IE2090

Professional Engineering Practice and Industrial Management

2nd Year, 2nd Semester

Final Report

Smart Breathalyzer Device

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

2024.06.18

Declaration

We declare that this project report or part of it was not a copy of a document done by any organization, university any other institute or a previous student project group at SLIIT and was not copied from the Internet or other sources.

Project Details

Project Title	Smart Breathalyzer Device
Project ID	PEP_07

Group Members

Reg. No	**Name**	**Signature**
IT22088000	W. Y. M. G. Thamasha	
IT22073228	D. W. A. I. Abeynayake	
IT22073082	E. M. B. V. Bandara	
IT22081520	K. M. D. N. B. Ariyasena	

Abstract

The Smart Breathalyzer Device project addresses one of the major issues of present days, drunk driving, which leads to numerous fatalities and injuries that happen because of road accidents. As a solution for this, we developed an IoT based smart breathalyzer device that accurately measures a person's breath alcohol level using a MQ-3 alcohol gas sensor and notify the user through a user-friendly mobile application. The device also has an alert system which consists of a RGB led and a buzzer which activates when the alcohol is detected in the person's breath. Mobile application displays the alcohol percentage with a message displaying whether it is safe to drive or not. Communication of the components and mobile application happens through an ESP8266 Wi-Fi module. Our development process included thorough research and requirement analysis, design and prototyping, and the implementation of sensors and communication modules. The device helps the individual drivers to make informed decisions and promotes responsible drinking behavior. Additionally, the device serves as an effective tool to law enforcement to quickly identify and manage drunk driving incidents. The project was successfully completed within the given period and within the estimated budget, and the results demonstrate the device's potential to reduce drunk driving incidents, ensuring and enhancing road safety.

Key Words

- MQ-3 sensor
- Blynk App
- ESP8266 Wi-Fi module
- Threshold value

Acknowledgement

We would like to convey our heartfelt gratitude to our lecturer Ms. Narmad Gamage, who gave us this great opportunity to do this project, giving invaluable advice and assistance in completing our project. We extend our sincere gratitude to all our group members for their unwavering dedication. Each individual's efforts have been nothing short of commendable, significantly contributing to the resounding success of our project. We would also like to thank all of the other supporting personnel who assisted us by supplying the equipment that was essential and vital, without which we would not have been able to perform efficiently on this project.

Table of Contents

List of Figures

List of Tables

1. Introduction

Alcohol-impaired driving is still a serious issue which causes road accidents with many fatalities and injuries every year globally. Statistics show that over 21% of vehicle accidents are caused by drunk driving. The goal is to create a product that detects alcohol through breath. The IoT Alcohol Breathalyzer Device aims to be easy-to-use, accurate and portable. The device can therefore protect more people from driving while intoxicated by detecting alcohol in a person's breath.

1.1 Problem Statement

Drunk driving poses a grave danger to public safety as it results in many fatal traffic mishaps and injuries. Thus, the IoT Alcohol Breathalyzer Device is seen as such a technological invention for this purpose. It is responsible for accurately determining how much alcohol is present in someone's breath before issuing instant notifications on these findings. These warnings are essential in preventing impaired drivers from operating automobiles thereby reducing accident risks together with related human costs.

1.2 Product Scope

The Smart Breathalyzer Device project aims to enhance road safety by developing a device that analyzes a person's breath for alcohol levels, providing timely alerts to prevent accidents caused by drunk driving. The project includes the creation of a user-friendly mobile application to accompany the device, catering to both drivers and law enforcement officers. The device will detect the alcohol level and notify the user through the mobile application. Utilizing available sensors, the device ensures precise alcohol detection for personal and professional use. Better security, ease of use, mental clarity, and energy economy are among the advantages. Reliability, compatibility, security features, and an easy-to-use interface are among the goals of the software. Innovating, customer-focused solutions and utilizing technology to tackle new market trends are corporate goals that the system supports.

In-scope

- Alcohol measurement in breath: It employs high precision sensors for alcohol level detection.
- Instant alerts sent through cell phone app: Whenever the blood alcohol levels exceed the allowed thresholds, the device sends updates to authorized users and user via mobile applications.
- Easy-to-use display and notifications: The gadget's layout incorporates an intuitive feature that makes it possible for anyone who wants to have a record of his drinking as well as get timely warnings.

Out-scope

- The Smart Alcohol Breathalyzer Device serves the above purposes only.
- The device does not handle some aspects such as integrative care totality which means monitoring factors other than alcoholic tendencies are outside its purview.
- It doesn't work with vehicle systems; it functions independently without being part of any vehicle control system.
- More complex analytics beyond initial detection and alerting: As such, there will be more concentration on basic data capturing and notification of presence of alcohol rather than advanced business intelligence capabilities.

1.3 Project Report Structure

The project report for the Smart Breathalyzer Device is structured to offer a detailed overview of the project's development stages, from the initial concept to the final implementation. It starts with an introduction that defines the problem, outlines the project's objectives, and explains the structure of the report. The Requirements and Analysis section follows, identifying the specific features and needs of the device to ensure it meets performance and safety criteria. The Design section outlines the device's architecture, covering both hardware and software design aspects. The Implementation section provides a detailed account of the construction and programming of the device, highlighting significant technical steps and challenges faced. The Testing section assesses the device's functionality and effectiveness, ensuring all requirements are met through comprehensive testing. The report concludes with a summary of the project's results, insights gained, and suggestions for future improvements, offering a critical analysis of the project's impact and opportunities for further advancement.

2. Methodology

2.1 Requirements and Analysis

2.1.1 Functional Requirements

Table 1: Functional Requirement 1

F1	Alcohol Level Detection
Input	Positive sensor reading from the mp3 alcohol gas sensor
Process	Identifying whether the alcohol level is higher or lower that the legal limit
Output	Detect the alcohol level
Definition	Legal limit – The maximum amount of alcohol allowed before driving (0.08%)

Table 2: Functional Requirement 2

F2	Implementing Wi-Fi Communication
Input	Digital alcohol concentration data from the ESP module
Process	Format the alcohol concentration data for transmission over Wi-Fi and establish a Wi-Fi connection with the mobile application
Output	Send the formatted data to the mobile application

Table 3: Functional Requirement 3

F3	Displaying the data in the mobile app interface
Input	Alcohol concentration data received via Wi-Fi communication
Process	Format the data for display on the mobile app interface
Output	Display the alcohol level with a message which indicates whether it is safe to drive or not

Table 4: Functional Requirement 4

F4	LED Alert Activation
Input	Sensor reading from the mp3 alcohol gas sensor
Process	Compare the alcohol level with the legal limit
Output	Active the LED to turn on and blink if the alcohol level exceeds the legal limit

