The University of Texas at Tyler

College of Engineering

Tyler, TX 75799

Final Design Report For SynkLinx – “The World’s First Smart Necklace"

A design project to fulfill the requirements of Senior Design in the Departments of Electrical and Computer Engineering at The University of Texas at Tyler

The individuals whose names and signatures appear below certify that the narrative, diagrams, figures, tables, calculations, and analyses contained within this document are their original work except as otherwise cited.

Jordan Abron	[illegible]	12/02/2025
	Signature	Date

Acknowledgements

I would like to thank the entire faculty and staff at UT Tyler Department of Electrical and Computer Engineering, specifically Dr. Frederic Rizk and Dr. Vijayalakshmi Saravanan for their advice, grace, analysis, and support during this semester and in the inception of this project. They both provided critical analysis and insight that weighed heavily on the success of the device. I also would like to thank my dear wife Kaitlyn and my parents for their financial contributions regarding the materials and project construction. Everyone mentioned above made an impact well beyond what words can convey. Without them, the SynkLinx prototype would not have been possible.

Executive Summary

Society as it stands today, is obsessed with sleek, ergonomic, subtle technological designs. Mobile phones and wearables have evolved from singular operational devices to ones that can execute multiple tasks at once, with the latter taking the form of watches, wristlets, rings, and pendants. The foundational makeup of wearable devices includes an infrared sensor to read vital signs, an accelerometer for movement tracking, a tiny rechargeable lithium-ion battery as a power supply and an occasional display for viewing notifications such as heart rate, steps, calories burned and battery life. If no display is included, the device connects to a mobile device via Bluetooth to view metrics.

With the smart device market rapidly expanding, it offers nothing in the realm of neckwear that accomplishes what other smart wearables can. Anatomically, the neck area provides a prominent pulse and core temperature readings from which to extract data. Equipping technology gives wrists and fingers freedom to outfit traditional jewelry. Necklaces are known to move freely, without restricting movement or breathing. The SynkLinx device must obtain accurate data by securing the sensors onto the necessary region, limiting movement, but still maintain the aesthetic of jewelry.

Due to this being a necklace, I will have to design a device that streams and transmits all the data to a mobile display via Bluetooth. Device real estate is of major importance considering that every component must be small enough to complement the look of a necklace. All components are designated to fit in a space of 1 x 5 inches (25mm x 127mm). Thickness is contingent on raw materials used yet must not exceed 0.5 inches (12.7 mm). The device requires a small, lithium-ion battery to support device functionality and portability.

I integrated the electrical components on the rear of the neck area, equipping it with optional interchangeable chains that attach on the front side for style purposes. The software portion displays three essential vitals (heart rate, blood oxygen saturation, core temperature) received from the hardware, for viewing. The application design is iOS based and uses React Native framework and Expo Go development servers. The

firmware uploaded into the device is derived from an Arduino IDE, then hardware and software use matching UUIDs to link both components together.

The margin of error for the respective metrics cannot exceed 5 percent. Dealing with health statistics involves the safety and well-being of the consumer and it is imperative to maintain a great level of accuracy. Health parameters vary by user, but safe ranges for vital signs are based on health institutions like American Heart Association, National Library of Medicine, and the Mayo Clinic.

Table of Contents

List of Figures

List of Tables

1. Project Description

Modern technology continues to assimilate itself into the ever-growing fitness industry in the form of “smart” bracelet/wristlets, watches, and the new wave, rings. The common features of the aforementioned devices are that they are geared towards monitoring specific features or actions of an individual’s (heart rate, steps, blood pressure, excetera.). Each device can be paired to a mobile device as well, but the most defining feature is that the devices mimic the aesthetic of traditional jewelry pieces. However, there is one facet of jewelry that has yet to make a presence within the tech/fitness world, and that is a smart… necklace.

My design project SynkLinx focuses on the same features as many of the current smart devices, but with more versatility. Regarding wearables, a point of emphasis is to seamlessly integrate the “wearable” into everyday wear inconspicuously. The consumer also wants to have the ability to view data, notifications and reply while not having an over reliance on their mobile phone. Some individuals prefer traditional jewelry (i.e., watches, bracelets) and would like something to keep track of personal health statistics, especially those with underlying medical conditions; that is my target audience. For fitness enthusiasts, like me, smartwatch and wrist wearable options seem practical until a dumbbell or weight plate accidentally crashes into it, causing mechanical and/or cosmetic damage. Anything attached to a hand or an arm always has a higher likelihood of being in harm’s way. The SynkLinx prototype is specifically designed to only monitor heart rate, blood oxygen level and body temperature of the individual wearing the device.

The concept is to place the device around the neck, in which the sensors are firmly pressed onto the skin, near the subclavian vein, carotid artery, and/or jugular vein to collect data for the given statistics. These blood vessels are all located in the upper torso/neck area and are near the heart, where body temperature and pulse readings are prominent and typically easy to obtain. Blood flow and body thermoregulation bear a direct correlation to each other (Figure 1.). Information obtained from the SynkLinx device will be collected and stored within a central development board that transmits the data via Bluetooth to a linked mobile phone application (iOS/Android), which allows the user to view the recorded information in real time. The app provides a hub for the user to not only look at data but receive notifications while wearing the device to make the user aware of connectivity issues or potential health risks. To prevent an inadvertent device reset, the plan is to reset everything directly from the app. There is also a safety breakaway feature, if the Synklinx device gets lodged or stuck in a compromising position, restricting the user's movement. To ensure minimal movement, the device will feature a material that contours around the user's neck to ensure the sensors hold position to yield accurate results for data transmission.

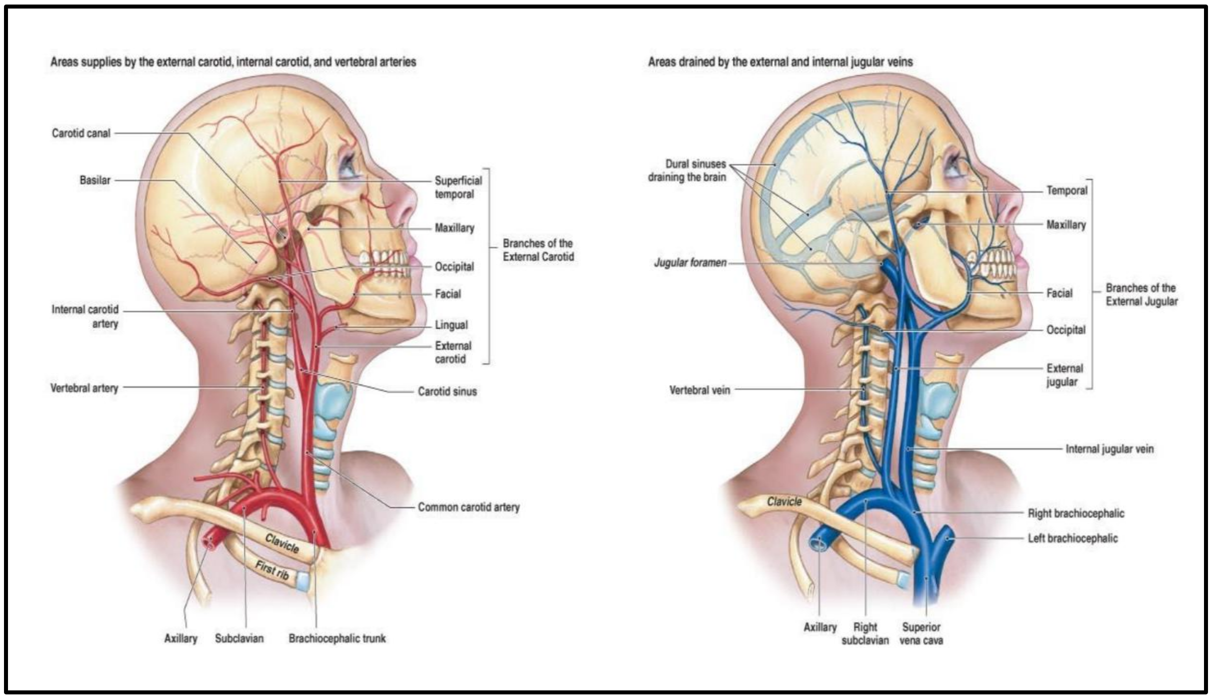


Figure 1. Diagram of Blood Vessels in Head/Neck Region

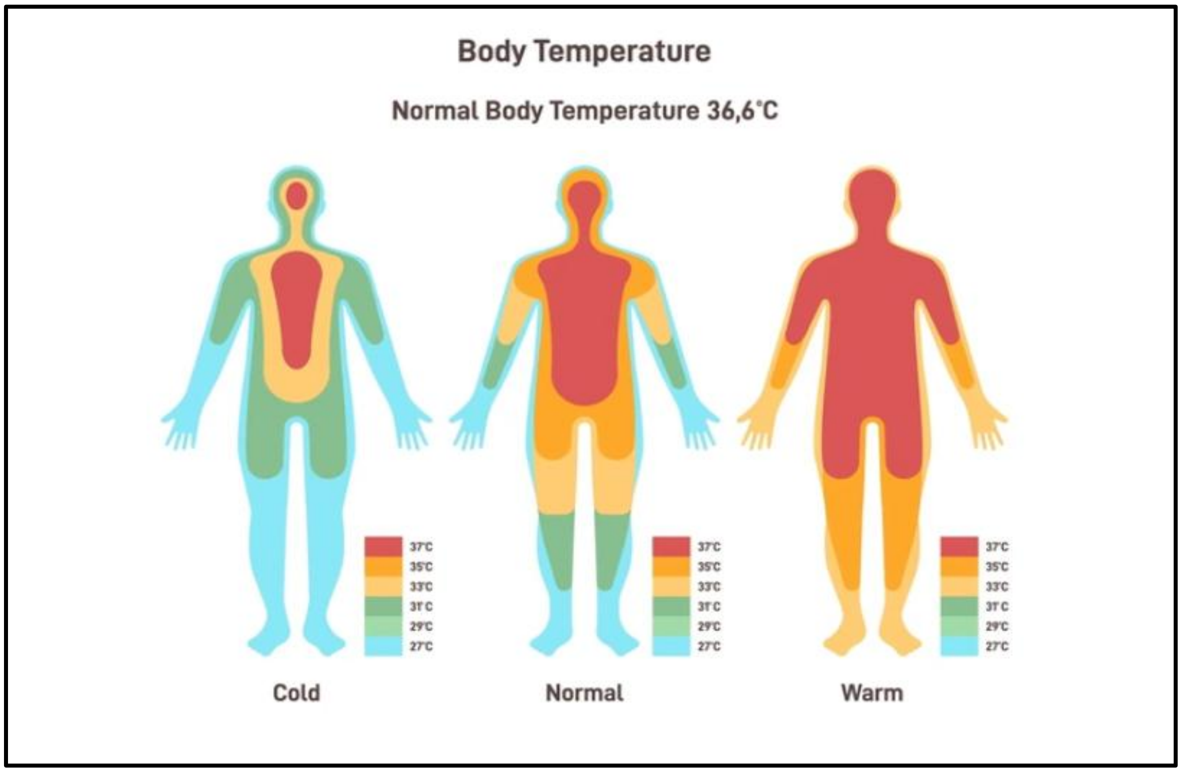


Figure 2. Body Thermoregulation Diagram

Commercially, there are minimal advances made within the realm of wearable necklaces. A few companies have invested in wearable pendants, intended to be worn with a lanyard, chain or some sort of neckwear. The purpose of each pendant varies; some are designed to share one's heartbeat with close family and friends. InvisaWear, is a pendant partnered with ADT that provides a person's GPS location and can alert loved ones and emergency services by double clicking a hidden button placed behind the charm. While personal safety is of high importance, that specific device offers limited functionality, and a pendant may not be appealing to certain consumers. Pricing for each device varies from brand to brand, starting at around $50 and can extend beyond the $1000 range, depending on materials used and supported functions.

SynkLinx will be introduced at a moderate base price point ($199.99), equipped with a lithium-ion battery USB-C charger, and the SynkLinx device with a standard length of 20 inches and an adjustable band. Eventually the product will feature interchangeable band covers to suit the consumer's personal style. The connected application will be free to download on the App Store, until the Android version is available.

2. Design Specifications

The total device weight includes a 3.7 V lithium-ion battery (1.65g), the metal contouring materials, the device shell material, the number of wires needed to connect the Pulse Oximeter and Heart Rate Sensor (MAX30102) (2.00g), and the Temperature Sensor Evaluation Board (MCP9808) (0.90g) to the Seeed Studios XIAO ESP32C3 (5.00g), the main board for the respective devices. Including the woven materials clip attachment, the estimated total weight of the SynkLinx device is around 15 ~ 25g. The ESP32 has 802.11 (Wi-Fi) and Bluetooth (BLE) connection capabilities, one of which serves as the primary transmission channel of the SynkLinx to the user's mobile device, the app location and platform of which the heart rate, oxygen saturation and the body temperature will be displayed.

The SynkLinx device itself is powered by a 3.7 V 500mAh rechargeable lithium-ion battery, which is wired directly to the battery terminals of the ESP32C3. The battery life is unknown now, but I am certain the charge dissipation depends on device usage, especially considering Bluetooth transmission. The two sensors mentioned above are connected to the XIAO ESP32C3 microcontroller, via SDA (Serial Data) and SCL (Serial Clock). SDA transmits the data and SCL synchronizes the data transfer by the address of a given chip. To prevent transmission issues, each sensor is assigned a unique address, denoted in hexadecimal (x00), sending and receiving data whenever the respective clock cycle arrives. The results are collected and pushed over air through Bluetooth (BLE) to the installed app on a mobile device. The temperature is recorded in Fahrenheit and Celsius; the heart rate and oxygen saturation are also displayed. Through the app, the user can view vitals and receive notifications if any health risks arise.

The unibody band will consist of a woven, nylon material like what you see with most of the available consumer grade smart devices. It will be a water-resistant, moisture-wicking material to protect the sensors, circuit board, and battery from any condensation. Ideally, I would like it to be a removable, washable material to eliminate any odors or stains that may occur from daily usage. There is also wire-like material that allows the user to wear and contour the SynkLinx device around the neck area, producing more accurate readings. Lastly, future markups will include customizable sleeves and if permissible, microphones and small speakers to add voice control to the device. My priorities lie in ensuring that the heart rate, oxygen saturation, and body temperature features are working optimally.

