BlindStick Project Documentation: From Wiring to Software Implementation to Final Implementation

ESP32 CONTROLLER THAT SENDS LONGI & LATITUDE TO WEB + DATABASE

Code:

```
#include <WiFi.h>              // Include the Wi-Fi library for ESP32

#include <TinyGPS++.h>         // Include the TinyGPS++ library

#include <HTTPClient.h>        // Include HTTPClient library for making
HTTP requests



TinyGPSPlus gps;               // Create a GPS object

HardwareSerial gpsSerial(1);   // Use UART1 for GPS communication



const int buttonPin = 4;       // GPIO pin for the button

int buttonState = 0;           // Variable to store button state



// Wi-Fi credentials

const char* ssid = "Alba Extender 24";        // Replace with your Wi-Fi
SSID

const char* password = "ExtenderNova24"; // Replace with your Wi-Fi
password



// API endpoint URL

const char* serverURL =
"https://elemsys-api.vercel.app/save-coordinates"; // URL endpoint



void setup() {

  Serial.begin(115200);        // Start the Serial Monitor
```

```
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);  // Start the GPS Serial (TX = 17, RX = 16)

  pinMode(buttonPin, INPUT_PULLUP);  // Configure the button pin as input with pull-up resistor

  // Wi-Fi setup
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);  // Start the Wi-Fi connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWi-Fi connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());  // Print the ESP32's IP address
}

void loop() {
  while (gpsSerial.available() > 0) {
    char c = gpsSerial.read();
    gps.encode(c);  // Feed the GPS data into the TinyGPS++ library
  }

  // Check if the button is pressed
  buttonState = digitalRead(buttonPin);
```

```
  if (buttonState == LOW) {  // Button is pressed
    if (gps.location.isValid()) {
      float latitude = gps.location.lat();
      float longitude = gps.location.lng();

      // Print GPS coordinates to Serial Monitor
      Serial.print("Latitude: ");
      Serial.println(latitude, 6);
      Serial.print("Longitude: ");
      Serial.println(longitude, 6);

      // Send GPS coordinates to the server
      sendDataToServer(latitude, longitude);
    } else {
      Serial.println("Waiting for GPS signal...");
    }
    delay(1000);  // Debounce delay
  }
}

void sendDataToServer(float latitude, float longitude) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Prepare the HTTP POST request
```

```
    http.begin(serverURL);  // Specify server URL

    http.addHeader("Content-Type", "application/json"); // Set content
type to JSON



    // Prepare the JSON payload with latitude and longitude

    String jsonPayload = "{\"latitude\": " + String(latitude, 6) + ",
\"longitude\": " + String(longitude, 6) + "}";



    // Send the POST request

    int httpResponseCode = http.POST(jsonPayload);



    // Handle the HTTP response

    if (httpResponseCode > 0) {

      Serial.print("HTTP Response code: ");

      Serial.println(httpResponseCode);

      Serial.println("Data sent successfully.");

    } else {

      Serial.print("Error in sending POST request. Response code: ");

      Serial.println(httpResponseCode);

    }



    // End the HTTP connection

    http.end();

  } else {

    Serial.println("Wi-Fi not connected!");

  }
```

```
}
```

WIRING:k

Wiring:

Component	ESP32 Pin	Description
GPS TX	GPIO16	Data from GPS to ESP32
GPS RX	GPIO17	Data to GPS from ESP32
GPS VCC	3.3V or 5V	Power for GPS module
GPS GND	GND	Ground for GPS module
Button	GPIO4	Data from button to ESP32
Button	GND	Ground for button

WORKING ESP32 + Vibration + Ultrasonic + Buzzer (Beeps on a certain distance)