Table 5: Functional Requirement 5

F5	Speaker Alert Activation
Input	Sensor reading from the mq3 alcohol gas sensor
Process	Compare the alcohol level with the legal limit
Output	Emit a beep sound from the speaker if the alcohol level exceeds the legal limit

Table 6: Functional Requirement 6

F6	Displaying the Battery Percentage
Input	Battery voltage level measured by the ESP module
Process	Convert the battery voltage level to a percentage representing the remaining battery capacity
Output	Display the calculated battery percentage on the digital screen of the device

2.1.2 Non-Functional Requirements

Performance Requirements

- Alcohol sensor: The sensor must provide accurate and reliable alcohol level readings to ensure the device's effectiveness in detecting impaired driving.
- Wi-fi module: The Wi-Fi module should enable seamless and stable communication between the breathalyzer device and the mobile application for real-time data transmission.
- Mobile Application: The mobile application must offer a user-friendly interface and prompt notifications to ensure users receive timely alerts about their alcohol levels.

Safety Requirements

- The smart breathalyzer device should be designed to minimize physical harm to users and property occupants, ensuring secure mounting of hardware components like sensors and nozzle. Safety measures, such as isolation of the user’s breath from other external influences should prevent accidental readings and false positives.

- The device is designed to withstand environmental factors such as temperature variations (-10°C to 40°C), humidity levels (10% to 90% RH), and mechanical shocks (up to 1 meter drop) without compromising its functionality or accuracy.

Security Requirements

The intelligent breathalyzer device must possess robust security features, ensuring impregnability against external interference or tampering attempts. Its design should incorporate advanced encryption protocols and authentication mechanisms, guaranteeing the integrity of data collection and analysis processes. This ensures utmost reliability in determining accurate blood alcohol content levels, fostering trust in its results among users and authorities alike.

Software Quality Attributes

- The smart breathalyzer device can achieve a reliability metric of at least 99.9%, ensuring that it accurately detects blood alcohol content levels with minimal false positives or false negatives during regular operation.

- The device facilitates easy maintenance and servicing, with a mean time to repair (MTTR) of no more than 30 minutes for common issues, allowing for swift resolution and minimal downtime.

- The device's user interface adheres to established usability guidelines, achieving a System Usability Scale (SUS) score of at least 75 out of 100, ensuring intuitive operation and minimizing user errors during testing procedures.

- The device is lightweight and compact, weighing no more than 300 grams and easily fitting into standard carrying cases or vehicle compartments, facilitating convenient transport and deployment in various settings.

- The device is compatible with commonly used operating systems (e.g., iOS, Android) and communication protocols (e.g., Bluetooth, Wi-Fi), enabling seamless integration with existing infrastructure and enabling data sharing across platforms.

- The device supports firmware updates and software upgrades to accommodate evolving regulatory requirements and technological advancements, ensuring long-term viability and compatibility with future standards.

- The device maintains a high availability rate of at least 99%, minimizing downtime due to maintenance, servicing, or unforeseen technical issues, thereby ensuring continuous accessibility for users.

- The device's measurement accuracy meets or exceeds industry standards, with a maximum allowable margin of error of ±5% compared to laboratory-grade equipment, ensuring confidence in its results among users and regulatory authorities.