Table 1. Evaluation Criteria

Evaluation Basis	
Public Health	The device should not emit excess heat or harmful substances.
Safety/Wellness	The device should not impede the safety and wellness of the user. Product should not pose any dangers to the user (i.e., choking hazards, burn hazard, skin irritant, excetera.)
Global Impact	The device should pass regulations set in different countries and territories.
Cultural Impact	The product should not interfere or subtract from cultural norms and practices.

Social factors	The device should consider all social constructs and social customs.
Environmental factors	Device should not contribute to hazardous waste.
Economic factors	Target budget is an economic price point of $149.99.

3. Design Solution

There are two sensors essential to the SynkLinx prototype, the first being the heart rate pulse oximeter (MAX30102). The other sensor collects the body temperature (MCP9808). They relay the information the XIAO Seeed Studio ESP32C3, all of which are linked together and fastened around the neck of the user, near the carotid artery.

3.1 Product Architecture

Below is a functional block diagram of the SynkLinx (Figure 3.), divided into a pulse oximeter and heart rate subsystem, temperature subsystem, device subsystem, and the server/application subsystem.

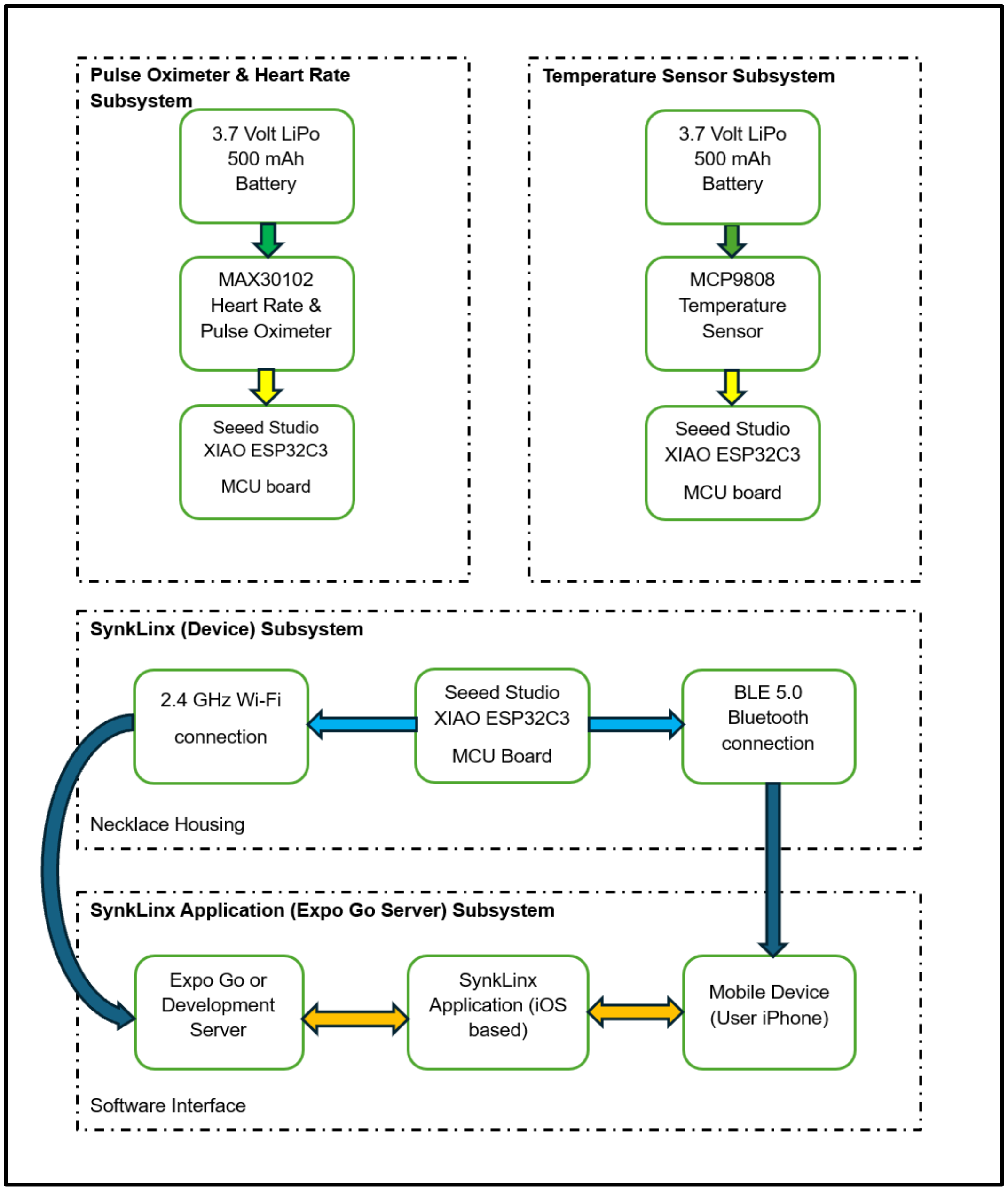

Pulse Oximeter & Heart Rate Subsystem
3.7 Volt LiPo 500 mAh Battery
MAX30102 Heart Rate & Pulse Oximeter
Seeed Studio XIAO ESP32C3 MCU board
Temperature Sensor Subsystem
3.7 Volt LiPo 500 mAh Battery
MCP9808 Temperature Sensor
Seeed Studio XIAO ESP32C3 MCU board
SynkLinx (Device) Subsystem
2.4 GHz Wi-Fi connection
Seeed Studio XIAO ESP32C3 MCU Board
BLE 5.0 Bluetooth connection
Necklace Housing
SynkLinx Application (Expo Go Server) Subsystem
Expo Go or Development Server
SynkLinx Application (iOS based)
Mobile Device (User iPhone)
Software Interface

Figure 3. System-Level Block Diagram

3.2 Hardware Subsystems

3.2.1 Pulse Oximeter and Heart Rate Sensor Subsystem

The pulse oximeter and heart rate sensor (MAX30102) is connected to the Seeed Studio XIAO ESP32C3 board that is powered by a 3.7 V lithium-ion battery (Figure 3. Figure 4. Wiring layout for the Temperature Sensor).

Listed below is a visual that details the wiring layout. The **red** wire links the input voltage to the 3.3 V pin of the ESP32. The **black** wire connects the ground pins together. The **yellow** wire links the serial clock data together for synchronizing pulses for data transmission. Lastly, the **green** wire is for serial data transmission. The wiring configuration of the MCP9808 is identical to that of MAX30102, allowing each sensor to send and receive information without mixing the signals up.

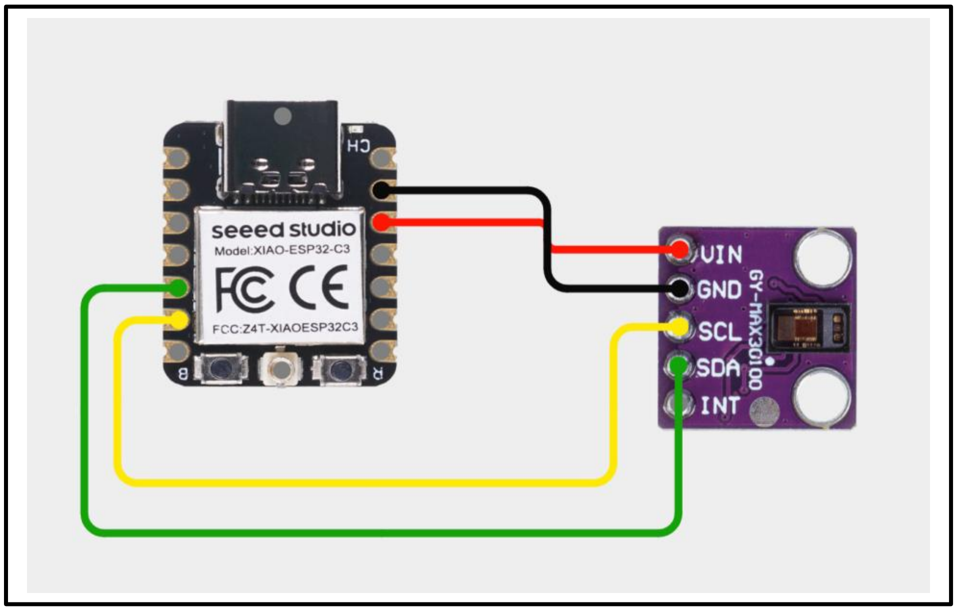


Figure 4. Wiring Layout for Heart Rate/Pulse Oximeter Sensor

3.2.2 Temperature Sensor Subsystem

The temperature sensor is connected to the Seeed Studio XIAO ESP32C3 board that is powered by a 3.7 V lithium-ion battery. Below is a visual that details the wiring layout (Figure 4.). As mentioned before, the wiring layout is identical to that of the MAX30102 and each sensor has a unique device address to data communication.

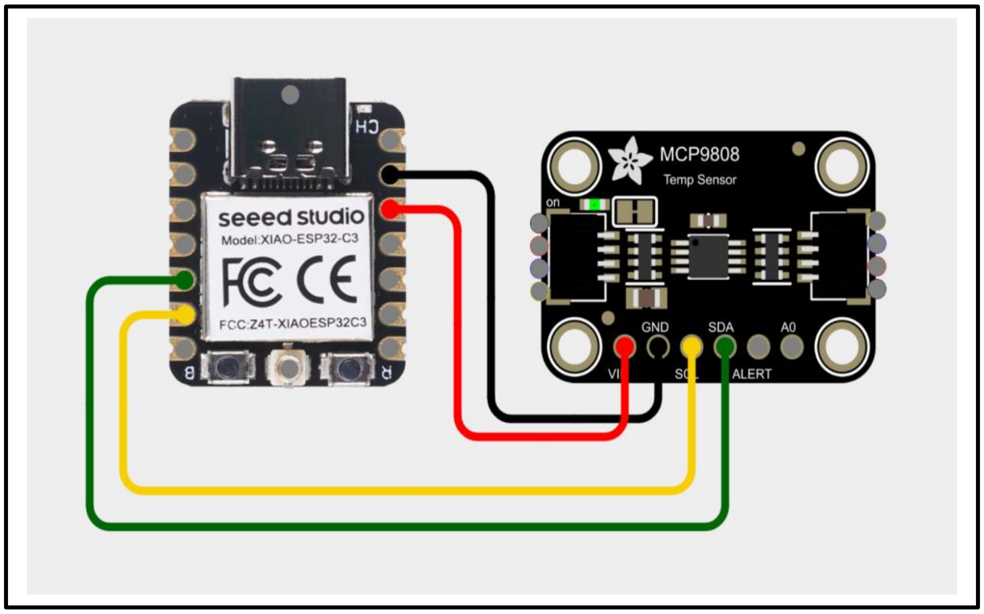


Figure 5. Wiring Layout for Temperature Sensor

3.2.3 Lithium-Ion Battery Placement

There are two terminals (+) and (-) located underneath the ESP32C3, designated for soldering batteries with compatible voltage and current. With contingencies in device usage, the battery selected to power SynkLinx should retain a charge that lasts around 4 to 5 hours.

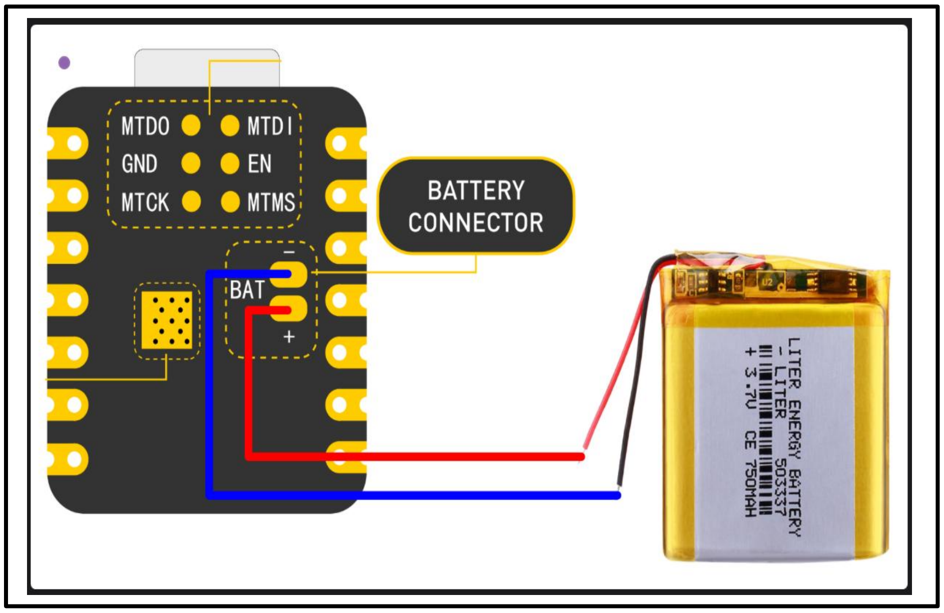
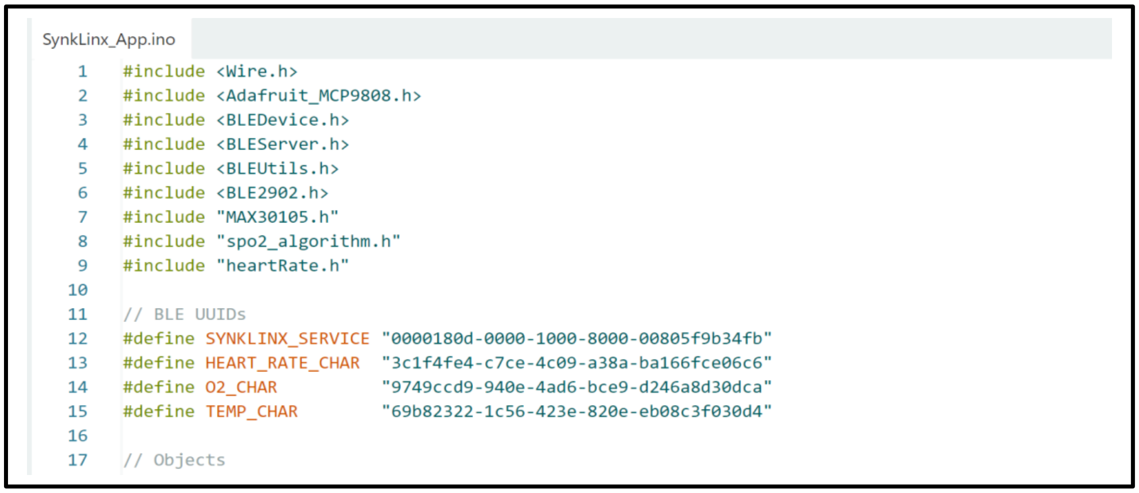


Figure 6. Lithium-ion Battery Terminal Wiring Layout

3.3 Software Subsystem

Both sensors process written code from the Arduino IDE (Figure 7.). The code is then transitioned into Microsoft Visual Studio and added to an extension named PlatformIO. Prior to transitioning, I adjusted the existing code to incorporate personalized text and notifications with respect to the SynkLinx device, confirmed the unique addresses of the respective sensors, downloaded the required libraries, and copied the UUIDs of the ESP32 into the code as defined variables so that the device will be discoverable via Bluetooth for the iPhone (Figure 8.).

