product 1 code:

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>
// GPIO where the DS18B20 is connected to
const int oneWireBus = 21;

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);
#define bleServerName "ESP32"
#define DOUT  5 // GPIO 5 (D1)
#define CLK   4 // GPIO 4 (D2)

#include <HX711.h>

HX711 scale;

float calibration_factor = 101.3; // New calibration factor
float temp;
float press;
unsigned long lastTime = 0;
unsigned long timerDelay = 3000;
bool deviceConnected = false;
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"
  BLECharacteristic bmpTemperatureCelsiusCharacteristics("cba1d466-344c-4be3-ab3f-189f80dd7518",
BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmpTemperatureCelsiusDescriptor(BLEUUID((uint16_t)0x2902));
```

```
BLECharacteristic bmppressureCharacteristics("ca73b3ba-39f6-4ab3-91ae-186dc9577d99",
BLECharacteristic::PROPERTY_NOTIFY);
BLEDescriptor bmppressureDescriptor(BLEUUID((uint16_t)0x2903));

//Setup callbacks onConnect and onDisconnect
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};

void setup() {
  Serial.begin(115200); // Initialize serial communication
  sensors.begin();
  scale.begin(DOUT, CLK);
  scale.tare(); // Reset to 0
    BLEDevice::init(bleServerName);

  // Create the BLE Server
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *bmpService = pServer->createService(SERVICE_UUID);

  // Create BLE Characteristics and Create a BLE Descriptor
  // Temperature
    bmpService->addCharacteristic(&bmpTemperatureCelsiusCharacteristics);
    bmpTemperatureCelsiusDescriptor.setValue("BME temperature Celsius");
    bmpTemperatureCelsiusCharacteristics.addDescriptor(&bmpTemperatureCelsiusDescriptor);
```

```
  bmpService->addCharacteristic(&bmppressureCharacteristics);
  bmppressureDescriptor.setValue("BMP pressure");
  bmppressureCharacteristics.addDescriptor(new BLE2902());

  // Start the service
  bmpService->start();

  // Start advertising
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

void loop() {
if (deviceConnected) {
    if ((millis() - lastTime) > timerDelay) {
    Serial.print("Pressure: ");
   float reading = scale.get_units();
   float pressure_pa = abs(reading) * calibration_factor; // Convert to Pascals
   float pressure_kpa = pressure_pa / 1000.0; // Convert to kilopascals

    press =pressure_kpa ;
    Serial.print(press);
    Serial.println(" hPa");
    sensors.requestTemperatures();
    float temperatureC = sensors.getTempCByIndex(0);
    temp = temperatureC;
    Serial.print("Temperature: ");
    Serial.print(temp);
    Serial.println(" C");
     //Notify temperature reading from BME sensor
       static char temperatureCTemp[6];
```

```
      dtostrf(temp, 6, 2, temperatureCTemp);
      //Set temperature Characteristic value and notify connected client
      bmpTemperatureCelsiusCharacteristics.setValue(temperatureCTemp);
      bmpTemperatureCelsiusCharacteristics.notify();
    static char pressureTemp[6];
    dtostrf(press, 6, 2, pressureTemp);
    bmppressureCharacteristics.setValue(pressureTemp);
    bmppressureCharacteristics.notify();
    lastTime = millis();
   }
 }
}
```

Product 2 :

```
#include "BLEDevice.h"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <LoRa.h>
//BLE Server name (the other ESP32 name running the server sketch)
#define bleServerName "ESP32"
#define SDA_PIN 21
#define SCL_PIN 22
// Define LoRa parameters for ESP32
#define SS 5       // GPIO5 - NSS
#define RST 14     // GPIO14 - RESET
#define DIO0 26    // GPIO26 - DIO0

int buzzPin = 4;
int tempInt;
int pressureInt;
/* UUID's of the service, characteristic that we want to read*/
// BLE Service
static BLEUUID bmeServiceUUID("91bad492-b950-4226-aa2b-4ede9fa42f59");
```

```
int lcdColumns = 20;
int lcdRows = 4;
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

// BLE Characteristics

  //Temperature Celsius Characteristic
static BLEUUID temperatureCharacteristicUUID("cba1d466-344c-4be3-ab3f-189f80dd7518");
// Humidity Characteristic
static BLEUUID pressureCharacteristicUUID("ca73b3ba-39f6-4ab3-91ae-186dc9577d99");

//Flags stating if should begin connecting and if the connection is up
static boolean doConnect = false;
static boolean connected = false;

//Address of the peripheral device. Address will be found during scanning...
static BLEAddress *pServerAddress;

//Characteristicd that we want to read
static BLERemoteCharacteristic* temperatureCharacteristic;
static BLERemoteCharacteristic* pressureCharacteristic;

//Activate notify
const uint8_t notificationOn[] = {0x1, 0x0};
const uint8_t notificationOff[] = {0x0, 0x0};

//Variables to store temperature and humidity
char* temperatureChar;
char* pressureChar;

