**Naan Mudhalvan**

**IOT Project**

**Public Transportation Optimization**

**Phase-4**

**DEVELOPMENT PHASE 2**

This code is designed for a system based on ESP32, which utilizes ultrasonic sensors and an LCD display to detect the presence of objects like passengers within a specific range. To enable remote monitoring of passenger counts, the code incorporates the Blynk platform. Here is a detailed, step-by-step explanation of how the code operates:

1. **Blynk Setup**:
   * The code begins by defining essential Blynk constants:
     + **BLYNK\_TEMPLATE\_ID** and **BLYNK\_TEMPLATE\_NAME**: Template ID and name used in Blynk.
     + **BLYNK\_AUTH\_TOKEN**: The Blynk authentication token that enables communication with the Blynk server.
2. **Library Inclusion**:
   * The code includes required libraries:
     + **WiFi.h**: For establishing Wi-Fi connectivity.
     + **WiFiClient.h**: For managing the Wi-Fi client.
     + **BlynkSimpleEsp32.h**: The Blynk library designed for ESP32.
     + **LiquidCrystal\_I2C.h**: A library used to control the I2C-connected LCD.
3. **Variable Declarations**:
   * Several variables are declared, including:
     + **distanceThreshold**: A specified distance (in centimeters) used to detect nearby objects (passengers).
     + Timing and distance measurement variables.
     + **entered** and **left**: Counters for passengers who have entered and left.
     + **boarding**: A boolean variable to track the boarding process's status.
     + Definitions for pins connected to ultrasonic sensors (trigger and echo).
     + **BlynkTimer**: An object for scheduling tasks.
     + An **LiquidCrystal\_I2C** object for controlling the I2C LCD.
     + Authentication token (**auth**), Wi-Fi network credentials (**ssid** and **pass**).
4. **Distance Measurement Functions**:

Two functions, **get\_distance1** and **get\_distance2**, are defined to measure distances using ultrasonic sensors. These functions trigger signals, measure echo signal durations, and calculate distances considering the speed of sound.

1. **myTimer Function**:
   * The **myTimer** function is scheduled to run at intervals by Blynk.
   * It invokes the distance measurement functions (**get\_distance1** and **get\_distance2**) to obtain distances from two ultrasonic sensors.
   * It checks if no boarding process is ongoing and if the distances from both sensors are less than the **distanceThreshold**. If both conditions are met, it sets **boarding** to true (indicating boarding is in progress).
   * If **boarding** is true and either of the distances exceeds the **distanceThreshold**, it implies the boarding process is complete. The code sets **boarding** to false and increments both **entered** and **left** counts.
   * The passenger count is updated on the LCD display, and this count is sent to Blynk using virtual pins **V0** and **V1**.
2. **Setup Function**:
   * The **setup** function is executed once at the program's start.
   * It initializes serial communication for debugging.
   * Sets up pins for the LED and the ultrasonic sensors (trigger as output and echo as input).
   * Initializes the I2C-connected LCD and turns on the backlight.
   * Initiates the Blynk connection using the provided authentication token, Wi-Fi credentials, and Blynk server address.
   * Schedules the **myTimer** function to run every 1000 milliseconds (1 second) using the Blynk timer.
3. **Loop Function**:
   * The **loop** function runs continuously, maintaining the Blynk connection and executing scheduled tasks through the Blynk timer.

In summary, this code creates an ESP32-based system that uses ultrasonic sensors to monitor the presence of objects within a certain proximity and displays the passenger count on an LCD. It also sends this count to the Blynk platform for remote monitoring. The code frequently checks the distances, updates the count, and informs Blynk about the passenger count changes.