2.1.3 User Case Diagram

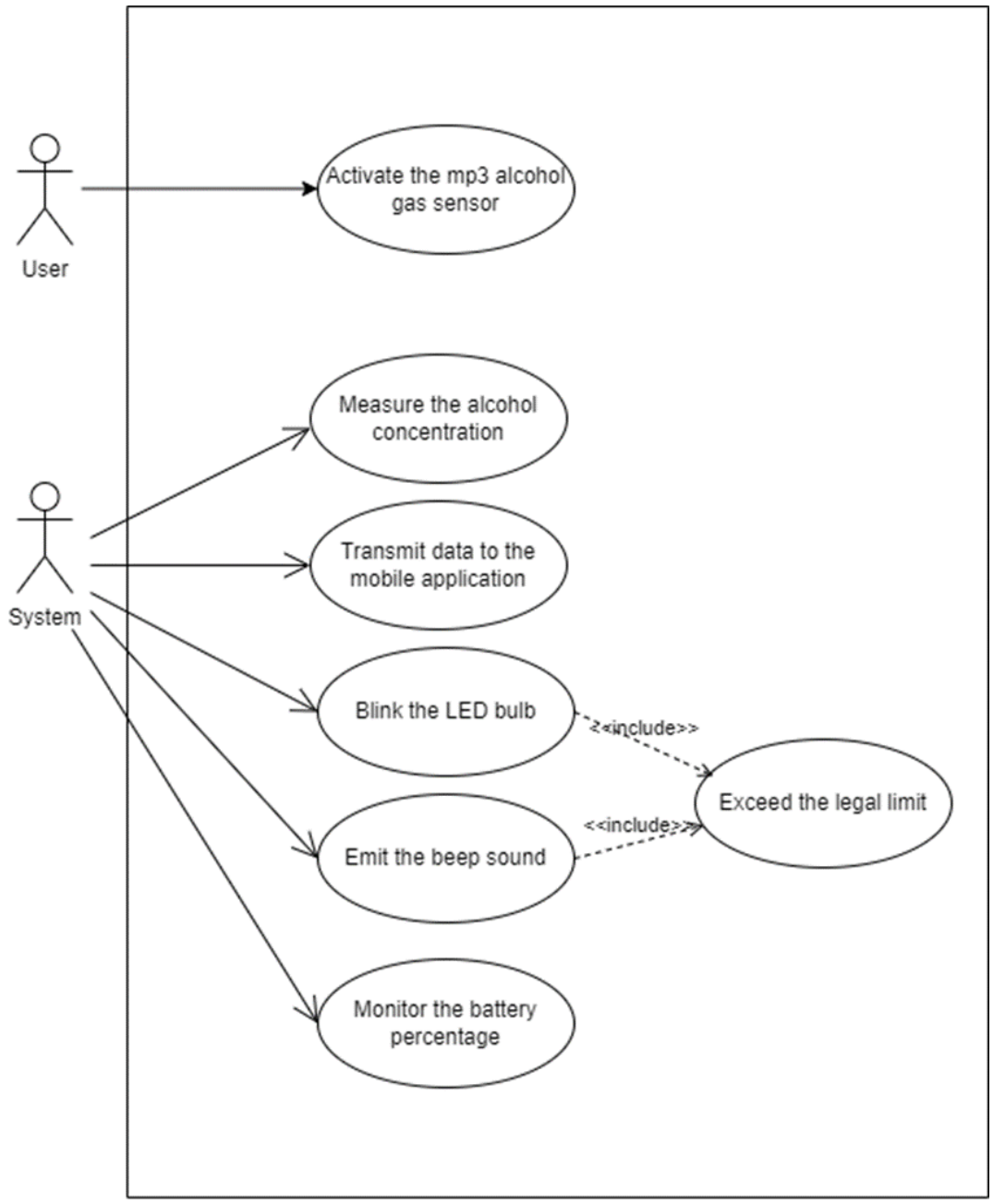

Figure 1: User Case Diagram

2.2 Design

2.2.1 High level architecture diagram

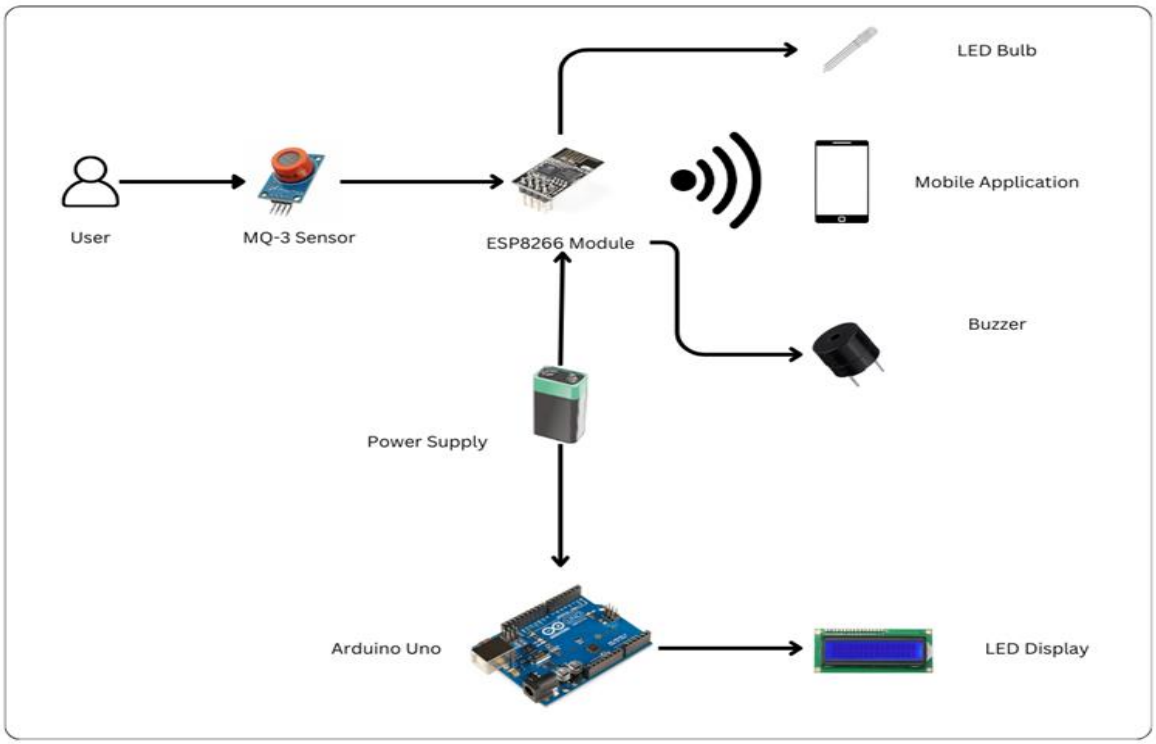

Figure 2: High Level Architecture Diagram

The MP3 alcohol gas sensor accurately detects the alcohol level in user-provided breath samples. The ESP8266 module servers as the central microcontroller, overseeing sensor data purchasing, processing, and communication tasks. It sends the processed data to the blynk mobile application through Wi-Fi. The user interface of the mobile application displays the alcohol level with a message indicating whether it is safe to drive or not. When the alcohol is detected in the user's breath the LED bulb turns red, if not it turns into green color. The buzzer emits a beep sound when the alcohol is detected. The device's battery percentage is calculated using a voltage divider connected to the Arduino Uno and it is displayed through the LED display.