//Flags to check whether new temperature and humidity readings are available
boolean newTemperature = false;
boolean newpressure = false;
```

```
//Connect to the BLE Server that has the name, Service, and Characteristics
bool connectToServer(BLEAddress pAddress) {
   BLEClient* pClient = BLEDevice::createClient();

  // Connect to the remove BLE Server.
  pClient->connect(pAddress);
  Serial.println(" - Connected to server");

  // Obtain a reference to the service we are after in the remote BLE server.
  BLERemoteService* pRemoteService = pClient->getService(bmeServiceUUID);
  if (pRemoteService == nullptr) {
    Serial.print("Failed to find our service UUID: ");
    Serial.println(bmeServiceUUID.toString().c_str());
    return (false);
  }

  // Obtain a reference to the characteristics in the service of the remote BLE server.
  temperatureCharacteristic = pRemoteService->getCharacteristic(temperatureCharacteristicUUID);
  pressureCharacteristic = pRemoteService->getCharacteristic(pressureCharacteristicUUID);

  if (temperatureCharacteristic == nullptr || pressureCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID");
    return false;
  }
  Serial.println(" - Found our characteristics");

  //Assign callback functions for the Characteristics
  temperatureCharacteristic->registerForNotify(temperatureNotifyCallback);
  pressureCharacteristic->registerForNotify(pressureNotifyCallback);
  return true;
}

//Callback function that gets called, when another device's advertisement has been received
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
```

```
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    if (advertisedDevice.getName() == bleServerName) { //Check if the name of the advertiser matches
      advertisedDevice.getScan()->stop(); //Scan can be stopped, we found what we are looking for
      pServerAddress = new BLEAddress(advertisedDevice.getAddress()); //Address of advertiser is the one we
need
      doConnect = true; //Set indicator, stating that we are ready to connect
      Serial.println("Device found. Connecting!");
    }
  }
};

//When the BLE Server sends a new temperature reading with the notify property
static void temperatureNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                      uint8_t* pData, size_t length, bool isNotify) {
  //store temperature value
  temperatureChar = (char*)pData;
  newTemperature = true;
}

//When the BLE Server sends a new humidity reading with the notify property
static void pressureNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                   uint8_t* pData, size_t length, bool isNotify) {
  //store humidity value
  pressureChar = (char*)pData;
  newpressure = true;
  Serial.print(newpressure);
}

void setup() {
  LoRa.setPins(SS, RST, DIO0);

  // Set frequency to 433 MHz (or the frequency appropriate for your region)
  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
```

```
    while (1);
  }
  Serial.println("LoRa initialized.");
  Wire.begin(SDA_PIN, SCL_PIN);
  pinMode(buzzPin, OUTPUT);
  // initialize LCD
  lcd.init();
  // turn on LCD backlight
  lcd.backlight();
  //Start serial communication
  Serial.begin(115200);
  Serial.println("Starting Arduino BLE Client application...");

  //Init BLE device
  BLEDevice::init("");

  // Retrieve a Scanner and set the callback we want to use to be informed when we
  // have detected a new device.  Specify that we want active scanning and start the
  // scan to run for 30 seconds.
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30);
}

void loop() {
  if (doConnect == true) {
    if (connectToServer(*pServerAddress)) {
      Serial.println("We are now connected to the BLE Server.");
      //Activate the Notify property of each Characteristic
      temperatureCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))-
>writeValue((uint8_t*)notificationOn, 2, true);
      pressureCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))->writeValue((uint8_t*)notificationOn, 2,
true);
```

```
    connected = true;
   } else {
    Serial.println("We have failed to connect to the server; Restart your device to scan for nearby BLE server again.");
   }
   doConnect = false;
  }

  // Convert received temperature and pressure values to float and then to integer
  if (newTemperature && newpressure) {
   newTemperature = false;
   newpressure = false;

   // Convert char* to float
   float tempFloat = atof(temperatureChar);
   float pressureFloat = atof(pressureChar);

   // Convert float to int
    tempInt = static_cast<int>(tempFloat);
    pressureInt = static_cast<int>(pressureFloat);

   // Print the integer values to the serial monitor
   Serial.print("Temperature: ");
   Serial.print(tempInt);
   Serial.println("C");

   Serial.print("Pressure: ");
   Serial.print(pressureInt);
   Serial.println("%");
  }
  int frontLeftPressure = pressureInt;  // Example pressure in PSI
  int frontLeftTemp = tempInt;      // Example temperature in °C