#define BLYNK\_TEMPLATE\_ID "Your\_Template\_ID"

#define BLYNK\_TEMPLATE\_NAME "Your\_Template\_Name"

#define BLYNK\_AUTH\_TOKEN "XEHxNF\_Ur1Nt2p7wB5B20dNI1ZUwj34P"

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

#include <LiquidCrystal\_I2C.h>

int distanceThreshold = 20;  // Adjust this value as needed

int duration1 = 0;

int distance1 = 0;

int duration2 = 0;

int distance2 = 0;

int dis1 = 0;

int dis2 = 0;

int entered = 0;

int left = 0;

bool boarding = false; // Track if boarding is in progress

#define LED 2

#define PIN\_TRIG1 15

#define PIN\_ECHO1 14

#define PIN\_TRIG2 13

#define PIN\_ECHO2 12

BlynkTimer timer;

LiquidCrystal\_I2C lcd(0x27, 16, 2);  // Adjust the I2C address if needed

char auth[] = BLYNK\_AUTH\_TOKEN;

char ssid[] = "Wokwi-GUEST";   // your network SSID (name)

char pass[] = "";

#define BLYNK\_PRINT **Serial**

long get\_distance1() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG1, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG1, LOW);

  // Read the result:

  duration1 = pulseIn(PIN\_ECHO1, HIGH);

  distance1 = duration1 / 58;

  return distance1;

}

long get\_distance2() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG2, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG2, LOW);

  // Read the result:

  duration2 = pulseIn(PIN\_ECHO2, HIGH);

  distance2 = duration2 / 58;

  return distance2;

}

void myTimer() {

  dis1 = get\_distance1();

  dis2 = get\_distance2();

  if (!boarding && dis1 < distanceThreshold && dis2 < distanceThreshold) {

    boarding = true; // Boarding in progress

  } else if (boarding and (dis1 > distanceThreshold || dis2 > distanceThreshold)) {

    boarding = false; // Boarding is complete

    entered++;

    left++;

  }

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Passengers: ");

  lcd.print(entered);

  Blynk.virtualWrite(V0, entered);

  Blynk.virtualWrite(V1, left);

}

void setup() {

**Serial**.begin(115200);

  pinMode(LED, OUTPUT);

  pinMode(PIN\_TRIG1, OUTPUT);

  pinMode(PIN\_ECHO1, INPUT);

  pinMode(PIN\_TRIG2, OUTPUT);

  pinMode(PIN\_ECHO2, INPUT);

  // Initialize the LCD in the setup function

  lcd.init();

  lcd.backlight();

  Blynk.begin(auth, ssid, pass, "blynk.cloud", 8080);

  timer.setInterval(1000L, myTimer);

