05 Bluetooth Module Interfacing With Arduino Uno | Ard... (electronicwings.com)

[3] Getting Started with the HC-SR04 Ultrasonic sensor | Arduino Project Hub

[4] Connect Bluetooth devices | Connectivity | Android Developers