Revised Arduino example code was used for the temperature, heart rate, and oxygen saturation percentage. For the SynkLinx mobile application, I used React Native programming language in Microsoft Visual Studio, added animations, text, and colors to make the app vibrant and more interactive. There were app dependencies and permissions built around BLE (Bluetooth Low Energy) that required updates and certain library versions had to be included for the software to properly execute. The service and characteristic UUIDs for Arduino and React Native need to be identical for both platforms so the hardware would be discoverable by the iOS app. I selected Expo Go, the most recommended app development builder for React Native-based applications. To receive access for building and submitting an iOS app, I had to purchase an Apple Developer's yearly subscription. There are multiple plans on the Expo website that aspiring developers can choose from, but I opted to stick with the free plan. Unfortunately, the free plan limits users to 15 iOS builds a month.

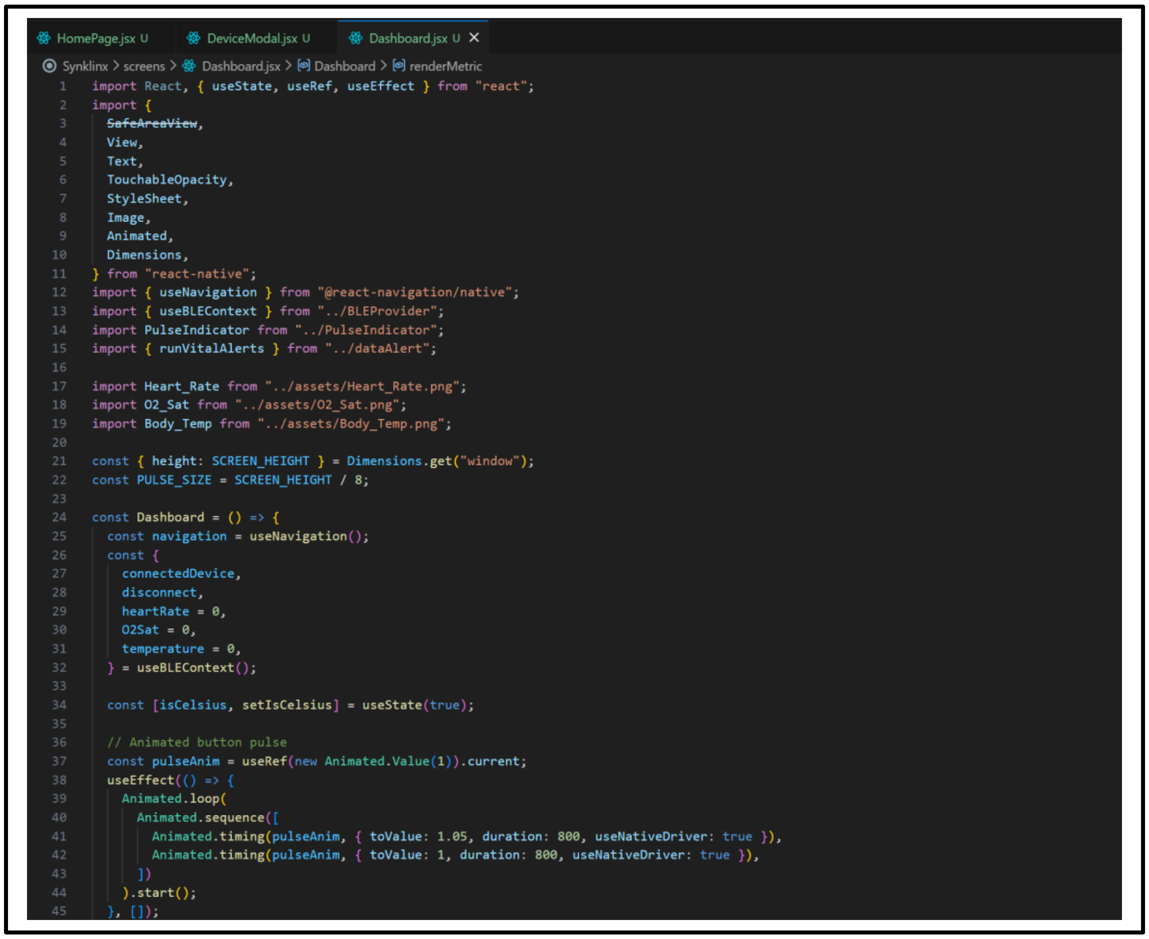


```
#include <Wire.h>
#include <Adafruit_MCP9808.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include "MAX30105.h"
#include "spo2_algorithm.h"
#include "heartRate.h"

// BLE UUIDs
#define SYNKLINX_SERVICE "0000180d-0000-1000-8000-00805f9b34fb"
#define HEART_RATE_CHAR  "3c1f4fe4-c7ce-4c09-a38a-ba166fce06c6"
#define O2_CHAR          "9749ccd9-940e-4ad6-bce9-d246a8d30dca"
#define TEMP_CHAR        "69b82322-1c56-423e-820e-eb08c3f030d4"

// Objects
```

Figure 7. Arduino IDE Code Snippet


HomePage.jsx U
DeviceModal.jsx U
Dashboard.jsx U
Synklinx > screens > Dashboard.jsx > Dashboard > renderMetric

```
import React, { useState, useRef, useEffect } from "react";
import {
  ~~SafeAreaView~~,
  View,
  Text,
  TouchableOpacity,
  StyleSheet,
  Image,
  Animated,
  Dimensions,
} from "react-native";
import { useNavigation } from "@react-navigation/native";
import { useBLEContext } from "../BLEProvider";
import PulseIndicator from "../PulseIndicator";
import { runVitalAlerts } from "../dataAlert";

import Heart_Rate from "../assets/Heart_Rate.png";
import O2_Sat from "../assets/O2_Sat.png";
import Body_Temp from "../assets/Body_Temp.png";

const { height: SCREEN_HEIGHT } = Dimensions.get("window");
const PULSE_SIZE = SCREEN_HEIGHT / 8;

const Dashboard = () => {
  const navigation = useNavigation();
  const {
    connectedDevice,
    disconnect,
    heartRate = 0,
    O2Sat = 0,
    temperature = 0,
  } = useBLEContext();

  const [isCelsius, setIsCelsius] = useState(true);

  // Animated button pulse
  const pulseAnim = useRef(new Animated.Value(1)).current;
  useEffect(() => {
    Animated.loop(
      Animated.sequence([
        Animated.timing(pulseAnim, { toValue: 1.05, duration: 800, useNativeDriver: true }),
        Animated.timing(pulseAnim, { toValue: 1, duration: 800, useNativeDriver: true }),
      ])
    ).start();
  }, []);
```

Figure 8. Microsoft Visual Studio Code Snippet

3.3.1 SynkLinx Application Prototype

There are multiple components that make up the SynkLinx application (Figure 9.). React Native is derived from the React language and is primarily used for mobile application development. Wearables and applications both share mobility as a common theme which was why I decided on an app versus a web page. The app itself is not overly complex, nor minimal. The beta process is streamlined to start, connect, and display data. Below are the essential files used to create the SynkLinx app.

- **App.jsx** – The hub file for combining Home page and Dashboard screens, most of the application.
- **HomePage.jsx** - The first screen displayed when tapping on the app. Has a start button that transitions user to the DeviceModal screen.
- **DeviceModal.jsx** – The file where device scanning and connectivity takes place.
- **Dashboard.jsx** – The file for the data metrics. Imports the other needed files and is responsible for handling the animations and the visuals.
- **dataAlert.jsx** – This file returns notifications from the three vitals for connectivity issues, warnings, and danger if vitals deviate from medically approved statistics.
- **PulseIndicator.jsx** – File responsible for pulse animation behind the heart rate, oxygen saturation, and body temperature.
- **useBLE.jsx** – File that links the service and characteristic data from the physical device to the mobile application and streams real time data on the app.

- **BLEProvider.jsx** – File that allows Bluetooth service to be used across multiple screens without overlap in the application interface.
- **Firmware.cpp** – Contains the firmware from Arduino IDE, uploaded into SynkLinx device to allow connection and data streaming.

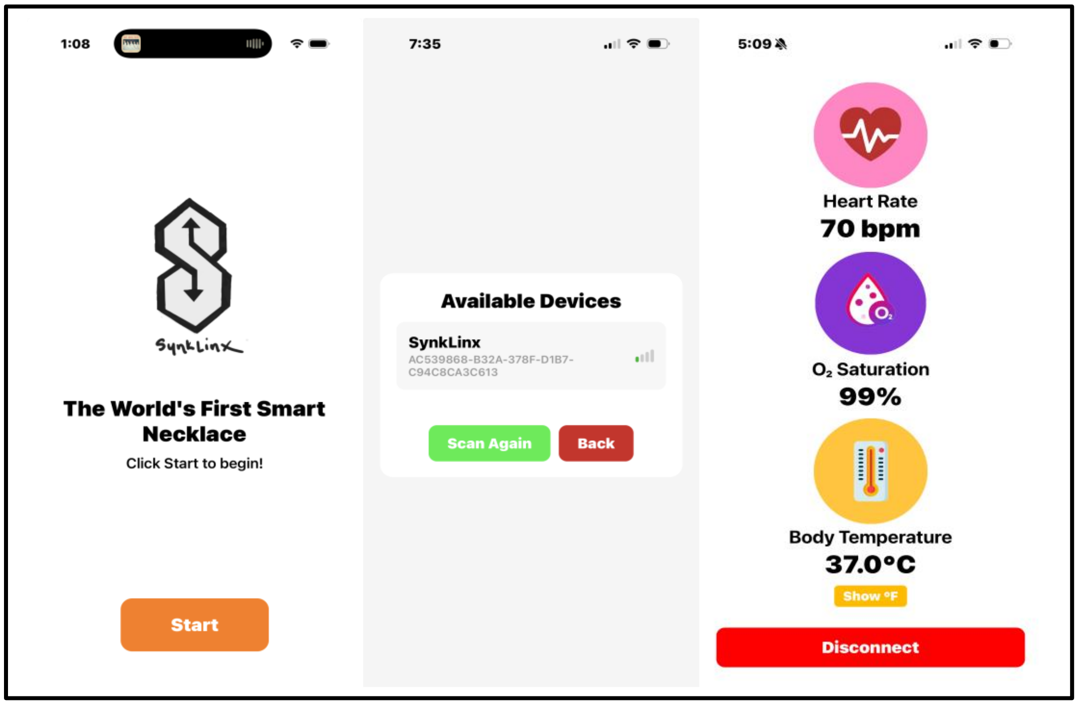


Figure 9. Snapshots of HomePage, Device Modal and Dashboard screens

The metrics presented in the app snapshot are mock generated, for testing purposes. When troubleshooting and debugging; the allotted 15 builds per month from Expo Application services are crucial to preserve. Early trial runs yielded solid results. The flowchart shown below outlines the general operating procedure for how the device and application interact.

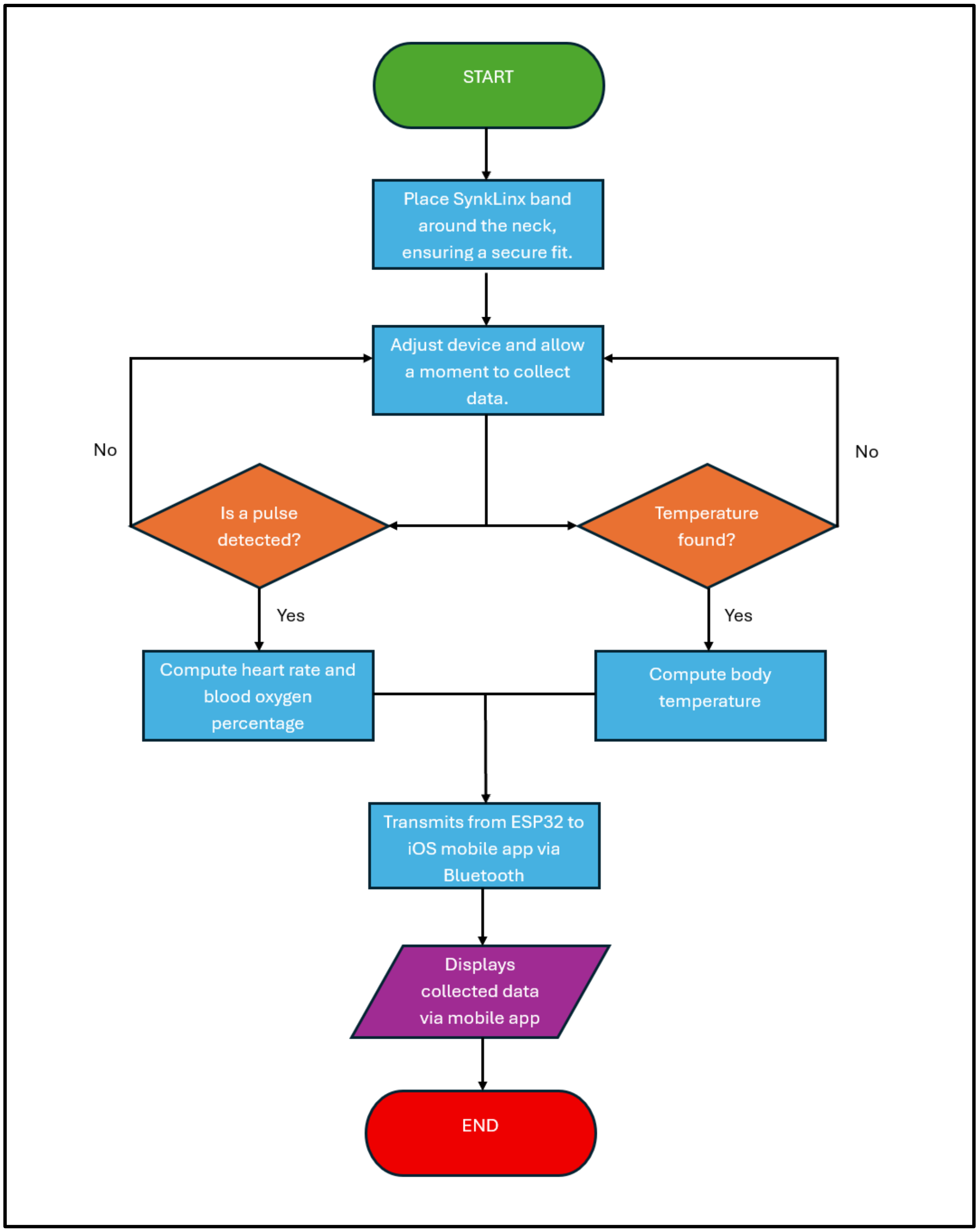

START
Place SynkLinx band around the neck, ensuring a secure fit.
Adjust device and allow a moment to collect data.
No
Is a pulse detected?
Temperature found?
No
Yes
Yes
Compute heart rate and blood oxygen percentage
Compute body temperature
Transmits from ESP32 to iOS mobile app via Bluetooth
Displays collected data via mobile app
END

Figure 10. Flowchart for SynkLinx Prototype and App Procedure

3.4 Hardware Components

Hardware components essential to device construction are listed by product name, manufacturer, and part number. Jumper wires, soldering equipment, and other raw materials used to complete the necklace aesthetic will not be included in the table below.