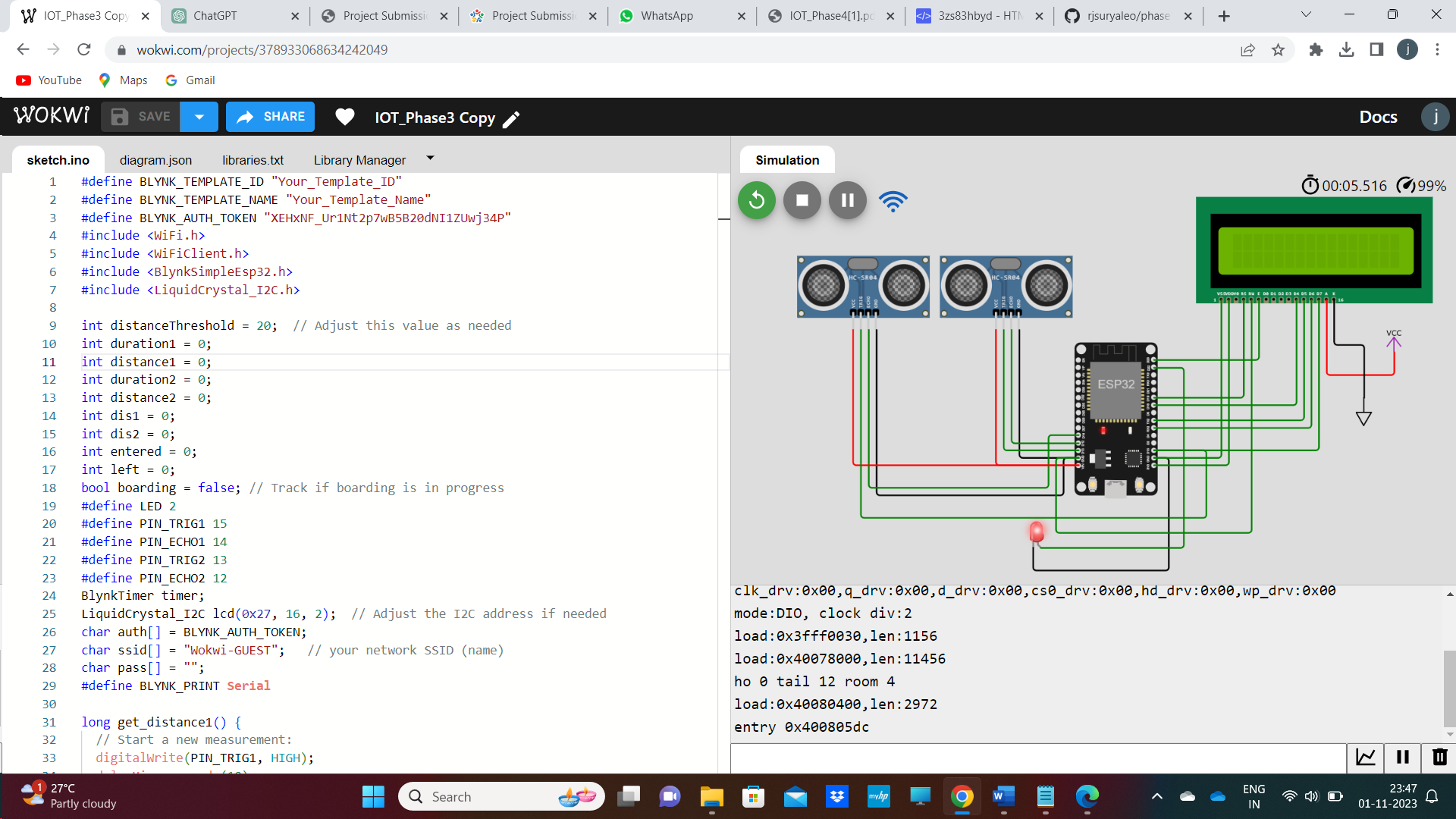
}

void loop() {

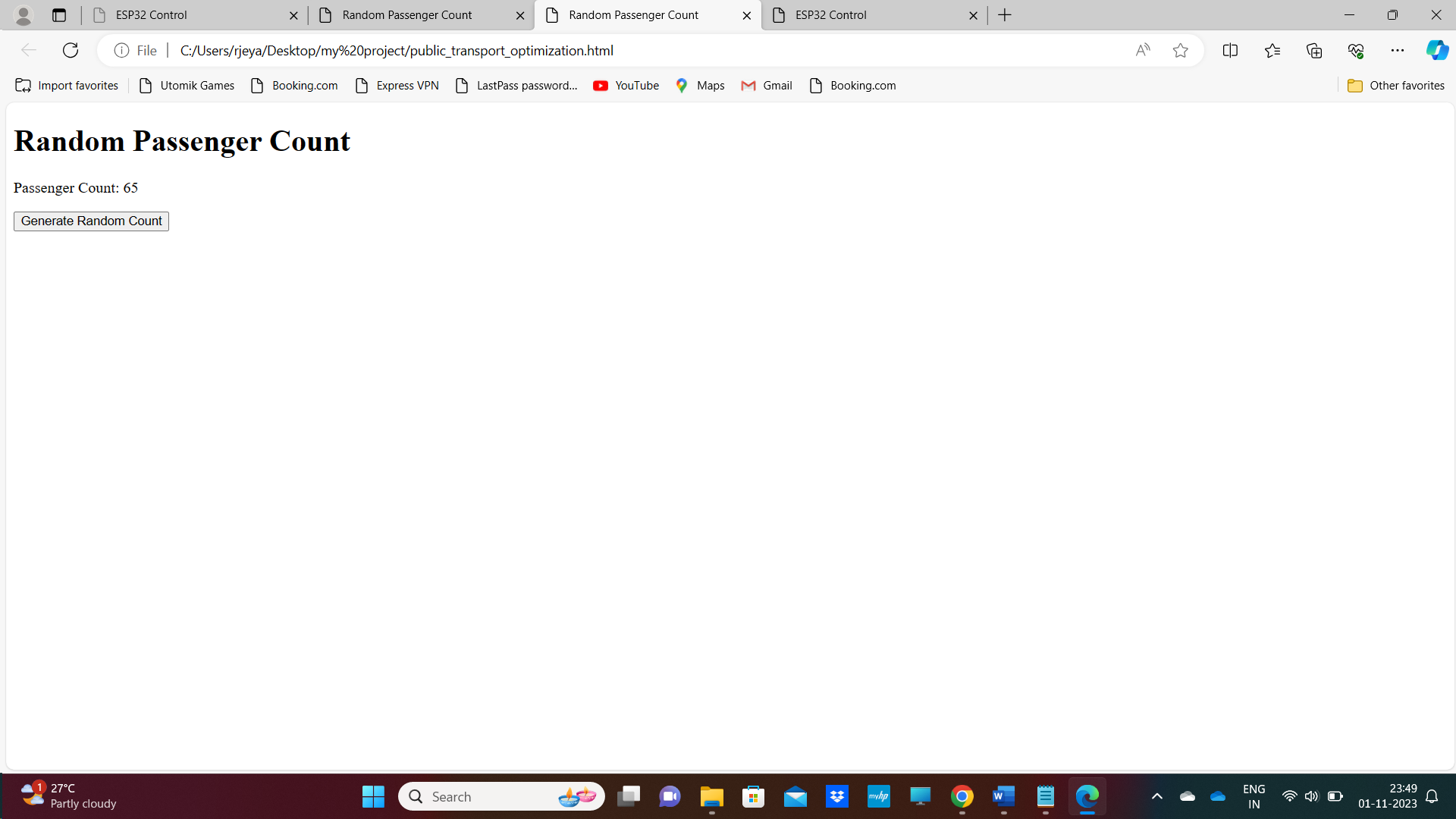
  Blynk.run();