  // Display Front Left Tire Pressure and Temperature
```

```
  lcd.setCursor(0, 0);
  lcd.print("FLP: ");
  lcd.print(frontLeftPressure);
  lcd.print(" KPa ");
  lcd.setCursor(0, 1);
  lcd.print("FLT: ");
  lcd.print(frontLeftTemp);
  lcd.print(" C");
  if(frontLeftPressure > 1000){
    digitalWrite(buzzPin, HIGH);
    delay(2000);
    digitalWrite(buzzPin, LOW);
  }
  if(frontLeftTemp > 35){
    digitalWrite(buzzPin, HIGH);
    delay(2000);
    digitalWrite(buzzPin, LOW);
  }
  int sensor1Value = frontLeftPressure; // Replace with actual sensor reading
  int sensor2Value = frontLeftTemp; // Replace with actual sensor reading

  // Format sensor data into a string
  String sensorData = String(sensor1Value) + "," + String(sensor2Value);

  // Send a LoRa packet with sensor data
  Serial.println("Sending packet...");
  LoRa.beginPacket();
  LoRa.print(sensorData); // Send formatted sensor data
  LoRa.endPacket();
  delay(1000); // Delay a second between loops.
}
```

Product 3:

```
#include <ESP8266WiFi.h>
#include <Firebase_ESP_Client.h>
#include <SPI.h>
#include <LoRa.h>
#define SS D8        // GPIO15 - NSS
#define RST D0       // GPIO16 - RESET
#define DIO0 D1
// Provide the token generation process info.
#include "addons/TokenHelper.h"
// Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"
int sensor1Value = 0;
int sensor2Value = 0;
// Insert your network credentials
#define WIFI_SSID "OPPO K10"
#define WIFI_PASSWORD "avanti13"

// Insert Firebase project API Key
#define API_KEY "AIzaSyDzHh_NFw7u4sZyL-WmolXdMMjWu_Ozko0"

// Insert RTDB URL
#define DATABASE_URL "https://myproject-8bd37-default-rtdb.asia-southeast1.firebasedatabase.app/"
#define USER_EMAIL "srilekhaj9171@gmail.com"
#define USER_PASSWORD "9514176533@Js"
// Define Firebase Data object
FirebaseData fbdo;

FirebaseAuth auth;
FirebaseConfig config;
bool signupOK = false;
unsigned long sendDataPrevMillis = 0;
uint32_t sendInterval = 1000;
void setup() {
  Serial.begin(115200);
```

```
LoRa.setPins(SS, RST, DIO0);

// Set frequency to 433 MHz (or the frequency appropriate for your region)
if (!LoRa.begin(433E6)) {
  Serial.println("Starting LoRa failed!");
  while (1);
}
Serial.println("LoRa initialized.");

// Start receiving LoRa data
LoRa.receive();
delay(2000);
// Connect to Wi-Fi
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();

// Set up Firebase configuration
config.api_key = API_KEY;
config.database_url = DATABASE_URL;
auth.user.email = USER_EMAIL;
auth.user.password = USER_PASSWORD;

// Sign up
if (Firebase.signUp(&config, &auth, "", "")) {
  Serial.println("ok");
  signupOK = true;
```

```
  } else {
    Serial.printf("%s\n", config.signer.signupError.message.c_str());
  }

  // Set the callback function for the long running token generation task
  config.token_status_callback = tokenStatusCallback; // see addons/TokenHelper.h
  fbdo.setBSSLBufferSize(4096, 1024);

  // Initialize Firebase
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

void loop() {
  // Check if a packet has been received
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // Read the packet and convert to string
    String receivedData = "";
    while (LoRa.available()) {
      receivedData += (char)LoRa.read();
    }
    int commaIndex = receivedData.indexOf(',');
    if (commaIndex != -1) {
      String sensor1Data = receivedData.substring(0, commaIndex);
      String sensor2Data = receivedData.substring(commaIndex + 1);

      // Debug: Print parsed data
      Serial.print("Parsed Sensor 1 Data: ");
      Serial.println(sensor1Data);
      Serial.print("Parsed Sensor 2 Data: ");
      Serial.println(sensor2Data);

      // Convert to integers
```

```
      int sensor1Value = sensor1Data.toInt();
      int sensor2Value = sensor2Data.toInt();

      // Print the sensor values
      Serial.print("Sensor 1 Value: ");
      Serial.println(sensor1Value);
      Serial.print("Sensor 2 Value: ");
      Serial.println(sensor2Value);
    } else {
      Serial.println("Data format error: No comma found.");
    }

    // Prepare for next packet
    LoRa.receive();
    delay(5000);
  }
  if (Firebase.ready() && (millis() - sendDataPrevMillis > sendInterval || sendDataPrevMillis == 0)) {
    sendDataPrevMillis = millis();

    // Send first sensor data
    if (Firebase.RTDB.setInt(&fbdo, "/truck/tyre/sensor1/data", sensor1Value)) {
      Serial.println("Sensor 1 data sent successfully.");
    } else {
      Serial.println(fbdo.errorReason());
    }

    // Send second sensor data
    if (Firebase.RTDB.setInt(&fbdo, "/truck/tyre/sensor2/data", sensor2Value)) {
      Serial.println("Sensor 2 data sent successfully.");
    } else {
      Serial.println(fbdo.errorReason());
    }
}
}
```