Table 2. Hardware Utilized in Building SynkLinx device, Linked to Datasheets.

Product Name	Manufacturer	Part #
Seeed Studios XIAO ESP32-C3 Dev Board	Seeed Technology Co., Ltd	1597-113991054-ND
MAX30102 Heart Rate and Pulse Oximeter	SunFounder	4411-ST0244-ND
MCP9808 Temperature Sensor Breakout board	Adafruit Industries LLC	1528-1032-ND
3.7 V 500mAh Rechargeable Lithium-ion Polymer Battery	YDL	702030

3.5 Software Components

The software needed to program and connect the device to its application are Arduino IDE, Microsoft Visual Studio Code and Expo Go, respectively. Information for all software is listed below. Full code for firmware and each application component can be found in Appendix H.

Table 3. Software Implemented to Develop SynkLinx Mobile Application

Name	URL	Software Version	Purpose
Arduino IDE	https://www.arduino.cc/en/software/	Arduino IDE 2.3.6	Program used to code hardware devices
Microsoft Visual Studio Code	https://code.visualstudio.com/download	1.105.1	Interface that combines code from hardware to application
Expo Development	https://expo.dev/signup	54.0.1	Development build for mobile application
GitHub	https://github.com/friyiajr/BLESampleExpo/tree/main	N/A	Reference for application design
Cirkit Designer	https://app.cirkitdesigner.com/	N/A	Reference for sensor wiring layout

3.6 Schematics/Circuit Diagrams

The wiring diagrams for the Seeed Studio ESP32-C3 dev board, MAX30102 Heart Rate Pulse Oximeter and MCP9808 Temperature I2C Sensor Breakout board are displayed in the figures below, respectively.

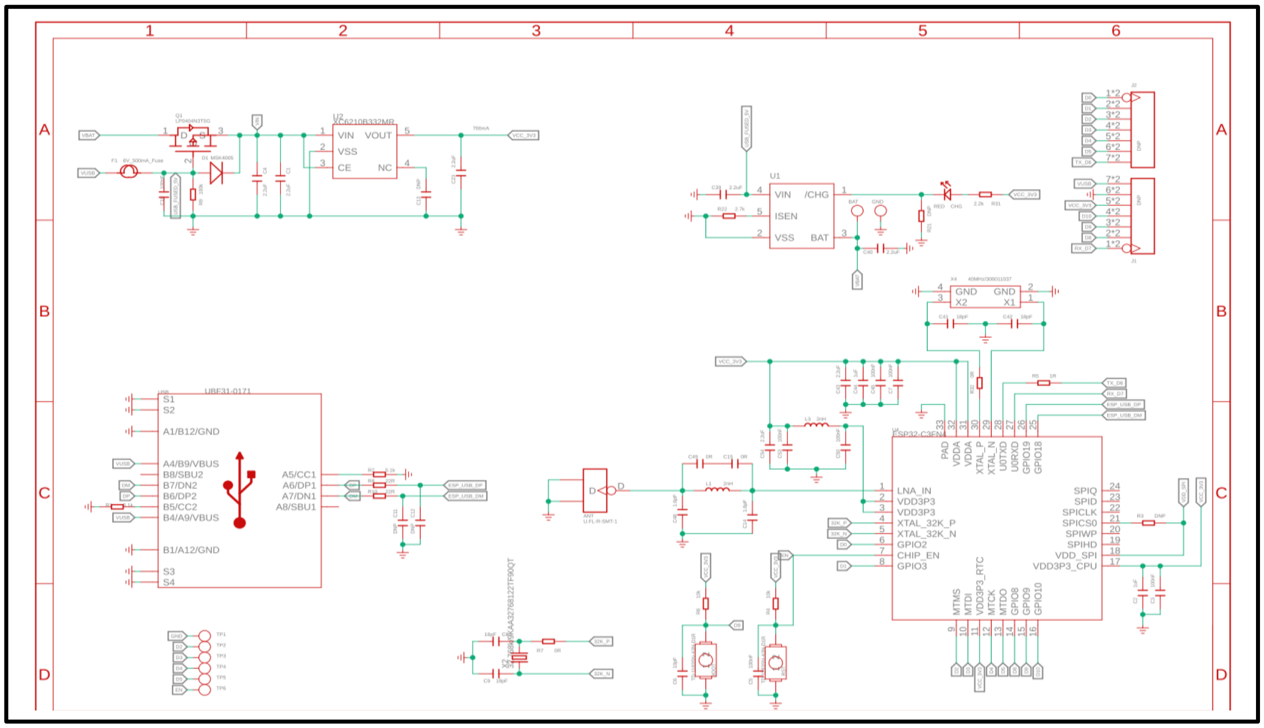


Figure 11. Circuit Diagram for Seeed Studio ESP32-C3 Dev Board.

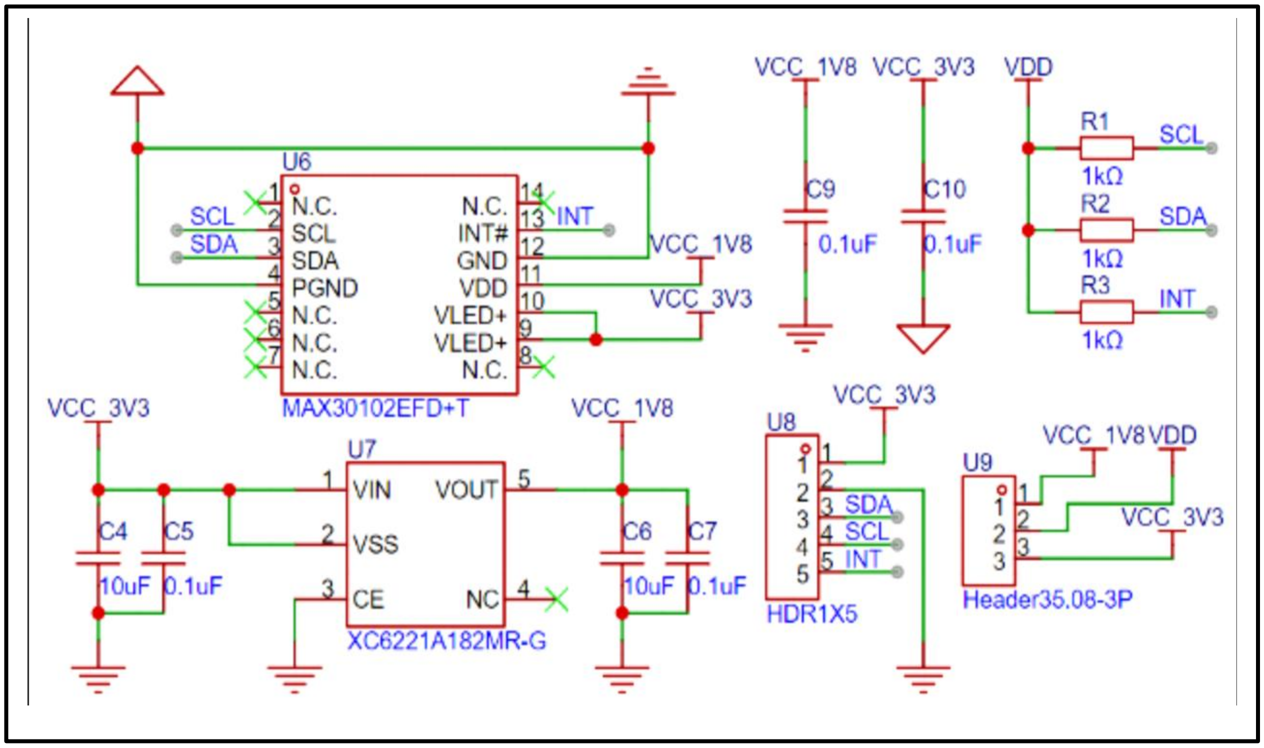


Figure 12. Circuit Diagram for MAX30102 Heart Rate Pulse Oximeter.

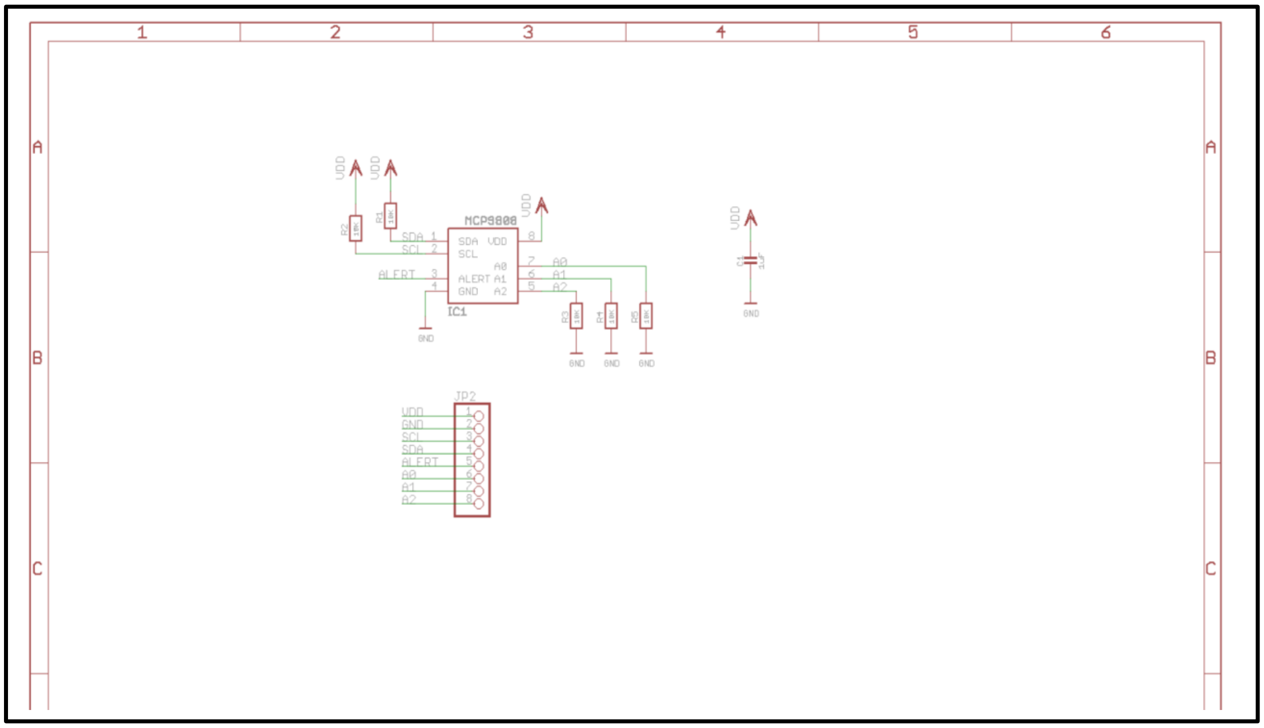


Figure 13. Circuit Diagram for MCP9808 Temperature I2C Sensor Breakout Board.

3.7 Custom Software

The custom Arduino software regarding the nature of the project will be listed in the Appendix H for reference, as will the React Native files that I briefly detailed earlier. While constructing the app, I employed YouTube as a source, which led me to explore the GitHub page https://github.com/friyiajr/BLESampleExpo/tree/main . To avoid plagiarism, I modified file layouts, animations, colors, and imported images. The foundation of the file is strongly referenced in that of my own code. Other than visual differences between the GitHub directory and mine, I opted to separate the screens by creating a “HomePage.jsx” , “DeviceModal.jsx” and “Dashboard.jsx” file instead of a single ‘App.jsx’ file. It was helpful when troubleshooting, and debugging errors, and provided flexibility to improve the main screen before connecting SynkLinx to the app interface.

4. Prototype Design and Fabrication

The SynkLinx prototype was designed and developed using a standard breadboard. The sensors and microcontrollers are powered using the breadboard via USB-C power supply. The prototype was tested using an index finger pulse of adult test subjects. The MAX30102 was pre-soldered and worked seamlessly in the Arduino IDE. I used a larger SunFounder ESP32 for initial testing but managed to solder duPont pins to the MCP9808 and the Seeed Studio ESP32-C3 boards.

During prototype testing, Each ESP32 boards were connected to a Windows PC for power supply and code uploading, but the final prototype will be fully operable on the 3.7 V lithium-ion battery. The development server links to a user's mobile device by Wi-Fi, but the working hardware prototype when complete will use Bluetooth to link to the app. The coding and development were completed using Microsoft Visual Studio code, Arduino IDE and Expo Go Development Server.

The approximate time to complete the first breadboard prototype totaled around 3 hours. During final prototype construction, the development process period took hours to achieve considerable progress; between building submissions and debugging alone, I would estimate ~12 -15 hours total. Below is a table covering time allocation by category (Table 4.).