2.2.2 Activity Diagram

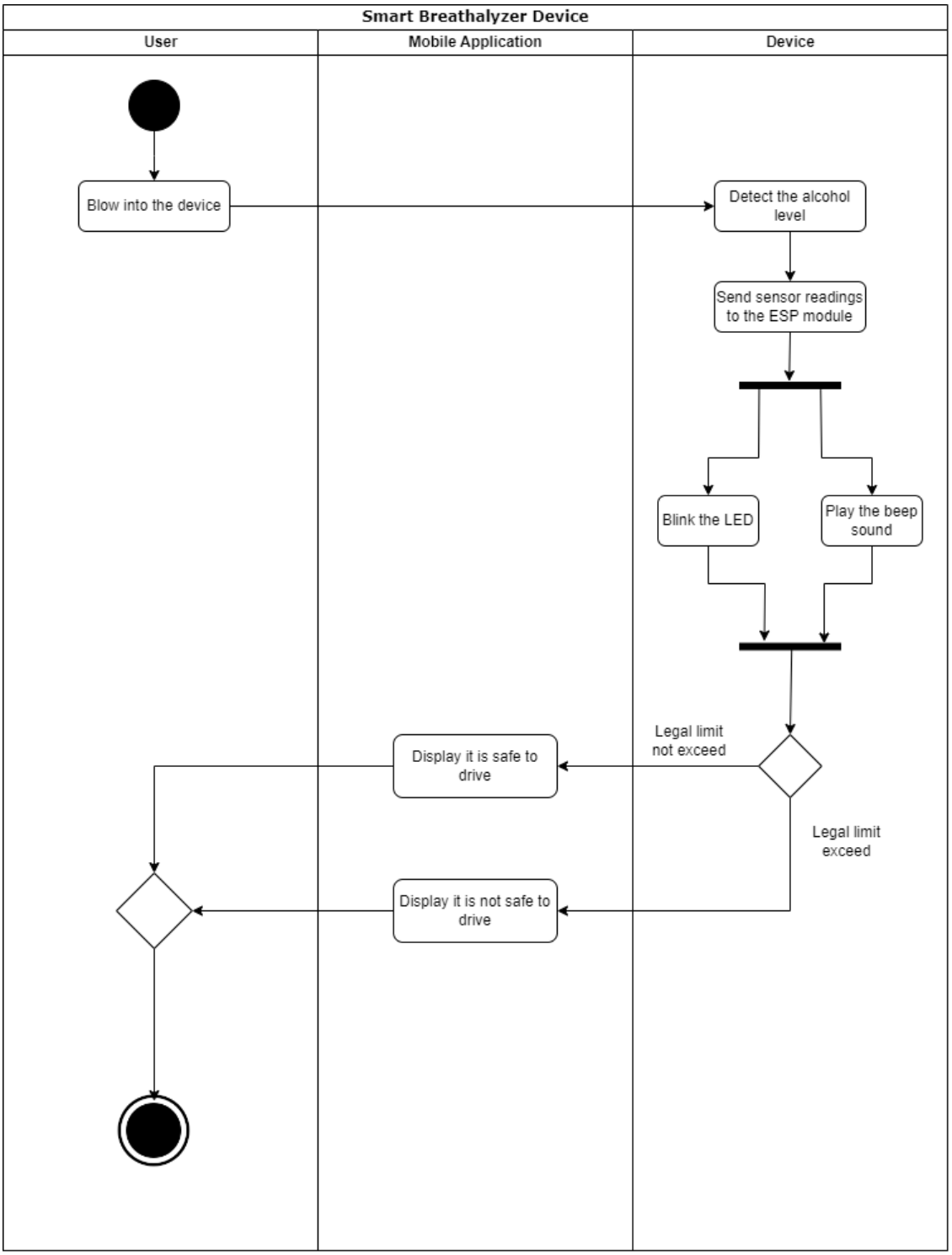

Figure 3: Activity Diagram

Activity Diagram for the LED display

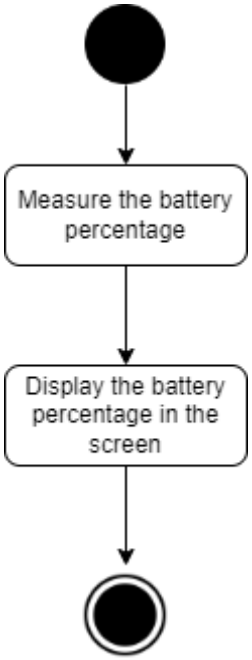

Figure 4: Activity Diagram for the LED Display

2.2.3 User Interface

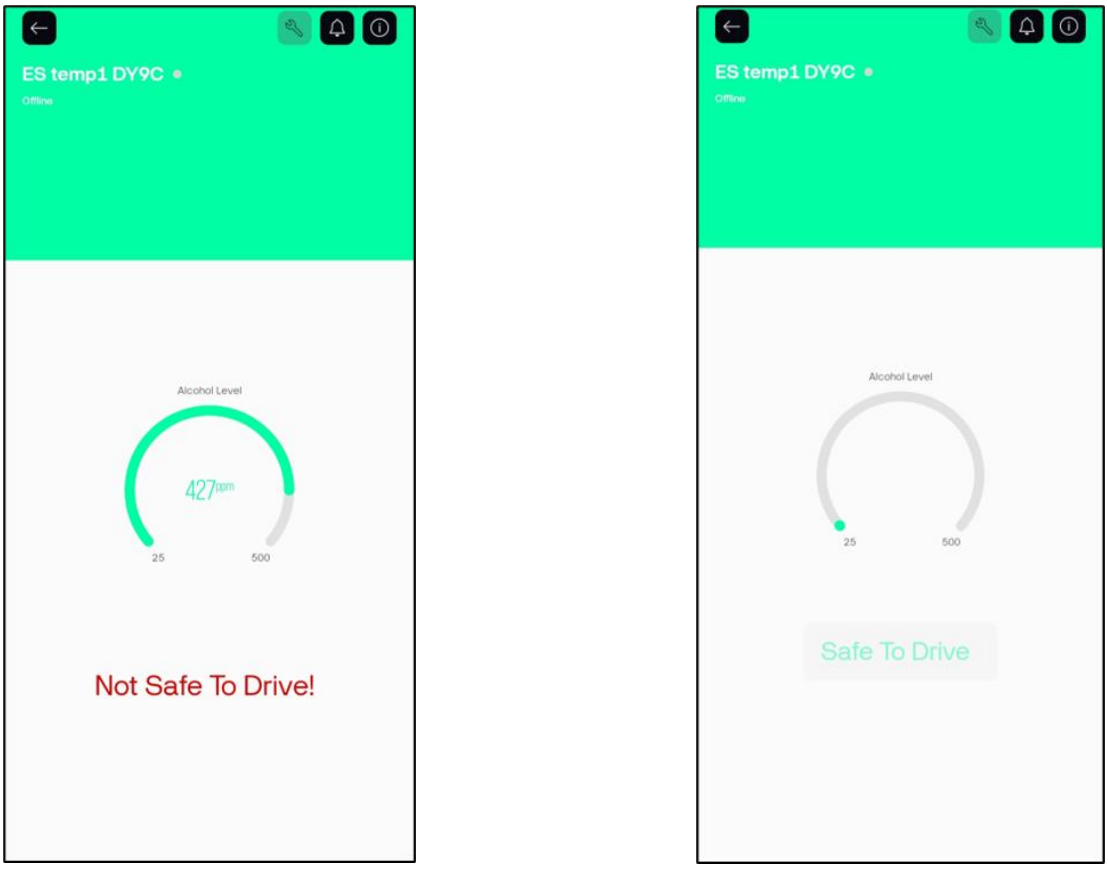

Figure 5: User Interface of the mobile application

The mobile application's user interface incorporates a gauge component to visually represent the alcohol level, accompanied by a message display component. These two components are connected to an ESP8266 module through separate data streams facilitated by virtual pins. The gauge is designed to display values ranging from 25 parts per million (ppm) to 500 ppm. The lower bound of 25 ppm corresponds to the minimum detectable value by the MQ3 sensor, while the upper bound of 500 ppm represents the maximum alcohol level that can be present in a person's breath. The threshold value to determine whether an individual is considered intoxicated or not is 400 ppm. If the detected alcohol level falls below this threshold, the message display indicates that it is safe to drive. Conversely, if the alcohol level exceeds the 400 ppm threshold, the message advises it is not safe to drive.

2.3 Implementation

The design of the Smart Breathalyzer Device can be distinguished by 2 different main systems. The Wi-Fi connected sensor & the voltage divider + battery percentage display.