CODE:

```
// Pin configuration
int buzzerPin = 19;             // GPIO 19 for Passive Buzzer
int trigPin = 5;                // GPIO 5 for Ultrasonic Trigger Pin
int echoPin = 18;               // GPIO 18 for Ultrasonic Echo Pin
int vibrationPin = 23;          // GPIO 23 for Vibration Module

// Threshold distance in cm
const int distanceThreshold = 30;  // Trigger at 30 cm distance

// Resonant frequency for the passive buzzer (experimentally determined)
const int buzzerFrequency = 4000;  // Adjust the frequency for the buzzer

void setup() {
  pinMode(buzzerPin, OUTPUT);       // Set buzzer pin as output
  pinMode(trigPin, OUTPUT);         // Set trigger pin as output
  pinMode(echoPin, INPUT);          // Set echo pin as input
  pinMode(vibrationPin, OUTPUT);    // Set vibration module pin as output

  Serial.begin(115200);             // Start serial communication for debugging
}

void loop() {
  long duration;
  int distance;

  // Send a 10us pulse to trigger pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the echo pin and calculate distance
  duration = pulseIn(echoPin, HIGH);
```

```
  distance = duration * 0.034 / 2;  // Convert the duration to distance
in cm

  // Print distance for debugging
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  if (distance > 0 && distance < distanceThreshold) {
    // Activate buzzer and vibration module
    tone(buzzerPin, buzzerFrequency);  // Set buzzer to its resonant
frequency
    digitalWrite(vibrationPin, HIGH);  // Turn on the vibration module
  } else {
    // Deactivate buzzer and vibration module
    noTone(buzzerPin);                 // Turn off the buzzer
    digitalWrite(vibrationPin, LOW);  // Turn off the vibration module
  }

  delay(100);  // Small delay to avoid excessive processing
}
```

WIRING:

Components:

1. ESP32 Development Board
2. Passive Buzzer
3. Ultrasonic Sensor (HC-SR04 or similar)
4. Vibration Module (Motor or DC Vibration Motor)

Wiring:

1. **Passive Buzzer:**
   - Buzzer Pin (GPIO 19) to Buzzer Positive (+).
   - Buzzer Negative (-) to GND on the ESP32.
2. **Ultrasonic Sensor (HC-SR04):**
   - VCC to 5V on the ESP32 (or 3.3V if your sensor operates at 3.3V).
   - GND to GND on the ESP32.
   - Trig Pin (GPIO 5) to TRIG pin on the ultrasonic sensor.
   - Echo Pin (GPIO 18) to ECHO pin on the ultrasonic sensor.
3. **Vibration Module:**
   - Vibration Motor's Positive (VCC) to GPIO 23.

    - Vibration Motor's Negative (GND) to GND on the ESP32.

Summary of Pin Connections:

- **Buzzer:**
    - GPIO 19 (ESP32) → Buzzer Positive (+)
    - Buzzer Negative (-) → GND (ESP32)
- **Ultrasonic Sensor (HC-SR04):**
    - VCC → 5V (or 3.3V if required) (ESP32)
    - GND → GND (ESP32)
    - GPIO 5 → TRIG Pin (Ultrasonic Sensor)
    - GPIO 18 → ECHO Pin (Ultrasonic Sensor)
- **Vibration Module:**
    - GPIO 23 → Vibration Motor Positive (VCC)
    - Vibration Motor Negative (GND) → GND (ESP32)

