**GigGauge Software Design Document**

**Project Overview**

This project consists of two main components: an Arduino device that collects data from an ADXL345 accelerometer and a React Native Expo application that receives and displays this data via Bluetooth Low Energy (BLE).

**Objective**

* Collect accelerometer data using an Arduino device.
* Display this data on an LCD screen connected to the Arduino.
* Save the data to an SD card.
* Transmit the data via an HC-05 Bluetooth module.
* Receive and display the data on a React Native Expo app using the react-native-ble-plx library.

**System Architecture**

**Arduino Device**

**Components:**

* Arduino Board
* ADXL345 Accelerometer
* LCD Screen
* SD Card Module
* HC-05 Bluetooth Module

**Functions:**

1. **Data Collection**: The ADXL345 accelerometer collects acceleration data.
2. **Data Display**: The data is displayed on the LCD screen.
3. **Data Storage**: The data is saved to an SD card.
4. **Data Transmission**: The data is sent to the React Native app via the HC-05 Bluetooth module.

**React Native Expo App**

**Components:**

* react-native-ble-plx library for BLE communication.
* UI components to display the accelerometer data.

**Functions:**

1. **Permissions**: Request necessary permissions for BLE communication.
2. **Device Scanning**: Scan for available Bluetooth devices.
3. **Device Connection**: Connect to the Arduino device.
4. **Data Reception**: Receive and decode data from the Arduino device.
5. **Data Display**: Display the received data in the app.

**Detailed Design**

**Arduino Device**

**1. Circuit Diagram:**

* Connect the ADXL345 accelerometer to the Arduino board.
* Connect the LCD screen to the Arduino.
* Connect the SD card module to the Arduino.
* Connect the HC-05 Bluetooth module to the Arduino.

**2. Arduino Code:**

* Initialize the ADXL345 accelerometer, LCD screen, SD card module, and HC-05 Bluetooth module.
* Collect data from the ADXL345 accelerometer.
* Display the data on the LCD screen.
* Save the data to the SD card.
* Transmit the data via the HC-05 Bluetooth module.

cpp

Copy code

#include <Wire.h>

#include <LiquidCrystal.h>

#include <SD.h>

#include <SoftwareSerial.h>

#include <Adafruit\_ADXL345\_U.h>

Adafruit\_ADXL345\_Unified accel = Adafruit\_ADXL345\_Unified(12345);

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

SoftwareSerial bluetooth(10, 9); // RX, TX

File dataFile;

void setup() {

Serial.begin(9600);

lcd.begin(16, 2);

if (!accel.begin()) {

lcd.print("No ADXL345");

while (1);

}

accel.setRange(ADXL345\_RANGE\_16\_G);

if (!SD.begin(4)) {

lcd.print("No SD card");

while (1);

}

bluetooth.begin(9600);

lcd.print("Setup done");

delay(2000);

lcd.clear();

}

void loop() {

sensors\_event\_t event;

accel.getEvent(&event);

lcd.setCursor(0, 0);

lcd.print("X: "); lcd.print(event.acceleration.x);

lcd.setCursor(0, 1);

lcd.print("Y: "); lcd.print(event.acceleration.y);

dataFile = SD.open("datalog.txt", FILE\_WRITE);

if (dataFile) {

dataFile.print("X: "); dataFile.print(event.acceleration.x);

dataFile.print(", Y: "); dataFile.println(event.acceleration.y);

dataFile.close();

}

bluetooth.print("X: "); bluetooth.print(event.acceleration.x);

bluetooth.print(", Y: "); bluetooth.println(event.acceleration.y);

delay(1000);

}

**React Native Expo App**

**1. Project Setup:**

* Create a new Expo project.
* Install necessary dependencies: react-native-ble-plx, react-native-permissions, expo-permissions.

sh

Copy code

expo init BluetoothApp

cd BluetoothApp

npm install react-native-ble-plx react-native-permissions expo-permissions

**2. Implement BLE Functionality:**

* Create a custom hook (useBLE) to handle BLE operations.
* Request necessary permissions.
* Scan for and connect to the Arduino device.
* Receive and decode data from the Arduino device.

**useBLE.ts**

typescript

Copy code

import { useMemo, useState } from "react";

import { PermissionsAndroid, Platform } from "react-native";

import { BleError, BleManager, Characteristic, Device } from "react-native-ble-plx";

import \* as ExpoDevice from "expo-device";

import base64 from "react-native-base64";

const ARDUINO\_SERVICE\_UUID = "your-arduino-service-uuid";

const ARDUINO\_CHARACTERISTIC\_UUID = "your-arduino-characteristic-uuid";

interface BluetoothLowEnergyApi {

requestPermissions(): Promise<boolean>;

scanForPeripherals(): void;

connectToDevice: (device: Device) => Promise<void>;

disconnectFromDevice: () => void;

connectedDevice: Device | null;

allDevices: Device[];

arduinoData: string;

}

function useBLE(): BluetoothLowEnergyApi {

const bleManager = useMemo(() => new BleManager(), []);

const [allDevices, setAllDevices] = useState<Device[]>([]);

const [connectedDevice, setConnectedDevice] = useState<Device | null>(null);

const [arduinoData, setArduinoData] = useState<string>("");

const requestAndroid31Permissions = async () => {

const bluetoothScanPermission = await PermissionsAndroid.request(

PermissionsAndroid.PERMISSIONS.BLUETOOTH\_SCAN,

{

title: "Location Permission",

message: "Bluetooth Low Energy requires Location",

buttonPositive: "OK",

}

);

const bluetoothConnectPermission = await PermissionsAndroid.request(

PermissionsAndroid.PERMISSIONS.BLUETOOTH\_CONNECT,

{

title: "Location Permission",

message: "Bluetooth Low Energy requires Location",

buttonPositive: "OK",

}

);

const fineLocationPermission = await PermissionsAndroid.request(

PermissionsAndroid.PERMISSIONS.ACCESS\_FINE\_LOCATION,

{

title: "Location Permission",

message: "Bluetooth Low Energy requires Location",

buttonPositive: "OK",

}

);

return (

bluetoothScanPermission === "granted" &&

bluetoothConnectPermission === "granted" &&

fineLocationPermission === "granted"