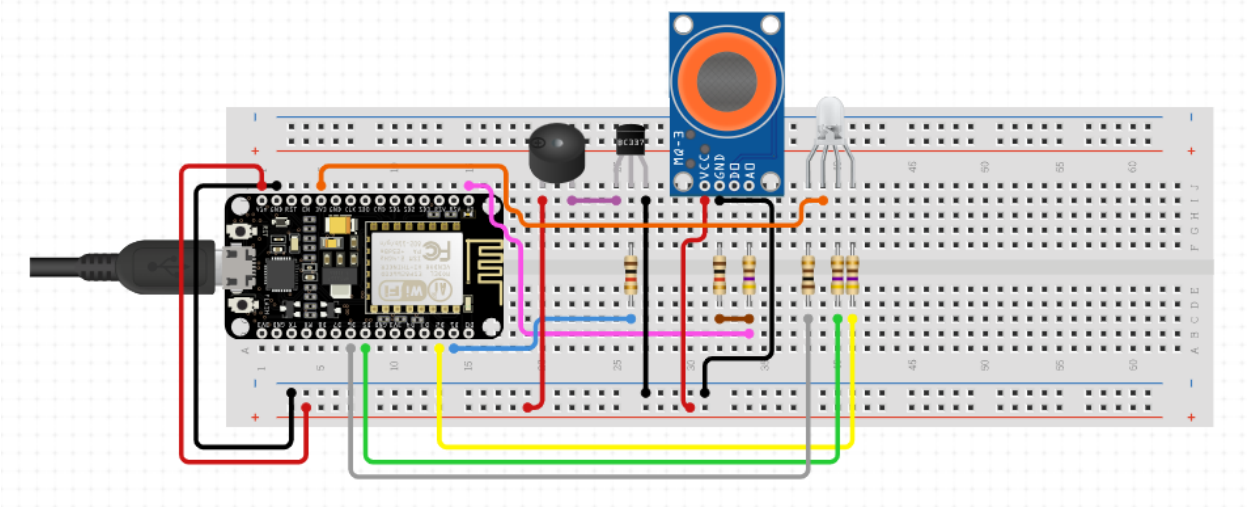

Figure 6: Circuit diagram of the ESP Module

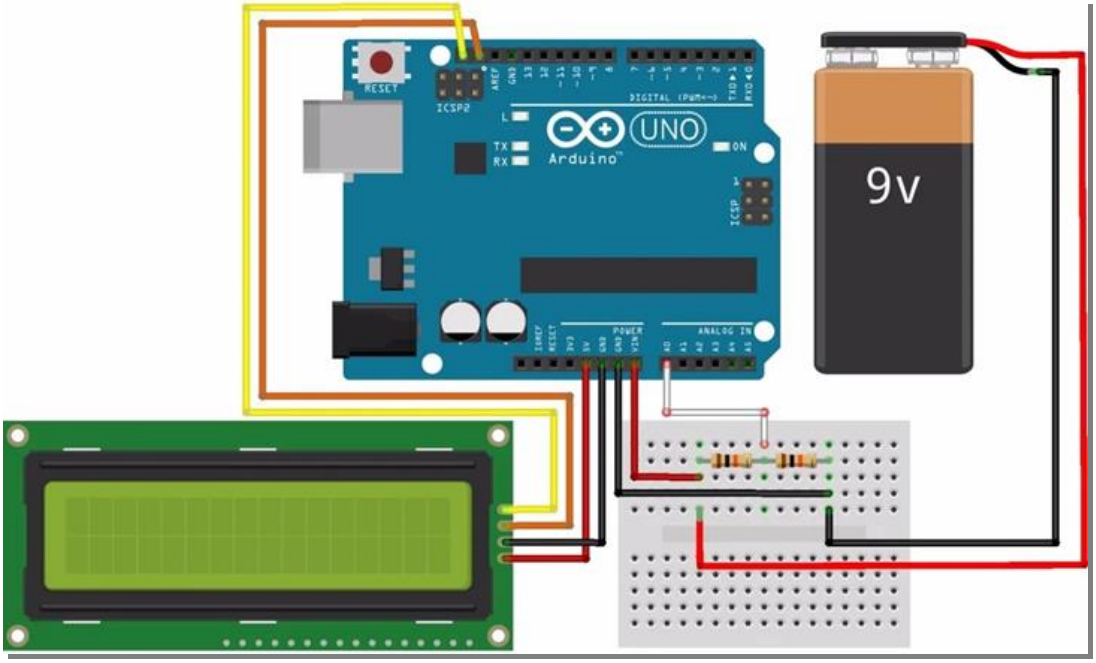

Figure 7: Circuit diagram of the LCD Display

Wi-Fi connected sensor

Overview

The system uses an ESP8266 module, to which all the other components are connected to. The connected MQ3 sensor takes the readings and sends the analog data to the ESP8266 module. The ESP8266 module will be connected to the same Wi-Fi network as the device which the app is used on, at which point, the data from the sensor will be shared with the app, The LED and buzzer will also indicate the state of the user i.e. whether they are drunk or sober. They are all powered by the battery.

Components

- ESP8266 module
- MQ3 gas sensor (detects alcohol)
- Piezo buzzer (emits buzzer sound)
- 4-pin multicolor LED
- Resistors
- Battery

Codes

Source code - Initialization

```
#include "BlynkEdgent.h"
#define Sober 399.00 // max value that we consider sober
#define Drunk 400.00 // min value that we consider drunk
#define MQ3pin 0
#define BuzzerPin D3 // Define the buzzer pin
#define RedPin 5    // Define the red LED pin
#define GreenPin 4  // Define the green LED pin
#define BluePin 2   // Define the blue LED pin
```

```
char auth[] = "Wkwve058gD2ZHFtQuUzYCKVU7TLX9iFi";
char ssid[] = "PEPIM";
char pass[] = "12347890";
```

Explanation

- Initializes “Sober” and “Drunk” threshold values
- Initializes the pins for the sensor, buzzer & LED
- Provides the ESP8266 module with the data to connect to the Wi-Fi network and the Blynk app

---

Source code - Boot Process

```
void setup()
{
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  Serial.println("MQ3 warming up!");
  delay(10000);

  pinMode(RedPin, OUTPUT);
  pinMode(GreenPin, OUTPUT);
  pinMode(BluePin, OUTPUT);
  pinMode(BuzzerPin, OUTPUT); // Set buzzer pin as output

 flashLED(BluePin, 500);
}
```

Explanation

- Initializes serial communication and the Blynk application
- Waits for 10 seconds to let the MQ3 sensor to heat up
- Initializes LED pins and buzzer pins
- Flashes blue pin to indicate the device is starting up

Source code - Sensor

```
double getAlcoholLevel() {
  double sensorValue;
  sensorValue = analogRead(MQ3pin); // read analog input pin 0

  Serial.print("Sensor Value: ");
  Serial.print(sensorValue);

  // Determine the status
  if (sensorValue < Sober) {
    Serial.println(" | Status: Sober");
  }/* else if (sensorValue >= Sober && sensorValue < Drunk) {
    Serial.println(" | Status: Drinking but within legal limits");
  }*/ else {
    Serial.println(" | Status: DRUNK");

  return sensorValue;
}
```

Explanation

- Reads the analog value from the MQ-3 alcohol sensor connected to pin MQ3pin and stores it in “sensorValue”
- Prints the sensor value to the Serial Monitor
- If the sensor value is below Sober, prints "Status: Sober" to the Serial Monitor.
- If the sensor value is not below Sober, prints "Status: DRUNK" to the Serial Monitor.