WORKING ESP32 + ALL MODULES WORKING

```
#include <WiFi.h>             // Include the Wi-Fi library for ESP32
#include <TinyGPS++.h>        // Include the TinyGPS++ library
#include <HTTPClient.h>       // Include HTTPClient library for making
HTTP requests

TinyGPSPlus gps;              // Create a GPS object
HardwareSerial gpsSerial(1);  // Use UART1 for GPS communication

// GPIO Pin Definitions
const int buttonPin = 4;       // GPIO pin for the button
const int buzzerPin = 19;      // GPIO 19 for Passive Buzzer
const int trigPin = 5;         // GPIO 5 for Ultrasonic Trigger Pin
const int echoPin = 18;        // GPIO 18 for Ultrasonic Echo Pin
const int vibrationPin = 23;   // GPIO 23 for Vibration Module

// Ultrasonic Sensor Settings
const int distanceThreshold = 30;  // Trigger at 30 cm distance
const int buzzerFrequency = 4000;  // Resonant frequency for buzzer

// Wi-Fi Credentials
const char* ssid = "Alba Extender 24";       // Replace with your Wi-Fi
SSID
const char* password = "ExtenderNova24";     // Replace with your Wi-Fi
password

// API Endpoint URL
const char* serverURL =
"https://elemsys-api.vercel.app/save-coordinates"; // URL endpoint

// Function Declarations
void handleUltrasonicSensor();
void handleGPS();
void sendDataToServer(float latitude, float longitude);

void setup() {
  Serial.begin(115200);       // Start the Serial Monitor
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);  // Start the GPS Serial
(TX = 17, RX = 16)

  // Configure GPIO pins
```

```
  pinMode(buttonPin, INPUT_PULLUP);  // Configure the button pin as
input with pull-up resistor
  pinMode(buzzerPin, OUTPUT);        // Set buzzer pin as output
  pinMode(trigPin, OUTPUT);          // Set trigger pin as output
  pinMode(echoPin, INPUT);           // Set echo pin as input
  pinMode(vibrationPin, OUTPUT);     // Set vibration module pin as
output

  // Wi-Fi Setup
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);  // Start the Wi-Fi connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nWi-Fi connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());  // Print the ESP32's IP address
}

void loop() {
  handleUltrasonicSensor();  // Check and handle ultrasonic sensor
  handleGPS();               // Check and handle GPS
}

// Function to handle the ultrasonic sensor
void handleUltrasonicSensor() {
  long duration;
  int distance;

  // Send a 10us pulse to trigger pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);


  // Read the echo pin and calculate distance
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;  // Convert the duration to distance
in cm
```

```
  // Print distance for debugging



  if (distance > 0 && distance < distanceThreshold) {
    // Activate buzzer and vibration module
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    tone(buzzerPin, buzzerFrequency);  // Set buzzer to its resonant
frequency
    digitalWrite(vibrationPin, HIGH);  // Turn on the vibration module
  } else {
    // Deactivate buzzer and vibration module
    noTone(buzzerPin);                 // Turn off the buzzer
    digitalWrite(vibrationPin, LOW);  // Turn off the vibration module
  }


  delay(100);  // Small delay to avoid excessive processing
}




// Function to handle GPS and send data
void handleGPS() {
  while (gpsSerial.available() > 0) {
    char c = gpsSerial.read();
    gps.encode(c);  // Feed the GPS data into the TinyGPS++ library
  }

  int buttonState = digitalRead(buttonPin);
  if (buttonState == LOW) {  // Button is pressed
    if (gps.location.isValid()) {
      float latitude = gps.location.lat();
      float longitude = gps.location.lng();

      // Print GPS coordinates to Serial Monitor
      Serial.print("Latitude: ");
      Serial.println(latitude, 6);
      Serial.print("Longitude: ");
```

```
      Serial.println(longitude, 6);

      // Send GPS coordinates to the server
      sendDataToServer(latitude, longitude);
    } else {
      Serial.println("Waiting for GPS signal...");
    }
    delay(1000);  // Debounce delay
  }
}

// Function to send data to the server
void sendDataToServer(float latitude, float longitude) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Prepare the HTTP POST request
    http.begin(serverURL);  // Specify server URL
    http.addHeader("Content-Type", "application/json"); // Set content
type to JSON

    // Prepare the JSON payload with latitude and longitude
    String jsonPayload = "{\"latitude\": " + String(latitude, 6) + ",
\"longitude\": " + String(longitude, 6) + "}";

    // Send the POST request
    int httpResponseCode = http.POST(jsonPayload);

    // Handle the HTTP response
    if (httpResponseCode > 0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      Serial.println("Data sent successfully.");
    } else {
      Serial.print("Error in sending POST request. Response code: ");
      Serial.println(httpResponseCode);
    }

    // End the HTTP connection
    http.end();
  } else {
    Serial.println("Wi-Fi not connected!");
  }
```

```
}
```

Component	Pin/Connection	ESP32 Pin
GPS Module (TX)	TX (GPS) → RX (ESP32)	GPIO 16
GPS Module (RX)	RX (GPS) → TX (ESP32)	GPIO 17
Ultrasonic Sensor (VCC)	VCC (Ultrasonic) → 3.3V	3.3V (ESP32)
Ultrasonic Sensor (GND)	GND (Ultrasonic) → GND	GND (ESP32)
Ultrasonic Sensor (Trig)	Trig (Ultrasonic) → GPIO 5	GPIO 5
Ultrasonic Sensor (Echo)	Echo (Ultrasonic) → GPIO 18	GPIO 18
Passive Buzzer (VCC)	VCC (Buzzer) → 3.3V	3.3V (ESP32)
Passive Buzzer (GND)	GND (Buzzer) → GND	GND (ESP32)
Passive Buzzer (Signal)	Signal (Buzzer) → GPIO 19	GPIO 19
Vibration Module (VCC)	VCC (Vibration) → 3.3V	3.3V (ESP32)
Vibration Module (GND)	GND (Vibration) → GND	GND (ESP32)
Vibration Module (Signal)	Signal (Vibration) → GPIO 23	GPIO 23
Push Button (One side)	Button → GPIO 4	GPIO 4
Push Button (Other side)	Button → GND	GND (ESP32)

