1 Problem: BLE Connection

1. Use your IMU and proximity sensors on Arduino to read data every 1 second.
2. Set your Arduino as a BLE sever publishing the accelerometer and the proximity sensor data in two separate characteristics. (For acceleration only publish the measurement on the x-axis. So each characteristic would report one number per measurement)

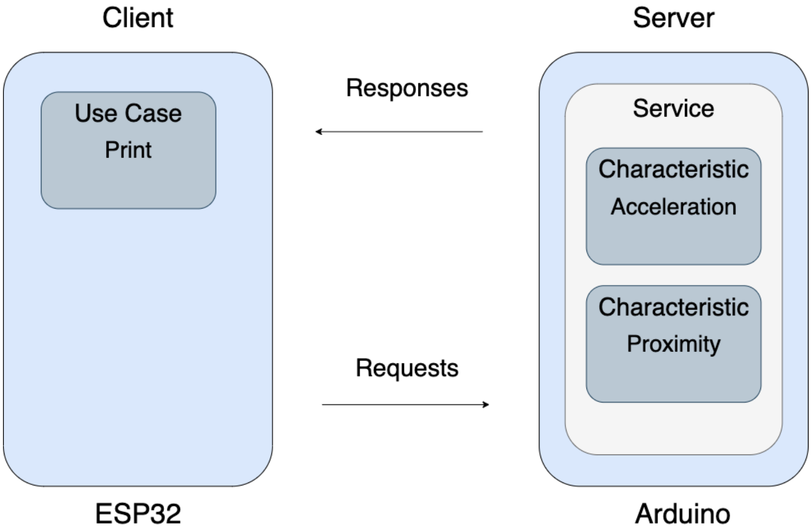
```
#include <ArduinoBLE.h>
#include <Arduino_LSM9DS1.h>
#include <Arduino_APDS9960.h>

BLEService ServiceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
BLEByteCharacteristic ProximityCharUuid("beb5483e-36e1-4688-b7f5-ea07361b26a8", BLERead | ↩
    BLENotify | BLEWrite);
BLEFloatCharacteristic XAccCharUuid("2ea70455-850e-498b-86c0-bd0f89d76b86", BLERead | ↩
    BLENotify | BLEWrite);

unsigned long initial_acc = 0;
unsigned long initial_prox = 0;
unsigned long previousMillis = 0;
float x, y, z;
int interval = 1000;

void setup() {
  Serial.begin(9600);
//  while (!Serial);

  if (!BLE.begin()) {
    Serial.println("* Starting BLE module failed!");
    while (1);
  }
  if (!IMU.begin()) {
    Serial.println("The IMU could not be initialized");
    while (1);
  }
  if (!APDS.begin()) {
    Serial.println("The proximity sensor could not be initialized");
    while (1);
  }

  BLE.setLocalName("Nano 33 BLE (Central) Carl");
  BLE.setAdvertisedService(ServiceUUID);

  ServiceUUID.addCharacteristic(ProximityCharUuid);
  ServiceUUID.addCharacteristic(XAccCharUuid);

  BLE.addService(ServiceUUID);
  ProximityCharUuid.writeValue(initial_prox);
  XAccCharUuid.writeValue(initial_acc);

  BLE.advertise();

  Serial.println("Arduino Nano 33 BLE Sense (Central Device)");
}

void getdata(){

  if (IMU.accelerationAvailable() & APDS.proximityAvailable()) {
      IMU.readAcceleration(x,y,z);
      XAccCharUuid.writeValue(x);
      Serial.print("Acceleration:");
      Serial.print(x);
      int proximity = APDS.readProximity();
      ProximityCharUuid.writeValue(proximity);
      Serial.println("Proximity:");
      Serial.print(proximity);
  }
}

void loop() {
    BLEDevice central = BLE.central();
    if (central){
      Serial.print("Connected to central");
      Serial.print(central.address());
      digitalWrite(LED_BUILTIN, HIGH);
      while (central.connected())
      {
        unsigned long currentMillis = millis();
        if (currentMillis - previousMillis >= interval)
        {
          getdata();
          previousMillis = currentMillis;
        }
      }
      digitalWrite(LED_BUILTIN, LOW);
      Serial.print("diconnected from central");
}
}
```