);

};

const requestPermissions = async () => {

if (Platform.OS === "android") {

if ((ExpoDevice.platformApiLevel ?? -1) < 31) {

const granted = await PermissionsAndroid.request(

PermissionsAndroid.PERMISSIONS.ACCESS\_FINE\_LOCATION,

{

title: "Location Permission",

message: "Bluetooth Low Energy requires Location",

buttonPositive: "OK",

}

);

return granted === PermissionsAndroid.RESULTS.GRANTED;

} else {

const isAndroid31PermissionsGranted =

await requestAndroid31Permissions();

return isAndroid31PermissionsGranted;

}

} else {

return true;

}

};

const isDuplicteDevice = (devices: Device[], nextDevice: Device) =>

devices.findIndex((device) => nextDevice.id === device.id) > -1;

const scanForPeripherals = () =>

bleManager.startDeviceScan(null, null, (error, device) => {

if (error) {

console.log(error);

return;

}

if (device && device.name?.includes("Arduino")) {

setAllDevices((prevState: Device[]) => {

if (!isDuplicteDevice(prevState, device)) {

return [...prevState, device];

}

return prevState;

});

}

});

const connectToDevice = async (device: Device) => {

try {

const deviceConnection = await bleManager.connectToDevice(device.id);

setConnectedDevice(deviceConnection);

await deviceConnection.discoverAllServicesAndCharacteristics();

bleManager.stopDeviceScan();

startStreamingData(deviceConnection);

} catch (e) {

console.log("FAILED TO CONNECT", e);

}

};

const disconnectFromDevice = () => {

if (connectedDevice) {

bleManager.cancelDeviceConnection(connectedDevice.id);

setConnectedDevice(null);

setArduinoData("");

}

};

const onDataUpdate = (

error: BleError | null,

characteristic: Characteristic | null

) => {

if (error) {

console.log(error);

return;

} else if (!characteristic?.value) {

console.log("No Data was received");

return;

}

const rawData = base64.decode(characteristic.value);

setArduinoData(rawData);

};

const startStreamingData = async (device: Device) => {

if (device) {

device.monitorCharacteristicForService(

ARDUINO\_SERVICE\_UUID,

ARDUINO\_CHARACTERISTIC\_UUID,

onDataUpdate

);

} else {

console.log("No Device Connected");

}

};

return {

scanForPeripherals,

requestPermissions,

connectToDevice,

allDevices,

connectedDevice,

disconnectFromDevice,

arduinoData,

};

}

export default useBLE;

**3. Update Main App Component:**

typescript

Copy code

// App.tsx

import React, { useEffect } from "react";

import { StyleSheet, Text, View, Button } from "react-native";

import useBLE from "./useBLE";

const App: React.FC = () => {

const {

requestPermissions,

scanForPeripherals,

connectToDevice,

allDevices,

disconnectFromDevice,

connectedDevice,

arduinoData,

} = useBLE();

useEffect(() => {

const init = async () => {

const granted = await requestPermissions();

if (granted) {

scanForPeripherals();

}

};

init();

}, []);

return (

<View style={styles.container}>

<Text style={styles.header}>Bluetooth Data</Text>

<Text style={styles.data}>{arduinoData}</Text>

<Button title="Scan for devices" onPress={scanForPeripherals} />

{allDevices.map((device) => (

<Button

key={device.id}

title={`Connect to ${device.name}`}

onPress={() => connectToDevice(device)}

/>

))}

{connectedDevice && (

<Button

title="Disconnect"

onPress={disconnectFromDevice}

/>

)}

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

justifyContent: 'center',

},

header: {

fontSize: 24,

fontWeight: 'bold',

marginBottom: 20,

},

data: {

fontSize: 18,

marginBottom: 20,

},

});

export default App;

**Development Plan**

**Phase 1: Arduino Development**

1. **Hardware Setup**:
   * Connect ADXL345 accelerometer, LCD screen, SD card module, and HC-05 Bluetooth module to the Arduino.
2. **Arduino Programming**:
   * Write and test the Arduino code to collect, display, store, and transmit accelerometer data.

**Phase 2: React Native App Development**

1. **Project Setup**:
   * Set up the Expo project and install necessary dependencies.
2. **Implement BLE Functionality**:
   * Develop the useBLE hook to handle BLE operations.
3. **Develop UI**:
   * Create UI components to display the scanned devices and received data.

**Phase 3: Integration and Testing**

1. **BLE Communication**:
   * Test BLE communication between the Arduino device and the React Native app.
2. **Data Validation**:
   * Validate the received data and ensure it matches the transmitted data.
3. **User Testing**:
   * Conduct user testing to ensure the app meets user requirements.

**Phase 4: Documentation and Deployment**

1. **Documentation**:
   * Document the code and create user guides.
2. **Deployment**:
   * Deploy the React Native app to app stores.

**Conclusion**

This software design document outlines the development process for an Arduino-based accelerometer data collection system and a React Native Expo app to display the collected data. By following this plan, we aim to create a robust and user-friendly application.