Source code - LED & app

```
void loop()
{
  // Assuming alcohol sensor data is available as a float variable named alcoholLevel
  double alcoholLevel = getAlcoholLevel();

  // Send alcohol level data to Blynk app
  Blynk.virtualWrite(V0, alcoholLevel); // Assuming V1 is the virtual pin for alcohol level

  if (alcoholLevel < Sober) {
    Blynk.setProperty(V1, "color", "#44D62C"); // Green color
    Blynk.virtualWrite(V1, messageDisplay(alcoholLevel));
    setLEDState(GreenPin, HIGH); // Turn on green LED
    setLEDState(RedPin, LOW);    // Turn off red LED
  } else {
    Blynk.setProperty(V1, "color", "#D40000"); // Red color
    Blynk.virtualWrite(V1, messageDisplay(alcoholLevel));
    setLEDState(GreenPin, LOW);  // Turn off green LED
    setLEDState(RedPin, HIGH);   // Turn on red LED
  } Blynk.run();
```

```
  delay(6500); // Adjust delay as needed
}
```

Explanation

- Reads the alcohol level from the sensor and stores it in the “alcoholLevel” variable.
- Sends the alcohol level data to the Blynk app
- Sends a message to the Blynk app using virtual pin V1 with a custom message based on “alcoholLevel”
- Turns on the green LED & turns off the red LED if the “alcoholLevel” is less than the Sober threshold.
- Also changes the color shown in the Blynk app to green
- Turns off the green LED & turns on the red LED if the “alcoholLevel” is greater than the Sober threshold
- Also changes the color shown in the Blynk app to red
- Waits for 6.5 seconds before the next iteration of the loop.

Source code - App

```
String messageDisplay(double sensorValue) {
  String message;
  if (sensorValue < Sober) {
    message = "Safe To Drive";
    Serial.print("Safe to drive \n");
  } else {
    message = "Not Safe To Drive! ";
    Serial.print("Not Safe to drive !\n");
  }

  delay(2000); // wait 2s for next reading
```

```
  return message;
}
```

Explanation

- If the sensor value is below Sober, print "Safe To Drive" to the Serial Monitor & app.
- If the sensor value is not below Sober, prints "Not Safe To Drive!" to the Serial Monito & app.
- Wait for 2 seconds before continuing.

Source code - Buzzer

```
void beep(double sensorValue) {
  // Beep the buzzer
  if(sensorValue > Sober){
  digitalWrite(BuzzerPin, HIGH); // Turn buzzer on
  delay(200); // Beep duration
  digitalWrite(BuzzerPin, LOW);
  }

}
```

Explanation

- If the sensor value is below Sober, the buzzer beeps

Voltage Divider + Battery Percentage Display

Overview

The system uses an Arduino Uno, to which the LCD display and Battery are connected to. The power is provided through a voltage divider which detects the voltage drop in the

Components

- Arduino Uno
- I2C 16x2 LCD Display
- Battery
- Resistors

Codes

Source code

```
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.clear();
}

void loop() {
  float variableBattery = readBatteryVoltage();

  int batteryPercentage = calculateBatteryPercentage(variableBattery);

    lcd.print(" Batt=");
    lcd.print(batteryPercentage);
    lcd.print("%");
```

```
  delay(1000); }
```

Explanation

- Initializes the LCD display
- Turns on the LCD backlight
- Clears any existing text or graphics on the LCD screen
- Reads the battery voltage using a custom function “readBatteryVoltage()” and stores it in the variable “variableBattery”
- Calculates the battery percentage based on the voltage using a custom function “calculateBatteryPercentage()” and stores it in the “variable batteryPercentage”
- Prints the battery percentage on the LCD

Source code

```
float readBatteryVoltage() {
  int rawValue = analogRead(batteryPin) ;
  float voltage = rawValue * (9.0/ 1023);
  return voltage;
}
```

Explanation

- Reads the analog value from the pin connected to the battery (referred to as batteryPin) and stores it in the variable rawValue.
- Converts the raw analog value to a voltage. This conversion assumes that the maximum voltage corresponds to the maximum raw analog value for 10-bit ADC(1023).
- Returns the calculated voltage.

Source code

```
int calculateBatteryPercentage(float voltage) {
  float minVoltage = 3.6;
  float maxVoltage = 9.0;

  int batteryPercentage = map(voltage, minVoltage, maxVoltage, 0, 100) ;
  batteryPercentage = constrain(batteryPercentage, 0, 100);

  return batteryPercentage;
}
```

Explanation

- Define the battery voltage range
- Map the battery voltage to a percentage
- Ensure the percentage is within the 0-100 range
- Return the calculated battery percentage

2.4 Testing

Function 1 :- Alcohol Level Detection

Table 7: Test scenario 1

Test Scenario ID	Alcohol Level Detection - 1	**Test Case ID**	Alcohol Detection - 1
Test Case Description	Calibrating the MQ3 sensor	**Test Priority**	High
Pre-Requisite	Power on circuit	**Post-Requisite**	Connection to NodeMCU

Test Execution Steps :

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
01.	Blow into the nozzle	Breath	The serial monitor shows the detected alcohol value	The serial monitor shows the detected alcohol value	Pass
02.	Check if the sensor gives the same value for each temperature value	Breath	The sensor gives same values for each reading	The sensor gives same values for each reading	Pass

Function 2 :- Implementing Wi-Fi Communication

Table 8: Test scenario 2

Test Scenario ID	Implementing Wi-Fi Communication - 1	**Test Case ID**	Wi-Fi Communication - 1
Test Case Description	ESP 8266 module connects to Wi-Fi	**Test Priority**	High
Pre-Requisite	Power up the module and provide it with the login information of the Wi-Fi network through the code.	**Post-Requisite**	Sending a signal to Wi-Fi module.