3. Set your ESP32 as a BLE client and make sure to receive both characteristics

```
#include "BLEDevice.h"
#include "BLEScan.h"

float AccelerationX = 0;
int Proximity = 0;

boolean regen_acc = false;
boolean regen_prox = false;

static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
static BLEUUID CharProximityUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
static BLEUUID CharAccelerationUUID("2ea70455-850e-498b-86c0-bd0f89d76b86");

static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* aRemoteCharacteristic;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;

static void notifyCallbackp( BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pDatap, size_t length, bool isNotify) {
    Proximity = *pDatap;
    regen_prox = true;
}

static void notifyCallbacka( BLERemoteCharacteristic* aBLERemoteCharacteristic, uint8_t* pDataa, size_t length, bool isNotify) {
    AccelerationX  = *(float*)pDataa;
    regen_acc = true;
}

class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient* pclient) {
  }

  void onDisconnect(BLEClient* pclient) {
    connected = false;
    Serial.println("onDisconnect");
  }
};

bool connectToServer() {
    Serial.print("Forming a connection to ");
    Serial.println(myDevice->getAddress().toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new MyClientCallback());

    pClient->connect(myDevice);
    Serial.println(" - Connected to server");

    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    Serial.print("passed");
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our service");

    aRemoteCharacteristic = pRemoteService->getCharacteristic(CharAccelerationUUID);
    if (aRemoteCharacteristic == nullptr) {
      Serial.print("Failed to find our characteristic UUID: ");
      Serial.println(CharAccelerationUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our acceleration characteristic");

    if(aRemoteCharacteristic->canRead()) {
      std::string value = aRemoteCharacteristic->readValue();
      Serial.print("The acceleration characteristic value was: ");
      Serial.println(value.c_str());
    }

    pRemoteCharacteristic = pRemoteService->getCharacteristic(CharProximityUUID);
    if (pRemoteCharacteristic == nullptr) {
      Serial.print("Failed to find our characteristic UUID: ");
      Serial.println(CharProximityUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our proximity characteristic");

    if(pRemoteCharacteristic->canRead()) {
      std::string value = pRemoteCharacteristic->readValue();
      Serial.print("The proximity characteristic value was: ");
      Serial.println(value.c_str());
    }

    if(aRemoteCharacteristic->canNotify())
      aRemoteCharacteristic->registerForNotify(notifyCallbacka);
    if(pRemoteCharacteristic->canNotify())
      pRemoteCharacteristic->registerForNotify(notifyCallbackp);

    connected = true;

}

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {

  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());

    if (advertisedDevice.haveServiceUUID() && advertisedDevice.isAdvertisingService(serviceUUID)) {

      BLEDevice::getScan()->stop();
      myDevice = new BLEAdvertisedDevice(advertisedDevice);
      doConnect = true;
      doScan = true;

    }
  }
};


void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");

  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setInterval(1349);
  pBLEScan->setWindow(449);
  pBLEScan->setActiveScan(true);
  pBLEScan->start(300, false);
}

void loop() {

  if (doConnect == true) {
    if (connectToServer())
    { Serial.println("We are now connected to the BLE Server.");

    } else {
      Serial.println("We have failed to connect to the server; there is nothin more we will ↩
          do.");
    }
      doConnect = false;
  }

  if (regen_acc && regen_prox) {
    Serial.print("Acceleration:");
    Serial.println(AccelerationX);
    Serial.print("Proximity:");
    Serial.println(Proximity);
    regen_acc = false;
    regen_prox = false;
  }

}
```

4. Draw a diagram of your system and explain, in less than 4 sentences, how your BLE connection works.

   The ESP32 acting as a peripheral(client), will broadcast signals about itself to the surrounding area. The Arduino picks up on this broadcast and establishes a connection which allows for reading to occur. During reading, the peripheral (ESP32) asks the central device (Arduino) for information like a specific characteristic of a service. As the server, the Arduino acts as the master and the client acts as the slave.