This wiring setup connects your components to the ESP32, allowing it to function with the ultrasonic

FINAL EMBEDDED SYSTEM BLIND-STICK CODE AND WIRING

WORKING ESP32 + ALL MODULES WORKING + LED/WIFI INDICATOR

```
#include <WiFi.h>             // Include the Wi-Fi library for ESP32
#include <TinyGPS++.h>        // Include the TinyGPS++ library
#include <HTTPClient.h>       // Include HTTPClient library for making
HTTP requests

TinyGPSPlus gps;              // Create a GPS object
HardwareSerial gpsSerial(1);  // Use UART1 for GPS communication

// GPIO Pin Definitions
const int buttonPin = 4;        // GPIO pin for the button
const int buzzerPin = 19;       // GPIO 19 for Passive Buzzer
const int trigPin = 5;          // GPIO 5 for Ultrasonic Trigger Pin
const int echoPin = 18;         // GPIO 18 for Ultrasonic Echo Pin
const int vibrationPin = 23;    // GPIO 23 for Vibration Module
const int led1 = 21;            // GPIO 21 for Wi-Fi Connected Indicator
LED
const int led2 = 22;            // GPIO 22 for Wi-Fi Not Connected
Indicator LED

// Ultrasonic Sensor Settings
const int distanceThreshold = 30;  // Trigger at 30 cm distance
const int buzzerFrequency = 4000;  // Resonant frequency for buzzer

// Wi-Fi Credentials
const char* ssid = "Alba Extender 24";       // Replace with your Wi-Fi
SSID
const char* password = "ExtenderNova24";     // Replace with your Wi-Fi
password

// API Endpoint URL
const char* serverURL =
"https://elemsys-api.vercel.app/save-coordinates"; // URL endpoint

// Function Declarations
void handleUltrasonicSensor();
void handleGPS();
void sendDataToServer(float latitude, float longitude);
void updateWiFiStatusLED();

void setup() {
  Serial.begin(115200);       // Start the Serial Monitor
```

```
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);  // Start the GPS Serial
(TX = 17, RX = 16)

  // Configure GPIO pins
  pinMode(buttonPin, INPUT_PULLUP);  // Configure the button pin as
input with pull-up resistor
  pinMode(buzzerPin, OUTPUT);        // Set buzzer pin as output
  pinMode(trigPin, OUTPUT);          // Set trigger pin as output
  pinMode(echoPin, INPUT);           // Set echo pin as input
  pinMode(vibrationPin, OUTPUT);     // Set vibration module pin as
output
  pinMode(led1, OUTPUT);             // Set Wi-Fi Connected LED pin as
output
  pinMode(led2, OUTPUT);             // Set Wi-Fi Not Connected LED pin
as output

  // Wi-Fi Setup
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);  // Start the Wi-Fi connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
    updateWiFiStatusLED(); // Update Wi-Fi status LEDs
  }
  Serial.println("\nWi-Fi connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());  // Print the ESP32's IP address
}

void loop() {
  updateWiFiStatusLED();      // Update Wi-Fi status LEDs in the loop
  handleUltrasonicSensor();   // Check and handle ultrasonic sensor
  handleGPS();                // Check and handle GPS
}

// Enhanced function to update Wi-Fi status LEDs in real-time
void updateWiFiStatusLED() {
  static bool wasConnected = false; // Track previous Wi-Fi connection
state

  if (WiFi.status() == WL_CONNECTED) {
    if (!wasConnected) { // Check if connection state has changed to
connected
```

```
      Serial.println("Wi-Fi connected!");
      Serial.print("IP Address: ");
      Serial.println(WiFi.localIP());
      wasConnected = true;
    }
    digitalWrite(led1, HIGH); // Turn on Wi-Fi Connected LED
    digitalWrite(led2, LOW);  // Turn off Wi-Fi Not Connected LED
  } else {
    if (wasConnected) { // Check if connection state has changed to
disconnected
      Serial.println("Wi-Fi disconnected!");
      wasConnected = false;
    }
    digitalWrite(led1, LOW);  // Turn off Wi-Fi Connected LED
    digitalWrite(led2, HIGH); // Turn on Wi-Fi Not Connected LED
  }
}

// Function to handle the ultrasonic sensor
void handleUltrasonicSensor() {
  long duration;
  int distance;

  // Send a 10us pulse to trigger pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the echo pin and calculate distance
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;  // Convert the duration to distance
in cm

  // Print distance for debugging
  if (distance > 0 && distance < distanceThreshold) {
    // Activate buzzer and vibration module
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    tone(buzzerPin, buzzerFrequency);  // Set buzzer to its resonant
frequency
```

```
    digitalWrite(vibrationPin, HIGH);  // Turn on the vibration module
  } else {
    // Deactivate buzzer and vibration module
    noTone(buzzerPin);                 // Turn off the buzzer
    digitalWrite(vibrationPin, LOW);  // Turn off the vibration module
  }

  delay(100);  // Small delay to avoid excessive processing
}

// Function to handle GPS and send data
void handleGPS() {
  while (gpsSerial.available() > 0) {
    char c = gpsSerial.read();
    gps.encode(c);  // Feed the GPS data into the TinyGPS++ library
  }

  int buttonState = digitalRead(buttonPin);
  if (buttonState == LOW) {  // Button is pressed
    if (gps.location.isValid()) {
      float latitude = gps.location.lat();
      float longitude = gps.location.lng();

      // Print GPS coordinates to Serial Monitor
      Serial.print("Latitude: ");
      Serial.println(latitude, 6);
      Serial.print("Longitude: ");
      Serial.println(longitude, 6);

      // Send GPS coordinates to the server
      sendDataToServer(latitude, longitude);
    } else {
      Serial.println("Waiting for GPS signal...");
    }
    delay(1000);  // Debounce delay
  }
}

// Function to send data to the server
void sendDataToServer(float latitude, float longitude) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
```

```
    // Prepare the HTTP POST request
    http.begin(serverURL);  // Specify server URL
    http.addHeader("Content-Type", "application/json"); // Set content
type to JSON

    // Prepare the JSON payload with latitude and longitude
    String jsonPayload = "{\"latitude\": " + String(latitude, 6) + ",
\"longitude\": " + String(longitude, 6) + "}";

    // Send the POST request
    int httpResponseCode = http.POST(jsonPayload);

    // Handle the HTTP response
    if (httpResponseCode > 0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      Serial.println("Data sent successfully.");
    } else {
      Serial.print("Error in sending POST request. Response code: ");
      Serial.println(httpResponseCode);
    }

    // End the HTTP connection
    http.end();
  } else {
    Serial.println("Wi-Fi not connected!");
  }
}
```