All the materials needed to construct the prototype will be listed below in the Materials table and time breakdown of prototype completion will be in Appendix B. The Hardware Materials table includes the Product Name, Manufacturer, Product ID, Quantity, and Price of each component (Table 5.)

Table 4. Time Breakdown of Design Project

Prototype Construction	
Operation	**Duration (hh:mm)**
Coding (Board, App Interface)	14:45
Wiring	02:30
Soldering	01:15
Design (Software, Hardware)	11:30
Total time:	30:00

Table 5. Bill of Materials

Product Name	**Manufacturer**	**Quantity**	**Price**
MAX30102 HEART RATE MODULE	SunFounder	1	$14.67

MCP9808 TEMP I2C BREAKOUT BRD	Adafruit Industries LLC	1	$4.95
XIAO ESP32-C3	Seeed Technology Co., Ltd	1	$4.95
3.7 V 500mAh Rechargeable Lithium-ion Polymer Battery	YDL	1	$7.49
		Total:	$32.10

Beginning the prototype build process consists of inserting the pins of the ESP32 into the breadboard. Next, plug the ESP32 into your computer's USB-C port to give the primary board power. Then, wire the heart rate sensor (MAX30102) and the temperature sensor (MCP9808) to 3.3 V (voltage in), (ground), SDA (serial data), and SCL (serial clock) using the required number of pins.

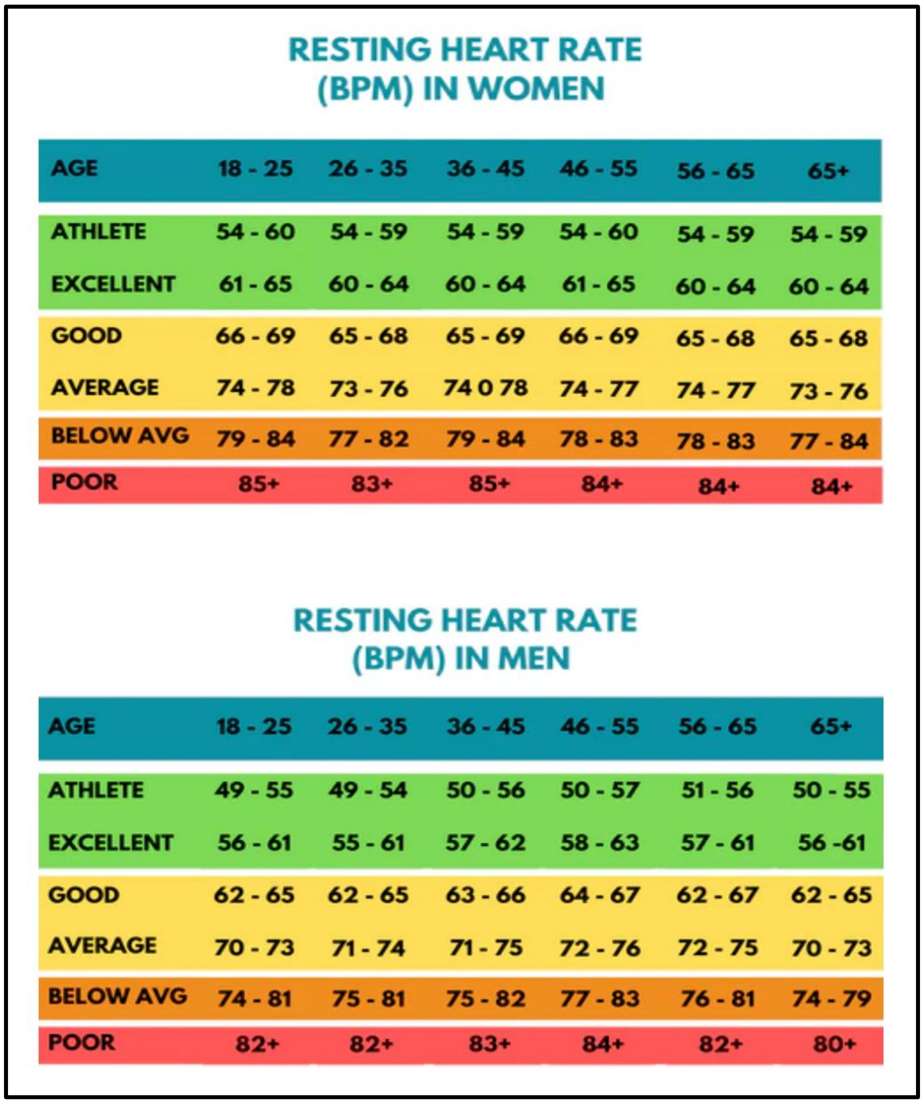
After the sensors are wired and powered, open the Arduino IDE application and upload the required code for device functionality. Test the circuit boards being used to test code by uploading a sample code, such as a blink code, to the boards. Once all circuit boards have been successfully tested, download the "Adafruit MCP9808 Library" and the "SparkFun MAX3010x Pulse and Proximity Sensor Library" from the Arduino IDE program which is used to run the temperature and heart rate sensor.

With the wiring already in place, verify the example code to ensure there are no syntax, semantic or runtime errors. Upload the code to the ESP32 board and view the Serial Monitor window to check results. If results are correct, save the code and upload the ".ino" file into Visual Studio Code (VS Code). For the Arduino code to work within the domain of the React Native code for the app, download the PlatformIO extension and download the respective libraires once again.

While in VS Code, you will also have to download a React Native extension for the SynkLinx application programming. I followed the file structure of a video from YouTube (www.youtube.com/@DanRNLab), where the user posted their GitHub link in the description to download the necessary files. I modified the code to display temperature, heart rate, and oxygen saturation as well as separated the app display screens for easier code modification. I also incorporated my own logo on the opening page, and app icon. This is an early app prototype, so I expect it to be a bit more user-friendly in the final stages. After the React Native code has been added, place the same UUID numbers from the Arduino Code into the React Native so that the mobile device can connect to the BLE (Bluetooth Low Energy) SynkLinx device. The user will be able to view the app either using the Expo Go app, or a personal development build. The former is contingent whether you have a Mac or PC, or an Android or iPhone.

When viewing the app, the user will be prompted by a start button, that takes you to the connection screen. With the UUIDs entered steps before, the SynkLinx device will be discoverable, and the user can scan and connect the hardware to the app. For testing purposes, place a finger on both the temperature and heart rate sensors. Wait for about 5-10 seconds for the device to collect and stream data to the app. Users have the option of viewing body temperature in Fahrenheit or Celsius.

Although I implemented warning and danger notifications for the designated vitals, it is best to consult a medical professional or verified medical institutional website to obtain the proper health information. Since I am my own test subject, I know the recommended range for my heart rate. The remaining two vitals (oxygen saturation, body temperature) can vary by age, gender, and activity levels. Most of the information I collected on health metrics can be found at the American Heart Association, Nation Library of Medicine, and the Mayo Clinic's website. You can also consult a local physician. Blood oxygen percentages 95 and above are considered normal; if less than 95 percent, you may be experiencing hypoxemia, and should seek medical attention immediately. Normal core temperatures for adults range from 97 F (36.1 C) to 99 F (37.2 C). If an individual's core temperature is lower than the aforementioned range, you may have hypothermia; any temperature above said range is common referred to as fever. Seek a physician’s care if unable to regulate core temperature back to normal.



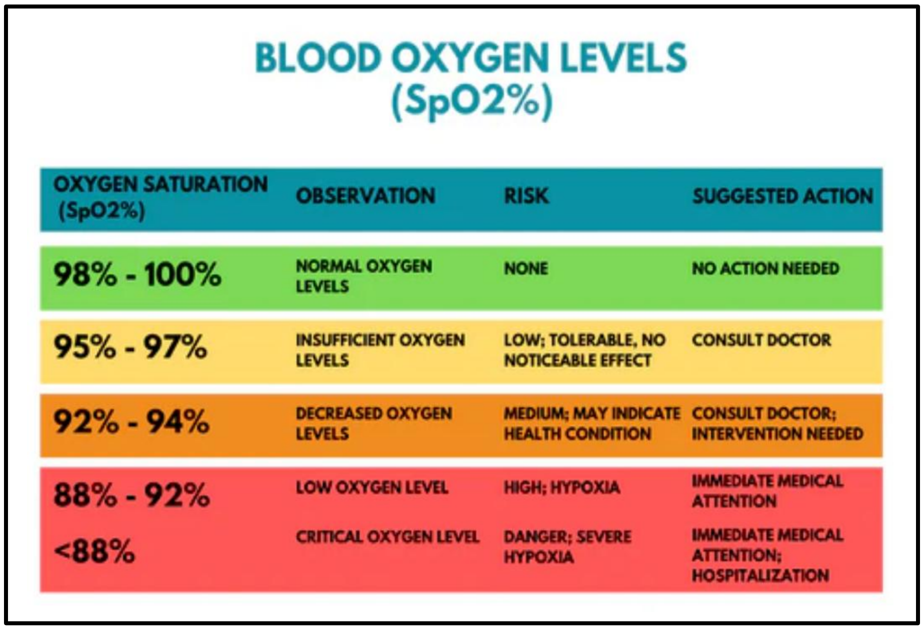
RESTING HEART RATE (BPM) IN WOMEN

AGE	18 - 25	26 - 35	36 - 45	46 - 55	56 - 65	65+
ATHLETE	54 - 60	54 - 59	54 - 59	54 - 60	54 - 59	54 - 59
EXCELLENT	61 - 65	60 - 64	60 - 64	61 - 65	60 - 64	60 - 64
GOOD	66 - 69	65 - 68	65 - 69	66 - 69	65 - 68	65 - 68
AVERAGE	74 - 78	73 - 76	74 0 78	74 - 77	74 - 77	73 - 76
BELOW AVG	79 - 84	77 - 82	79 - 84	78 - 83	78 - 83	77 - 84
POOR	85+	83+	85+	84+	84+	84+

RESTING HEART RATE (BPM) IN MEN

AGE	18 - 25	26 - 35	36 - 45	46 - 55	56 - 65	65+
ATHLETE	49 - 55	49 - 54	50 - 56	50 - 57	51 - 56	50 - 55
EXCELLENT	56 - 61	55 - 61	57 - 62	58 - 63	57 - 61	56 -61
GOOD	62 - 65	62 - 65	63 - 66	64 - 67	62 - 67	62 - 65
AVERAGE	70 - 73	71 - 74	71 - 75	72 - 76	72 - 75	70 - 73
BELOW AVG	74 - 81	75 - 81	75 - 82	77 - 83	76 - 81	74 - 79
POOR	82+	82+	83+	84+	82+	80+

Figure 14. Resting Heart Rate Chart



BLOOD OXYGEN LEVELS (SpO2%)

OXYGEN SATURATION (SpO2%)	OBSERVATION	RISK	SUGGESTED ACTION
98% - 100%	NORMAL OXYGEN LEVELS	NONE	NO ACTION NEEDED
95% - 97%	INSUFFICIENT OXYGEN LEVELS	LOW; TOLERABLE, NO NOTICEABLE EFFECT	CONSULT DOCTOR
92% - 94%	DECREASED OXYGEN LEVELS	MEDIUM; MAY INDICATE HEALTH CONDITION	CONSULT DOCTOR; INTERVENTION NEEDED
88% - 92%	LOW OXYGEN LEVEL	HIGH; HYPOXIA	IMMEDIATE MEDICAL ATTENTION
<88%	CRITICAL OXYGEN LEVEL	DANGER; SEVERE HYPOXIA	IMMEDIATE MEDICAL ATTENTION; HOSPITALIZATION

Figure 15. Blood Oxygen Chart

5. Testing and Validation

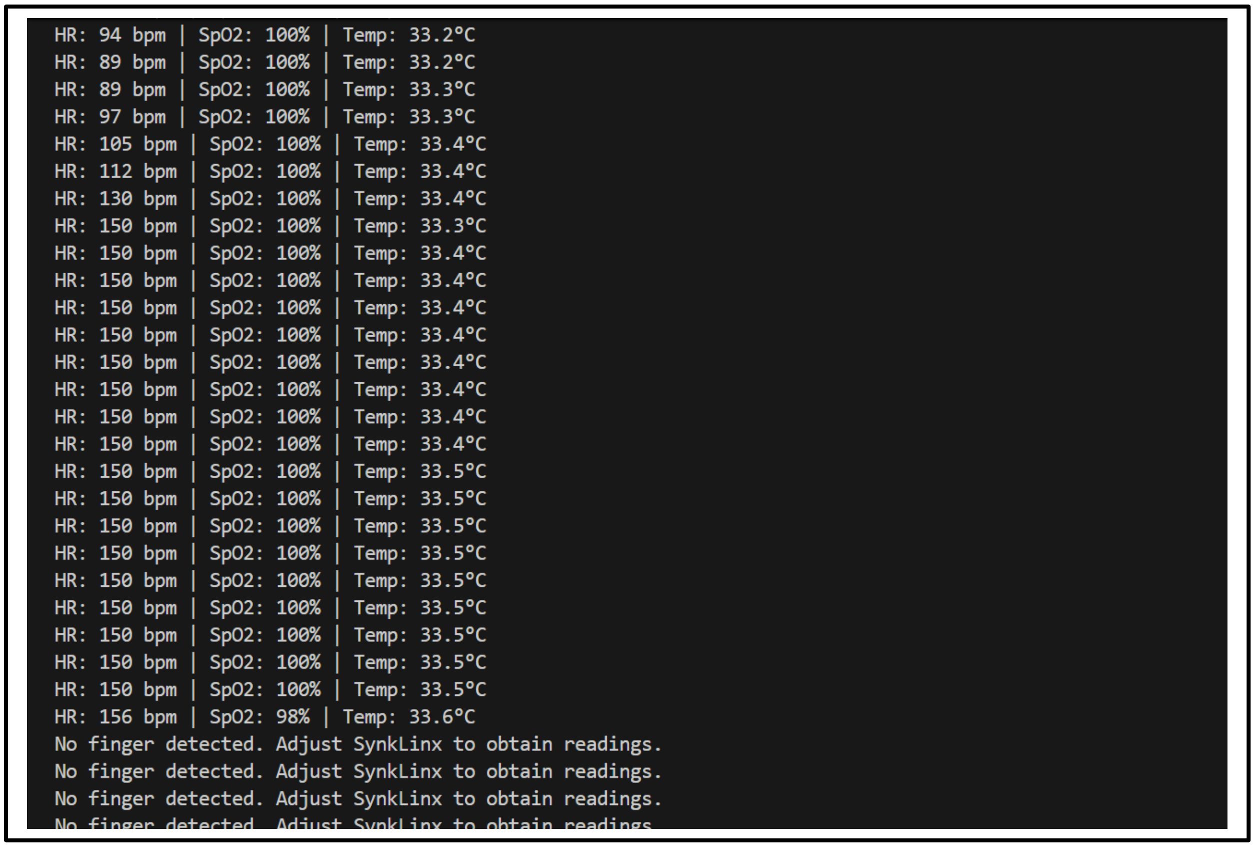
During the testing and validation phase of the project, I outlined a simple, five step process of how SynkLinx prototype works in daily activity. I used information from certified and reputable health institutions like the American Heart Association, National Library of Medicine, and the Mayo Clinic to determine the warning and danger alerts for the three vital signs (HR, SpO_2, temperature) collected during usage.