2 Problem

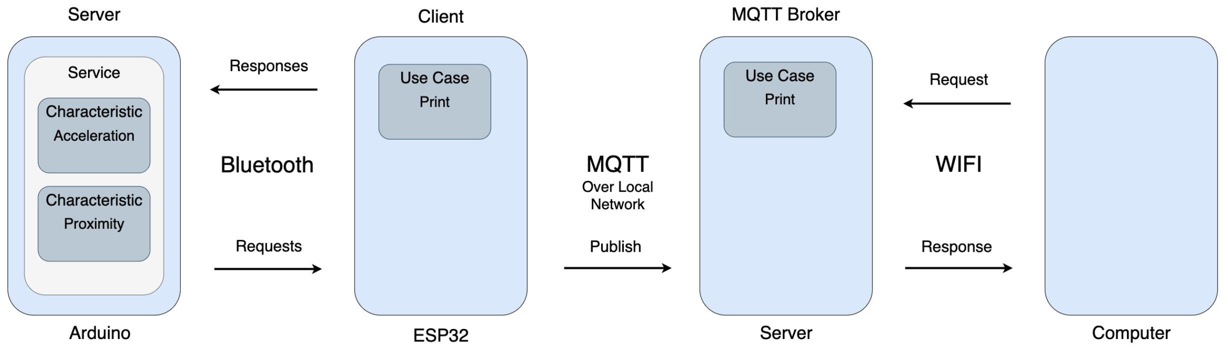
1. The data that you received from your Arduino is now sent to your cloud. Use two MQTT topics to publish your sensor data (one for each of your sensors)

```
#include "BLEDevice.h"
#include "BLEScan.h"
#include <UbiConstants.h>
#include <UbiTypes.h>
#include <UbidotsEsp32Mqtt.h>
#include <WiFi.h>

const char *UBIDOTS_TOKEN = "BBFF-uPnU0yq1G4EwIXV0nQqEmmQxxKyH67";
const char *WIFI_SSID = "iPhone";
const char *WIFI_PASS = "sasa8888";
const char *DEVICE_LABEL = "ESP32";
const char *ubiaccpoint = "AccelerationX";
const char *ubiproxpoint = "Proximity";

const int PUBLISH_FREQUENCY = 5000;
unsigned long previousMillis = 0;

float AccelerationX = 0;
int Proximity = 0;

boolean regen_acc = false;
boolean regen_prox = false;

static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
static BLEUUID CharProximityUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
static BLEUUID CharAccelerationUUID("2ea70455-850e-498b-86c0-bd0f89d76b86");

static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* aRemoteCharacteristic;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;
Ubidots ubidots(UBIDOTS_TOKEN);

static void notifyCallbackp( BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pDatap, size_t length, bool isNotify) {
    Proximity = *pDatap;
    regen_prox = true;
}

static void notifyCallbacka( BLERemoteCharacteristic* aBLERemoteCharacteristic, uint8_t* pDataa, size_t length, bool isNotify) {
    AccelerationX  = *(float*)pDataa;
    regen_acc = true;
}

class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient* pclient) {
  }

  void onDisconnect(BLEClient* pclient) {
    connected = false;
    Serial.println("onDisconnect");
  }
};

bool connectToServer() {
    Serial.print("Forming a connection to ");
    Serial.println(myDevice->getAddress().toString().c_str());

    BLEClient*  pClient  = BLEDevice::createClient();
    Serial.println(" - Created client");

    pClient->setClientCallbacks(new MyClientCallback());

    pClient->connect(myDevice);
    Serial.println(" - Connected to server");

    BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
    Serial.print("passed");
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(serviceUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our service");

    aRemoteCharacteristic = pRemoteService->getCharacteristic(CharAccelerationUUID);
    if (aRemoteCharacteristic == nullptr) {
      Serial.print("Failed to find our characteristic UUID: ");
      Serial.println(CharAccelerationUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our acceleration characteristic");

    if(aRemoteCharacteristic->canRead()) {
      std::string value = aRemoteCharacteristic->readValue();
      Serial.print("The acceleration characteristic value was: ");
      Serial.println(value.c_str());
    }

    pRemoteCharacteristic = pRemoteService->getCharacteristic(CharProximityUUID);
    if (pRemoteCharacteristic == nullptr) {
      Serial.print("Failed to find our characteristic UUID: ");
      Serial.println(CharProximityUUID.toString().c_str());
      pClient->disconnect();
      return false;
    }
    Serial.println(" - Found our proximity characteristic");