  timer.run();

}



HTML WEBPAGE



<!DOCTYPE html>

<html>

<head>

    <title>Random Passenger Count</title>

</head>

<body>

    <h1>Random Passenger Count</h1>

    <p>Passenger Count: <span id="passengerCount">0</span></p>

    <button onclick="generateRandomCount()">Generate Random Count</button>

    <script>

        function generateRandomCount() {

            // Generate a random passenger count between 0 and 100

            var randomCount = Math.floor(Math.random() \* 101);

            // Display the random count on the web page

            document.getElementById("passengerCount").textContent = randomCount;

        }

    </script>

</body>

</html>

This HTML code creates a simple web page for displaying a random passenger count and allows the user to generate a new random count with the click of a button. Here's an explanation of the code:

1. **HTML Structure**:
   * The document begins with a declaration, **<!DOCTYPE html>**, specifying that it's using HTML5.
   * The content of the web page is enclosed within the **<html>** element.
   * The **<head>** section typically contains document metadata but, in this case, includes the web page title.
2. **Page Title**:
   * The **<title>** element is used to define the title of the web page, which appears in the browser's title bar or tab.
3. **Body Content**:
   * The **<body>** element encompasses the visible content of the web page.
4. **Heading and Passenger Count Display**:
   * A top-level heading **<h1>** displays the text "Random Passenger Count."
   * A paragraph **<p>** contains the text "Passenger Count:".
   * A **<span>** element, uniquely identified by the **id** attribute "passengerCount," is used within the paragraph to display the passenger count.
5. **Generate Random Count Button**:
   * A button element **<button>** allows user interaction on the page.
   * The **onclick** attribute specifies that a JavaScript function, **generateRandomCount()**, is triggered when the button is clicked.
6. **JavaScript Function**:
   * JavaScript code is included within the **<script>** element.
   * The JavaScript function, **generateRandomCount()**, is defined. When called, it performs the following tasks:
     + It generates a random passenger count within the range of 0 to 100 using the **Math.random()** function.
     + The **Math.floor()** function is used to round down the random number to the nearest integer, ensuring it falls within the specified range.
     + It updates the text content of the **<span>** element with the ID "passengerCount" to display the newly generated random count.

In summary, this code creates a web page with a title, a heading, a display area for the passenger count, and a button. JavaScript function generates a random passenger count and updates the display when the button is clicked, a. This webpage demonstrates how HTML and JavaScript collaborate to deliver engaging content to users.