5.1 Prototype Testing:

1) Check that the SynkLinx device has sufficient charge, then power the device up.
2) Attach and adjust the prototype to comfortably fit around the test subject's neck.
3) Open your iPhone and select the SynkLinx iOS app, press start, then scan for available devices (SynkLinx).

4) Once connected, give the device about 5-10 seconds to process the subject’s heart rate, oxygen saturation, and body temperature.
5) While monitoring the application, results should show up in real-time, along with warnings and danger alerts regarding heart rate, blood oxygen percentage, and core temperature.

During the testing phase, the ESP32-C3 main board and two MAX30102 and MCP9808 sensors were soldered together using jumper wires and fastened around the test subjects’ neck to resemble the feel of a necklace. Once data was collected and streamed to the iOS app, I recorded the results of each metric in the serial monitor window of VS code and compared the SynkLinx data to readings from an existing pulse oximeter and non-contact thermometer. Links for both devices will be linked to figures


```
HR: 94 bpm | SpO2: 100% | Temp: 33.2°C
HR: 89 bpm | SpO2: 100% | Temp: 33.2°C
HR: 89 bpm | SpO2: 100% | Temp: 33.3°C
HR: 97 bpm | SpO2: 100% | Temp: 33.3°C
HR: 105 bpm | SpO2: 100% | Temp: 33.4°C
HR: 112 bpm | SpO2: 100% | Temp: 33.4°C
HR: 130 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.3°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.4°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 150 bpm | SpO2: 100% | Temp: 33.5°C
HR: 156 bpm | SpO2: 98% | Temp: 33.6°C
No finger detected. Adjust SynkLinx to obtain readings.
No finger detected. Adjust SynkLinx to obtain readings.
No finger detected. Adjust SynkLinx to obtain readings.
No finger detected. Adjust SynkLinx to obtain readings.
```
Figure 16. Serial Monitor Data Window from Microsoft VS Code

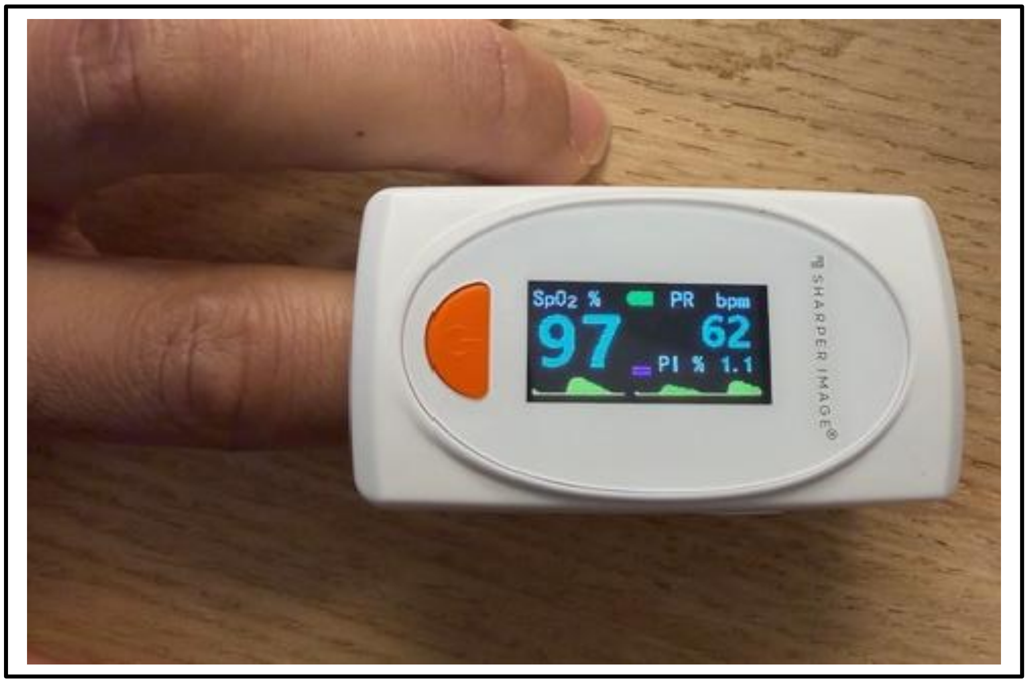


Figure 17. Heart Rate and Blood Oxygen percentage for a Sharper Image Pulse Oximeter

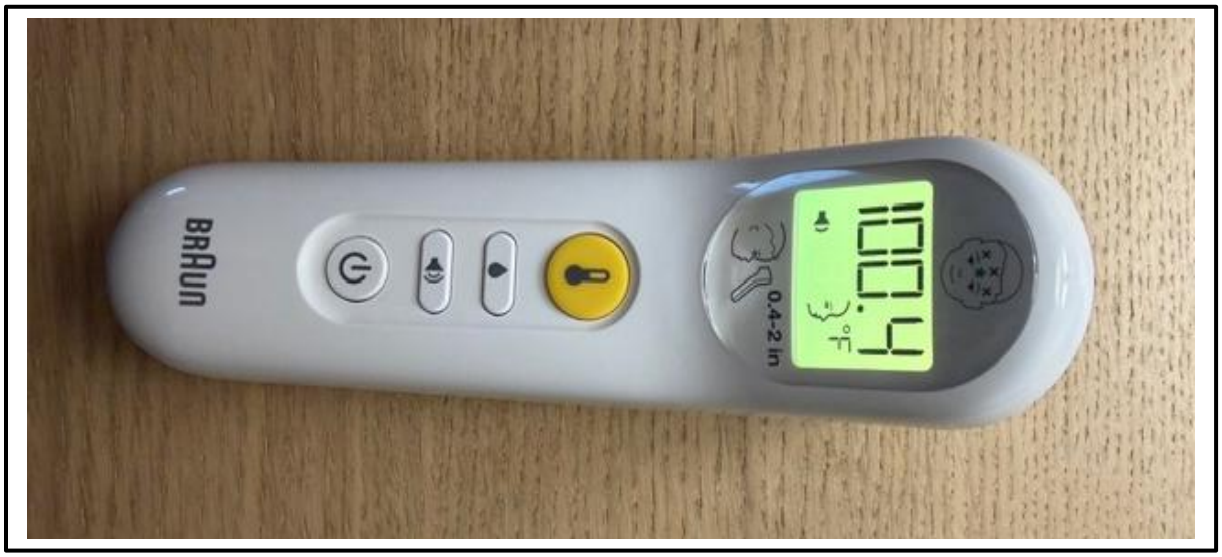


Figure 18. Core temperature from a Braun Non-contact Thermometer

The approach for validating the designated SynkLinx metrics was to allow each device to collect information for 30 seconds, record the number in one second intervals, calculate the accuracy of my device at each interval, then determine the average per second based on the percent error formula. The plan to reduce any abnormalities in data is to incorporate smoothing functions, which adds safe vital readings (65 bpm, 100% SpO_2, 37.0 Celsius) over several samples to normalize the readings. I also set ranges for each vital sign that filters out extreme, unrealistic outliers. The objective to improve measurement accuracy is to reduce percent error for metrics to less than 5% by maintaining a secure fit around the subject's neck, minimizing light exposure and movement. Since neck skin has a greater thickness than the wrists and fingers, I adjusted the IR sensor to successfully penetrate skin layers to reach the carotid artery. After error reduction, I will proceed to place the device in its housing and protect the respective sensors to prevent any damage or incorrect data streaming.

$$\%\ \text{error} = \left| \frac{\#\,\text{experimental} - \#\,\text{actual}}{\#\,\text{actual}} \right| \times 100$$

The testing results are still volatile with slightly lower-than average core temperatures and higher resting heart rates. Initial testing was done with the outside of device housing, where ambient lighting, airflow and sensor exposure play a factor in skewed readings. Reasons include light and air exposure to sensors; both are intended to apply directly to skin. Skin types also play a factor. Oily skin tends to produce a film over sensors, slightly altering true measurement. Sampling rates that are set too high could also play a key role in variances.

Table 5. Percent Error Chart

	Experimental	**Actual**	**Percent Error**
Heart Rate (bpm)	57.67	61.40	6.07%
Core Temperature F(C)	99.32(37.40)	97.22(36.23)	2.16%
Blood Oxygen (%)	96.20	97.22	1.05%