Test Execution Steps :

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
01.	Upload the code to	Code	The light on the	the light on the module	Pass

	the module		module is blinking (which means it's connected)	is blinking (which means it's connected)	

Test Scenario ID	Implementing Wi-Fi Communication - 1	**Test Case ID**	Wi-Fi Communication - 2
Test Case Description	ESP 8266 module connects to Blynk app	**Test Priority**	High
Pre-Requisite	Power up the module and provide it with the code. Connect the device and the module to the same Wi-Fi.	**Post-Requisite**	N/A

Test Execution Steps :

S.No	Action	Inputs	Expected Output	Actual Output	Test Result
01.	Upload the code to the module	Code	The app switches to online mode from offline mode	The app switches to online mode from offline mode	Pass

Test Scenario ID	Implementing Wi-Fi Communication - 1	**Test Case ID**	Wi-Fi Communication - 3
Test Case Description	Connection Range	**Test Priority**	High
Pre-Requisite	Power up the module and provide it with the login information of the Wi-Fi network through the code.	**Post-Requisite**	Working app

Test Execution Steps :

S.No	Action	Inputs	Expected Output	Actual Output	Test Result
01.	Blow on the sensor nozzle	Breath	The app works normally	The app works normally	Pass

Function 3 :- Displaying the data in the mobile app interface

Table 9: Test scenario 3

Test Scenario ID	App interface implementation - 1	**Test Case ID**	App interface - 1
Test Case Description	Testing the app response to sensor data	**Test Priority**	High
Pre-Requisite	Device with the app and Wi-Fi module must be connected to the same Wi-Fi network	**Post-Requisite**	N/A

Test Execution Steps :

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
01.	Blow into the nozzle	Breath	App responds accordingly by displaying whether the sensor value is over or under the sensor data	App responds accordingly by displaying whether the sensor value is over or under the sensor data	Pass

Function 4 :- **LED Alert Activation**

Table 10: Test scenario 4

Test Scenario ID	LED Alert Activation (Red)	**Test Case ID**	LED Activation -1
Test Case Description	LED bulb changing color according to sensor data	**Test Priority**	High
Pre-Requisite	Sensor reading goes through Bulb is already Green	**Post-Requisite**	Bulb changing color

Test Execution Steps : Compare the sensor alcohol level with the threshold limit

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
01.	Blow on the nozzle connected to the sensor with no alcohol in the	Sensor reading from the mq3 alcohol gas sensor	No changes to the LED bulb	No changes to the LED bulb	Pass

	system				
02.	Blow on the nozzle connected to the sensor with alcohol in the system	Sensor reading from the mq3 alcohol gas sensor	LED bulb turns red	LED bulb turns red	Pass

Test Scenario ID	LED Alert Activation (Green)	**Test Case ID**	LED Activation -1
Test Case Description	LED bulb changing color according to sensor data	**Test Priority**	High
Pre-Requisite	Sensor reading goes through Bulb is already Red	**Post-Requisite**	Bulb changing color

Test Execution Steps : Compare the sensor alcohol level with the threshold limit

S.No	Action	Inputs	Expected Output	Actual Output	Test Result
01.	Blow on the nozzle connected to the sensor with alcohol in the system	Sensor reading from the mq3 alcohol gas sensor	No changes to the LED bulb	No changes to the LED bulb	Pass
02.	Blow on the nozzle connected to the sensor with no alcohol in the system	Sensor reading from the mq3 alcohol gas sensor	LED bulb turns green	LED bulb turns red	Pass

Function 5 :- **Speaker Alert Activation**

Table 11: Test scenario 5

Test Scenario ID	Speaker Alert Activation	**Test Case ID**	Speaker Activation -1
Test Case Description	Piezo buzzer beeping according to sensor data	**Test Priority**	High
Pre-Requisite	Sensor reading	**Post-Requisite**	Buzzer beeping

Test Execution Steps : Compare the sensor alcohol level with the threshold limit

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
01.	Blow on the nozzle connected to the sensor with no alcohol in the system	Sensor reading from the mq3 alcohol gas sensor	Buzzer does not beep	Buzzer does not beep	Pass
02.	Blow on the nozzle connected to the sensor with alcohol in the system	Sensor reading from the mq3 alcohol gas sensor	Buzzer beeps	Buzzer beeps	Pass

Function 6 :- Displaying the Battery Percentage

Table 12: Test scenario 6

Test Scenario ID	Displaying the Battery Percentage - 1	**Test Case ID**	Battery Percentage - 1
Test Case Description	Testing the voltage divider with the Arduino	**Test Priority**	High
Pre-Requisite	Turn on the power	**Post-Requisite**	N/A

Test Execution Steps :

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
01.	Turn on the power	Power	The serial monitor shows voltage of the power supply	The serial monitor shows voltage of the power supply	Pass

Test Scenario ID	Displaying the Battery Percentage - 1	**Test Case ID**	Battery Percentage - 1
Test Case Description	Testing the LCD panel	**Test Priority**	High
Pre-Requisite	The code must be on the Arduino	**Post-Requisite**	N/A

Test Execution Steps :

S.No	Action	Inputs	Expected Output	Actual Output	Test Result
01.	Connect to power	Arduino code	The voltage is read	The voltage is read	Pass

3. Conclusion

3.1 Assessment of the project results

The Smart Breathalyzer Device project met its objectives by developing an effective tool to monitor and prevent drunk driving while enhancing road safety. The device ensured accurate alcohol detection using advance sensors. However, the original plan was to get the sensor readings only once when the user blows into the device and display it in the mobile application. But the sensor kept continuously taking the readings and this problem remained unresolved. Additionally, there were lag issues when displaying the value and the message in the mobile application, which affects the user experience. Despite these challenges, the device successfully met its primary objectives, showing a significant potential in preventing drunk driving.

3.2 Lessons Learned

Several key lessons were learned during the development of the Smart Breathalyzer Device:

1. Technical Challenges: The continuous sensor readings and unresolved lag issues highlighted the need for deeper technical troubleshooting and validation processes.
2. Importance of Testing and User Feedback: Iterative testing and feedback from users were crucial in refining the device and mobile application, ensuring a user-friendly and effective product.
3. Time Management: Meeting deadlines required good time management and task prioritization.
4. Resource Allocation: Efficient use of the allocated budget underscored the need for careful resource planning and management .

4. References

[1] *Hackaday.io*. [Online]. Available: https://hackaday.io/project/18705-iot-breathalyzer. [Accessed: 18-Jun-2024].

[2] *Amazon.com*. [Online]. Available: https://www.amazon.com/Best-Sellers-Personal-Breathalyzers/zgbs/hpc/15992781. [Accessed: 18-Jun-2024].

[3] "What is a breath alcohol test?," *WebMD*. [Online]. Available: https://www.webmd.com/mental-health/addiction/breath-alcohol-test. [Accessed: 18-Jun-2024].