Component	Pin/Connection	ESP32 Pin
GPS Module (TX)	TX (GPS) → RX (ESP32)	GPIO 16
GPS Module (RX)	RX (GPS) → TX (ESP32)	GPIO 17
Button	One side (Button) → GPIO 4	GPIO 4
Passive Buzzer (VCC)	VCC (Buzzer) → 3.3V	3.3V (ESP32)
Passive Buzzer (GND)	GND (Buzzer) → GND	GND (ESP32)
Passive Buzzer (Signal)	Signal (Buzzer) → GPIO 19	GPIO 19
Ultrasonic Sensor (VCC)	VCC (Ultrasonic) → 3.3V	3.3V (ESP32)
Ultrasonic Sensor (GND)	GND (Ultrasonic) → GND	GND (ESP32)
Ultrasonic Sensor (Trig)	Trig (Ultrasonic) → GPIO 5	GPIO 5
Ultrasonic Sensor (Echo)	Echo (Ultrasonic) → GPIO 18	GPIO 18
Vibration Module (VCC)	VCC (Vibration) → 3.3V	3.3V (ESP32)
Vibration Module (GND)	GND (Vibration) → GND	GND (ESP32)
Vibration Module (Signal)	Signal (Vibration) → GPIO 23	GPIO 23
LED (Wi-Fi Connected)	LED (Wi-Fi Connected) → GPIO 21	GPIO 21
LED (Wi-Fi Not Connected)	LED (Wi-Fi . Connected) → GPIO 22	GPIO 22