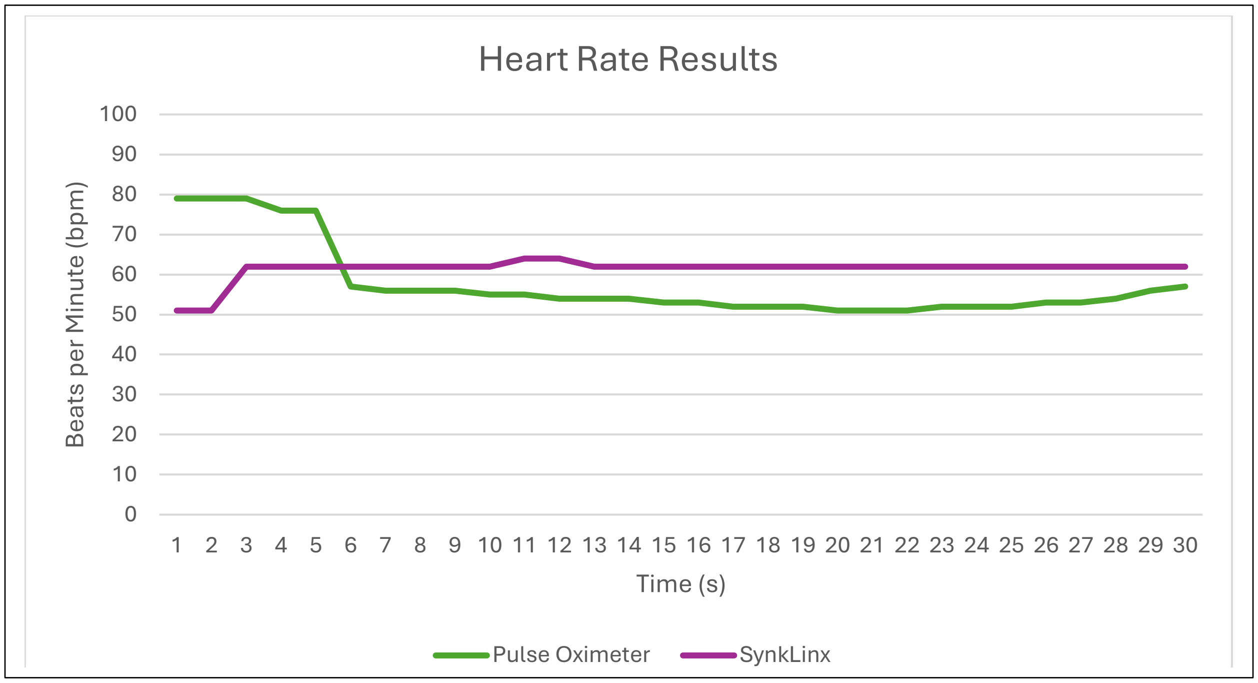


Figure 19. Graph of Heart Rate Validation

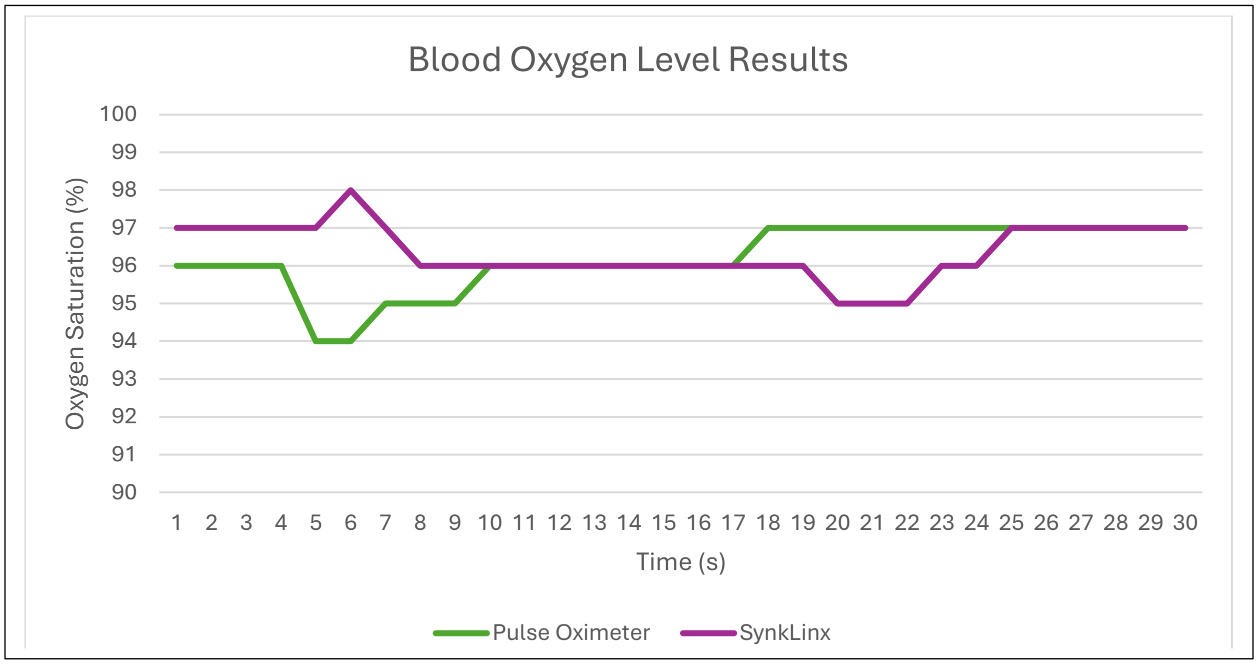


Figure 20. Graph of Blood Oxygen Level Validation

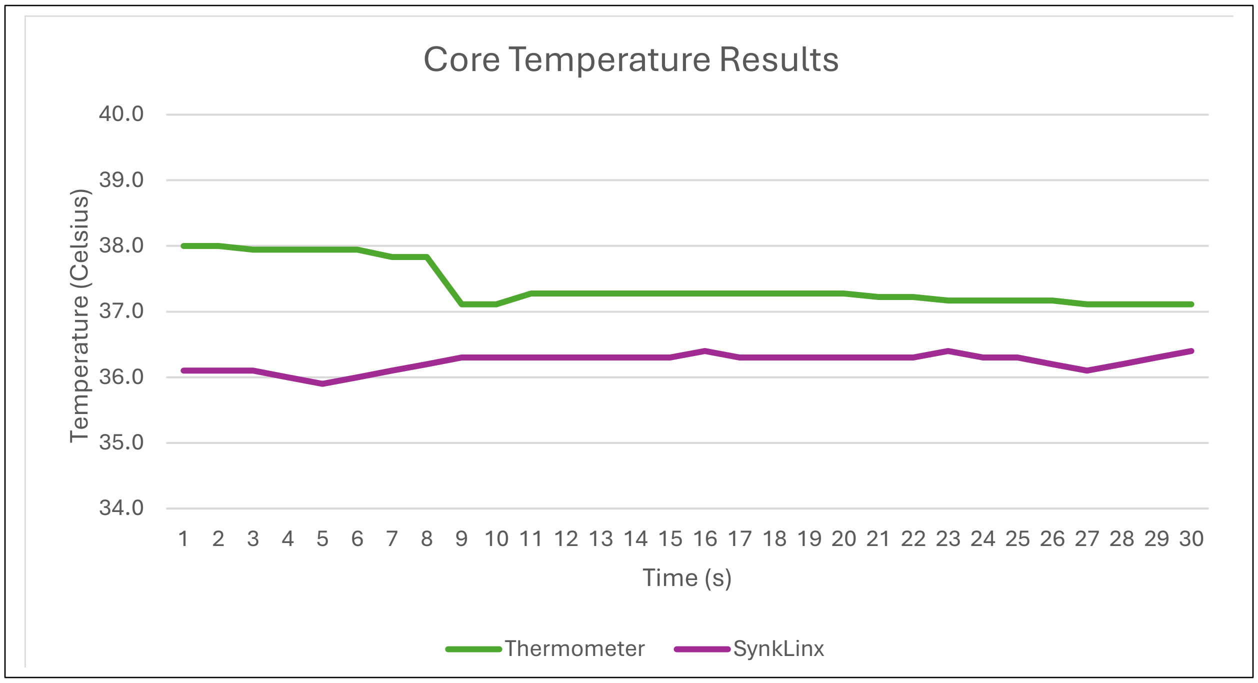


Figure 21. Graph of Core Temperature Validation

After installing the hardware into its shell, I added a vital sign filtering function to the firmware and performed 5 more test runs on the device/application. Readings yielded realistic results in a timely fashion. On the fifth and final test run, I recorded the SynkLinx data and compared it to the measurements taken by the non-contact thermometer and pulse oximeter. In Appendix H is the table listing the numbers from each respective device during testing.

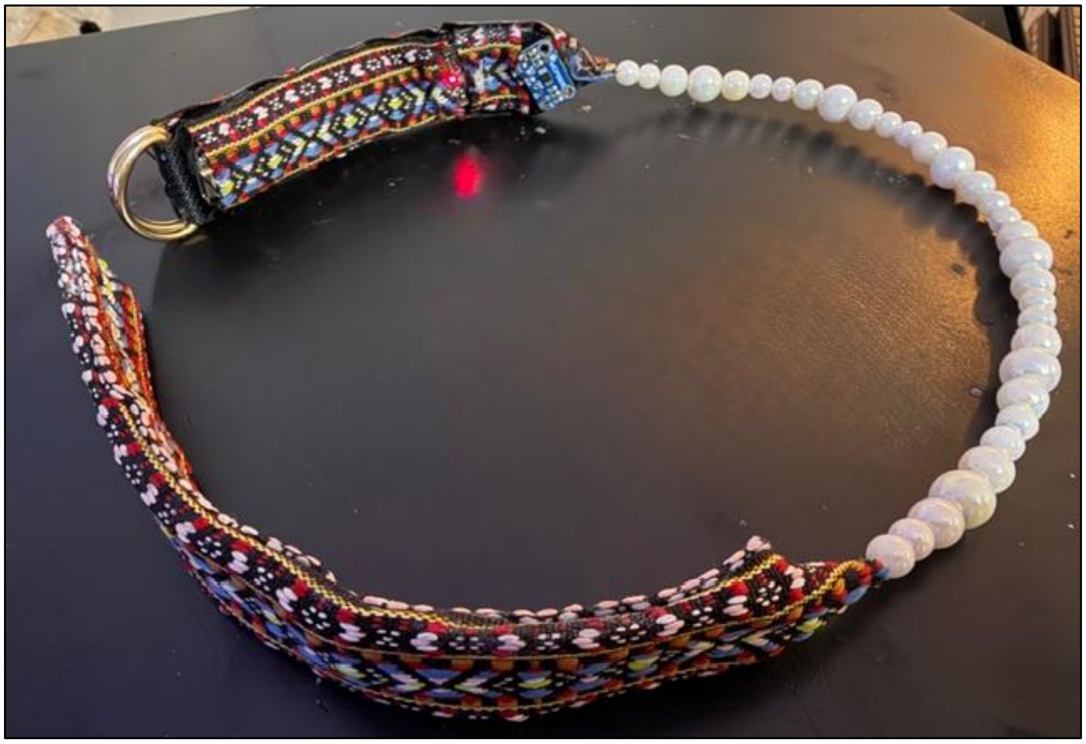


Figure 22. Final SynkLinx Necklace Prototype

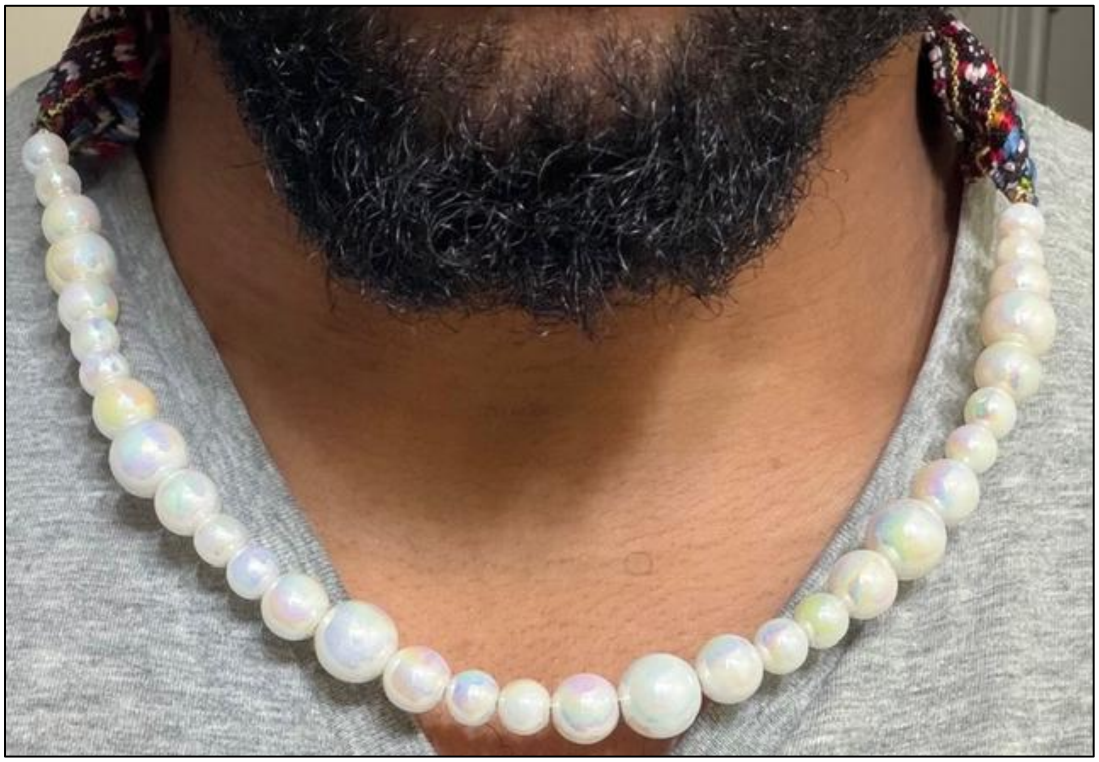


Figure 23. Final SynkLinx Necklace Prototype (in-use)

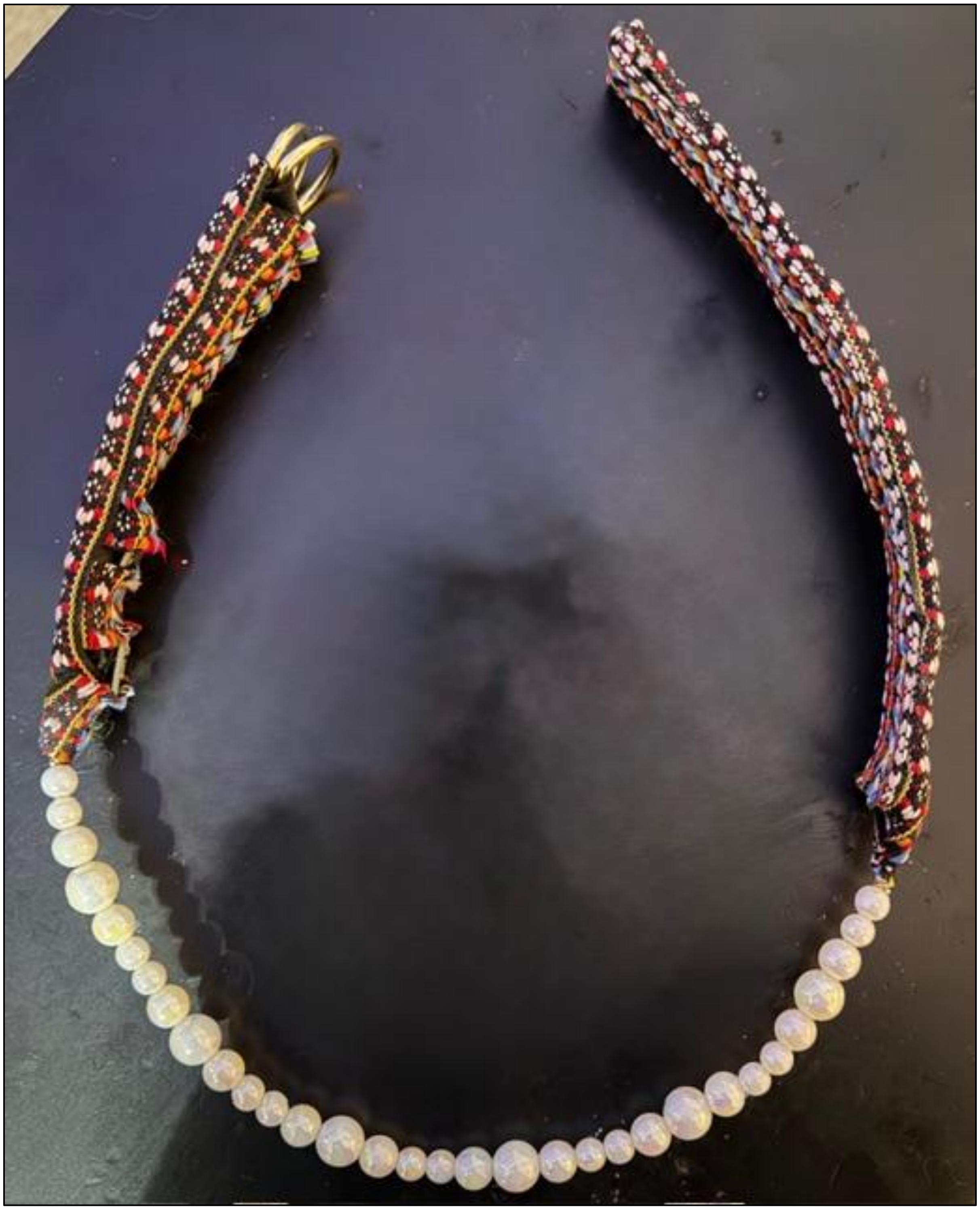


Figure 24. Final SynkLinx Necklace Prototype (top-view)

6. Broader Impacts of the Project

During the early stages of design and development, my objective was to create an alternative wearable device option for consumers that do not own a smart watch, smart ring, or desire to purchase one, but would like a device that allows them to monitor basic yet important core vital parameters. Fitness, health, and versatility were the central themes of project development. My initial target audience were adults aged 18-45; however, safe vital signs for test subjects or consumers vary with respect to human anatomy.

Concerning the impact of the SynkLinx device, it is unknown. However, I am confident that at the pace technological advancements move there will be a robust market for smart necklaces soon, just from a fashion perspective alone. The current prototype has limited functionality, but future iterations will offer more features than just health monitoring. The market for ergonomic, multipurposed technology, especially for active individuals, grows every year. For individuals who may be in bad health but may be unaware of the extent, SynkLinx app alerts will be convincing enough to prompt one to seek medical attention. My belief is that after more research, development and testing is conducted on this product or comparable products, more preventative features will be installed to detect early health complications.

Expo Go’s developmental process is secure, encrypted by device authentication IDs, and password-protected. Data transmitted from hardware to software is confidential and should be treated as such. Since I chose to develop using an iOS platform, the user must purchase a yearly subscription ($99.99) for an Apple Developers account to have the ability to build and submit the apps for approval. There are numerous security measures taken on behalf of Apple to avoid unauthorized device access or data breaching. Expo provided an option to use their Expo Application server (EAS) for Windows PC users coding React Native for iOS devices, a framework that handles the permissions and dependencies to develop and deploy apps. It is a streamlined one-to-one approach, but if SynkLinx production increases, server volume and safety will need to also.