[4] Last Minute Engineers, "How MQ3 Alcohol Sensor works? & interface it with Arduino," *Last Minute Engineers*, 18-Dec-2020. [Online]. Available: https://lastminuteengineers.com/mq3-alcohol-sensor-arduino-tutorial/. [Accessed: 18-Jun-2024].

[5] "Battery percentage," *Arduino Forum*, 22-Aug-2023. [Online]. Available: https://forum.arduino.cc/t/battery-percentage/1160621/8. [Accessed: 18-Jun-2024].

[6] upir, "DIY Battery Indicator (Arduino Project)," 15-Jul-2022. [Online]. Available: https://www.youtube.com/watch?v=Mq0WBPKGRew. [Accessed: 18-Jun-2024].

[7] "16x2_display_indicator__upir.Ino - wokwi ESP32, STM32, Arduino simulator," *Wokwi.com*. [Online]. Available: https://wokwi.com/projects/336618647787143762. [Accessed: 18-Jun-2024].

[8] *Robu.in*. [Online]. Available: https://robu.in/arduino-pin-configuration/. [Accessed: 18-Jun-2024].

[9] » M. A. C., "Arduino Battery Voltage Indicator," *Instructables*, 08-May-2015. [Online]. Available: https://www.instructables.com/Arduino-Battery-Voltage-Indicator/. [Accessed: 18-Jun-2024].

[10] *Making your first project on New Blynk 2.0 ⚡*. 2021.

[11] *IOT blynk create gauge widget*. 2023.

[12] *How to fix errors in new Blynk2.0 | compilation error |connection error | board error| library error*. 2022.

Appendix : Selected Code Listings

Code for the Blynk app and the alert system

```
#define BLYNK_TEMPLATE_ID "TMPL6egdYquoa"
#define BLYNK_TEMPLATE_NAME "ES temp1"
#define BLYNK_FIRMWARE_VERSION "0.1.0"

#define BLYNK_PRINT Serial

#define APP_DEBUG

#include "BlynkEdgent.h"

#define Sober 399.00 // max value that we consider sober
#define Drunk 400.00 // min value that we consider drunk
#define MQ3pin 0
#define BuzzerPin D3 // Define the buzzer pin
#define RedPin 5    // Define the red LED pin
#define GreenPin 4  // Define the green LED pin
#define BluePin 2   // Define the blue LED pin

char auth[] = "Wkwve058gD2ZHFtQuUzYCKVU7TLX9iFi"; //Blynk Auth Token
char ssid[] = "PEPIM";
char pass[] = "12347890";

void setup()
{
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  Serial.println("MQ3 warming up!");
  delay(10000); // allow the MQ3 to warm up

  // Set LED pins as outputs
  pinMode(RedPin, OUTPUT);
  pinMode(GreenPin, OUTPUT);
  pinMode(BluePin, OUTPUT);
  pinMode(BuzzerPin, OUTPUT); // Set buzzer pin as output
```

```
  // Set initial LED state (flashing blue)
  flashLED(BluePin, 500); // Flash blue LED for 500ms
}

void loop()
{

  double alcoholLevel = getAlcoholLevel();

  // Send alcohol level data to Blynk app
  Blynk.virtualWrite(V0, alcoholLevel);

  if (alcoholLevel < Sober) {
    Blynk.setProperty(V1, "color", "#44D62C"); // Green color
    Blynk.virtualWrite(V1, messageDisplay(alcoholLevel));
    setLEDState(GreenPin, HIGH); // Turn on green LED
    setLEDState(RedPin, LOW);    // Turn off red LED
  }

else {
    Blynk.setProperty(V1, "color", "#D40000"); // Red color
    Blynk.virtualWrite(V1, messageDisplay(alcoholLevel));
    setLEDState(GreenPin, LOW);  // Turn off green LED
    setLEDState(RedPin, HIGH);   // Turn on red LED
  }

  Blynk.run();
  delay(6500);
}

double getAlcoholLevel() {
  double sensorValue;
  sensorValue = analogRead(MQ3pin); // read analog input pin 0

  Serial.print("Sensor Value: ");
  Serial.print(sensorValue);

  // Determine the status
  if (sensorValue < Sober) {
    Serial.println(" | Status: Sober");
```

```
  }/* else if (sensorValue >= Sober && sensorValue < Drunk) {
    Serial.println(" | Status: Drinking but within legal limits");
  }*/ else {
    Serial.println(" | Status: DRUNK");

  }

  beep(sensorValue); // Beep the buzzer
  delay(1000); // wait 2s for next reading

  return sensorValue;
}

String messageDisplay(double sensorValue) {
  String message;
  if (sensorValue < Sober) {
    message = "Safe To Drive";
    Serial.print("Safe to drive \n");
  } else {
    message = "Not Safe To Drive! ";
    Serial.print("Not Safe to drive !\n");
  }

  delay(2000); // wait 2s for next reading
  return message;
}

void beep(double sensorValue) {
  // Beep the buzzer
  if(sensorValue > Sober){
  digitalWrite(BuzzerPin, HIGH);
  delay(200); // Beep duration
  setLEDState(GreenPin, LOW);  // Turn off green LED
  setLEDState(RedPin, HIGH); // Turn buzzer on
  digitalWrite(BuzzerPin, LOW);
  delay(200); // Beep duration

  }// Turn buzzer off
}
```

```
void setLEDState(int pin, int state) {
  digitalWrite(pin, state);
}

void flashLED(int pin, int duration) {
  setLEDState(pin, HIGH);
  delay(duration);
  setLEDState(pin, LOW);
}
```

Code for the LCD display

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C address 0x27, 16 column and 2 rows
const int batteryPin = A2;

void setup() {
  lcd.init(); // initialize the lcd
  lcd.backlight();
  lcd.clear();
}

void loop() {
  float variableBattery = readBatteryVoltage(); // function to read voltage

  int batteryPercentage = calculateBatteryPercentage(variableBattery);

    lcd.print("Battery Per =");
    lcd.print(batteryPercentage);
    lcd.print("%");

    delay(50000); }


float readBatteryVoltage() {
  int rawValue = analogRead(batteryPin) ;
  // Convert rawValue to voltage
  float voltage = rawValue * (5.0/ 1023);
```

```
  return voltage;
}

int calculateBatteryPercentage(float voltage) {
  // battery's voltage range and corresponding capacity
  float minVoltage = 0.0; // Minimum voltage
  float maxVoltage = 9.5; // Maximum voltage

  // Calculate battery percentage based on voltage range
  int batteryPercentage = map(voltage, minVoltage, maxVoltage, 0, 100) ;
  batteryPercentage = constrain(batteryPercentage, 0, 100); // Limit to 0-100

  return batteryPercentage;
}
```