    if(pRemoteCharacteristic->canRead()) {
      std::string value = pRemoteCharacteristic->readValue();
      Serial.print("The proximity characteristic value was: ");
      Serial.println(value.c_str());
    }

    if(aRemoteCharacteristic->canNotify())
      aRemoteCharacteristic->registerForNotify(notifyCallbacka);
    if(pRemoteCharacteristic->canNotify())
      pRemoteCharacteristic->registerForNotify(notifyCallbackp);

    connected = true;
}

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {

  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("BLE Advertised Device found: ");
    Serial.println(advertisedDevice.toString().c_str());

    if (advertisedDevice.haveServiceUUID() && advertisedDevice.isAdvertisingService(↩
        serviceUUID)) {

      BLEDevice::getScan()->stop();
      myDevice = new BLEAdvertisedDevice(advertisedDevice);
      doConnect = true;
      doScan = true;

    }
  }
};


void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");
  ubidots.setDebug(true);
  ubidots.connectToWifi(WIFI_SSID, WIFI_PASS);

  ubidots.setup();
  ubidots.reconnect();

  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setInterval(1349);
  pBLEScan->setWindow(449);
  pBLEScan->setActiveScan(true);
  pBLEScan->start(300, false);
}


void loop() {

  if (doConnect == true) {
    if (connectToServer()) {
      Serial.println("We are now connected to the BLE Server.");
    }
    else {
      Serial.println("We have failed to connect to the server; there is nothin more we will do.");
    }
      doConnect = false;
  }
  if (!ubidots.connected()){
    ubidots.reconnect();
    }

  if (regen_acc && regen_prox) {
    Serial.print("Acceleration:");
    Serial.println(AccelerationX);
    Serial.print("Proximity:");
    Serial.println(Proximity);
    regen_acc = false;
    regen_prox = false;
  }

  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= PUBLISH_FREQUENCY) {
    ubidots.add(ubiaccpoint, AccelerationX);
    ubidots.add(ubiproxpoint, Proximity);
    previousMillis = currentMillis;
    ubidots.publish(DEVICE_LABEL);
  }
ubidots.loop();

}
```

2. Using the Ubidots platform now you can check your sensor data in real-time.

3. Complete your system diagram from problem 1. Add all your MQTT components and explain the overall system in less than 6 sentences.

   Ubidots acts as a MQTT broker, that means that the ESP32 publishes the acceleration float and the proximity string to the broker using the MQTT publish-and-subscribe protocol. This protocol works by first connecting to the broker and then publishing all using the existing local internet home network. Once Ubidots has received the data, the data from the bluetooth characteristic is stored online to a variable. The data can be then accessed and read using a smartphone or computer using the Wi-Fi protocol. Using radio waves Wi-Fi connects the device to the nearest router, which sends the request for the data to the ubidots.com host.


Server
Service
Characteristic
Acceleration
Characteristic
Proximity
Arduino
Responses
Bluetooth
Requests
Client
Use Case
Print
ESP32
MQTT
Over Local
Network
Publish
MQTT Broker
Use Case
Print
Server
Request
WIFI
Response
Computer