Table 6. Ethical and Professional Considerations

Ethical and Professional Considerations	
Public Health	Aligning with the Hippocratic Oath and the National Society of Professional Engineers (NSPE) Code of Ethics, the user's health and well-being will be protected.
Safety/Wellness	The device shall be tested on the subject in normal conditions, using safe practices and assistance on standby.
Global Impact	If prototype goals are accomplished, device production will span worldwide.
Cultural Impact	The product will create a new sub-category of wearables, featuring customizable necklace hardware, for consumer personalization.
Social factors	The device will monitor important daily health metrics in real time and notify users to seek medical attention if necessary.
Environmental factors	Device uses minimal amounts of plastics. Raw materials are non-toxic and do not contribute to hazardous waste.
Economic factors	Target budget is based on market competition. For a new device, seller discretion and demand determines price.

7. Conclusions

In closing, the “SynkLinx” device was successful in collecting the heart rate, blood oxygen percentage, and core temperature through the designated sensors, then transmitting the data to the iOS app for viewing. I failed to yield a 5 percent error for all data measurements but managed to get each vital sign within a 5-10% error; a respectable showing given the materials and time limit. Due to time constraints, I opted to not include daily or hourly averages for the monitored vitals. The device focuses on real-time output and sends the user notifications for any abnormal readings that may occur while wearing the device. Safety and financial obligations were of high priority; the latter being based on current market trends. The raw materials used to create the necklace are of moderate quality, therefore did not accrue any substantial expenses. All materials, including the battery, sensors and boards were lightweight, assisting with the pragmatism of a necklace feel. The price point for the SynkLinx prototype ($199.99), is reasonable value, especially for a product that is the first of its kind.

Major challenges included finding a lithium-ion battery that fit within my volume requirements and had the ability to power the device for a consider amount of time. Although the ESP32-C3 is a low energy circuit board, I did not implement a “sleep” feature, which means the device does not conserve battery life and is always “on”. Minor challenges included finding a detachable shell for the device. I am certain that it can be done, but the technical aspect is the focal point of the prototype. Eventually, other modifications will be made to give the consumer a chance to swap bands for personal style preferences (Fitbit, Samsung, Apple, Withings, excetera.). Managing to find shells or coverings to protect the MAX30102 and MCP9808 sensors was tough. The iOS app has a minimal layout and animation, but achieves the goals set. Ideally, I wanted to keep an index for periodic vital readings as a reference for a physician if health problems occurred while wearing, but app development is a building block process, and I will need more than 3.5 months to implement additional features.

Looking ahead to post product launch, I believe smaller boards and quality materials will be manufactured to accommodate smart necklaces. This space has yet to be explored in depth, but I hope that my vision can garner the attention of technological field , and many will consider the viability of SynkLinx or a like-product in real world applicability. I am sure that I am not the first nor the last individual to contemplate or pursue the idea of a wearable smart necklace, but this device is the missing “link” to attach this dream to reality.

8. References

1) “Target heart rates chart,” www.heart.org, https://www.heart.org/en/healthy-living/fitness/fitness-basics/target-heart-rates (accessed Nov. 23, 2025).

2) C. Design, “Cirkit designer tutorials,” How to Use ESP32-C3: Pinouts, Specs, and Examples | Cirkit Designer, https://docs.cirkitdesigner.com/component/61fdce84-8458-d3fe-e238-99bd7de481ac (accessed Oct. 14, 2025).

3) “Getting started with Seeed Studio Xiao ESP32C3: Seeed Studio Wiki,” Seeed Studio Wiki RSS, https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/mean?srsltid=AfmBOoqvWcNRbIs7d_XkGeXQt0bry0GWvOcF_mrBCosNHNAv1wTUSIxP (accessed Nov. 26, 2025).

4) E. V. Osilla, “Physiology, temperature regulation,” StatPearls [Internet]., https://www.ncbi.nlm.nih.gov/books/NBK507838/ (accessed Dec. 3, 2025).

5) D. Akins, “Understanding pulse oximeter metrics: What your oxyknight® fingertip Oximeter Readings mean,” CMI Health Store, https://www.cmihealth.com/blogs/news/understanding-pulse-oximeter-metrics-what-your-oxyknight%C2%AE-fingertip-oximeter-readings-

mean?srsltid=AfmBOoqvWcNRbIs7d_XkGeXQt0bry0GWvOcF_mrBCosNHNAv1wTUSIxP (accessed Sept. 12, 2025).

6) "The World's Best Hospital," Mayo Clinic, https://www.mayoclinic.org/ (accessed Nov. 8, 2025).

Appendices

Appendix A: WBS and Gantt Chart, Resources Required

Work Break Down Structure

1) Project Planning & Requirements

- Define project scope and goals
- Research wearable technologies
- Identify accuracy requirements
- Determine device constraints (size, weight, power)
- Review relevant standards
- Develop initial design specifications

2) Hardware Design & Development

- Select development board (ESP32-C3)
- Select sensors (MAX30102, MCP9808)
- Power system & battery planning
- Breadboard wiring prototype
- Soldering and assembly

3) Firmware Development

- MAX30102 heart rate firmware
- SpO_2 algorithm integration

- Temperature firmware (MCP9808)
- BLE service & characteristic creation
- UUID mapping & syncing
- Firmware flashing & verification

4) Software / App Development

- React Native project setup
- Develop app screens
  - ❖ Home Page
  - ❖ Device Connection Modal
  - ❖ Dashboard
- BLE communication (useBLE hook)
- Implement UI animations
- Add vital alerts logic
- Expo Go iOS build testing

5) System Integration

- BLE stability testing
- Verify firmware ↔ app sync
- Real-time data streaming validation
- Sensor positioning optimization
- Troubleshoot device pairing issues

6) Testing & Validation

- Gather vital sign readings
- Compare with reference devices
- Capture serial logs
- Calculate percent error
- Optimize algorithms based on results

7) Documentation & Writing

- Executive Summary
- System documentation
- Create figures, tables, and charts
- Write percent error analysis
- Compile appendices
- Finalize report

8) Prototype Preparation

- Refine hardware assembly
- Confirm app demonstration functionality
- Prepare presentation materials
- Backup firmware/app builds
- Final review build submission

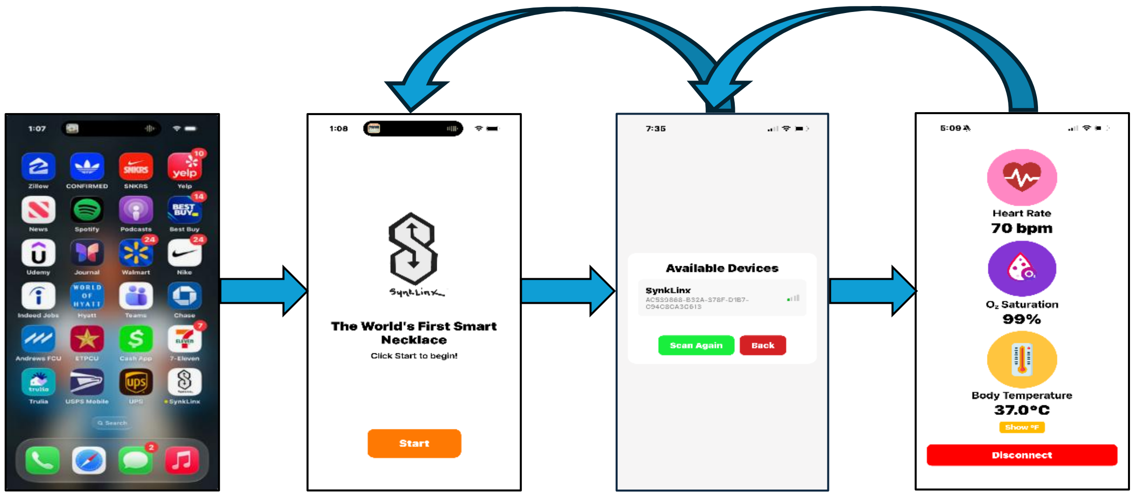
Gantt Chart

Task	Wk 1	Wk 2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12	Wk 13	Wk 14	Wk 15
Planning	🟦	🟦													
Hardware Build		🟦	🟦	🟦	🟦	🟦									
Firmware			🟦	🟦	🟦	🟦	🟦								
App Development					🟦	🟦	🟦	🟦	🟦						
Integration							🟦	🟦	🟦	🟦					
Testing								🟦	🟦	🟦	🟦	🟦	🟦		
Report Writing	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Final Prep												🟦	🟦	🟦	🟦

Appendix B: User Manuals/Instructions

Device in Use

App Transition Screens

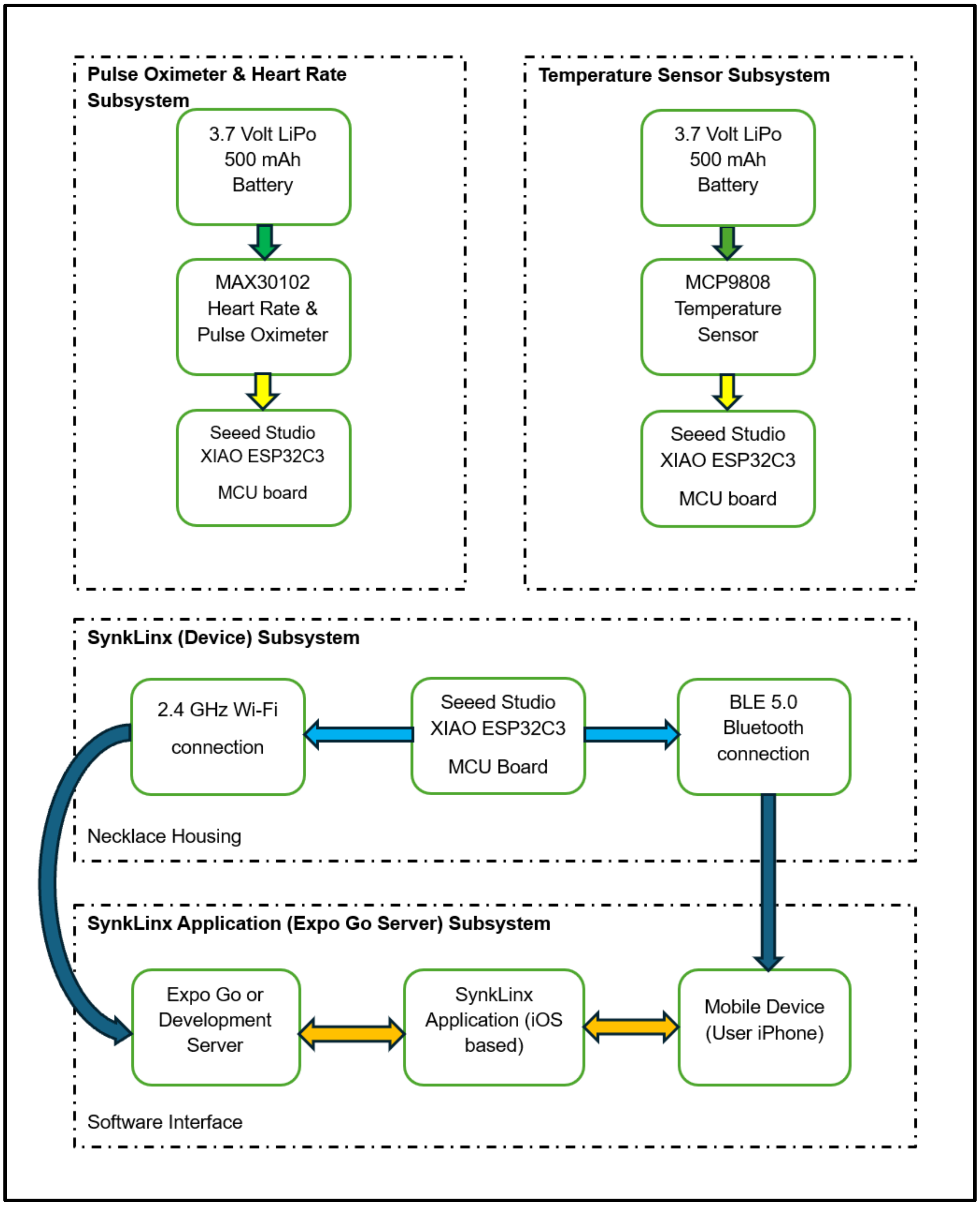


1) After device purchase, on your iPhone open the iOS App Store and in the search tab, type “SynkLinx.” Once found, download the application.
2) Check that the SynkLinx device has sufficient charge, then power the device up, (current prototype has no LED power indicator)
3) Attach and adjust the prototype to comfortably fit around the test subject’s neck.
4) Open your iPhone and select the SynkLinx iOS app, press start, then scan for available devices (SynkLinx).
5) Once connected, give the device about 5-10 seconds to process the subject’s heart rate, oxygen saturation, and body temperature.
6) While monitoring the application, results should show up in real-time, along with warnings and danger alerts regarding heart rate, blood oxygen percentage, and core temperature.
7) Users can toggle between Fahrenheit and Celsius temperature units. And view notifications are vital signs deviate from normal ranges.
8) After usage, select disconnect, and application will return to the pairing screen. If user has no desire to pair device again, press back, or close the app.

Appendix C: Knowledge and Skills acquired from Previous Coursework

Course	Knowledge/Skills
EENG 3304	Circuit Analysis
EENG 3308	Arduino coding and circuit design
EENG 4307	Flowchart analysis
COSC 4345	JavaScript programming, web/app design
CMPE 4309	ESP32 handling and development board experience

Appendix D: Block Diagram


Pulse Oximeter & Heart Rate Subsystem
3.7 Volt LiPo 500 mAh Battery
MAX30102 Heart Rate & Pulse Oximeter
Seeed Studio XIAO ESP32C3 MCU board
Temperature Sensor Subsystem
3.7 Volt LiPo 500 mAh Battery
MCP9808 Temperature Sensor
Seeed Studio XIAO ESP32C3 MCU board
SynkLinx (Device) Subsystem
2.4 GHz Wi-Fi connection
Seeed Studio XIAO ESP32C3 MCU Board
BLE 5.0 Bluetooth connection
Necklace Housing
SynkLinx Application (Expo Go Server) Subsystem
Expo Go or Development Server
SynkLinx Application (iOS based)
Mobile Device (User iPhone)
Software Interface

Appendix E: Hardware Components Used in Project

Product Name	Manufacturer	Part #
Seeed Studios XIAO ESP32-C3 Dev Board	Seeed Technology Co., Ltd	1597-113991054-ND
MAX30102 Heart Rate and Pulse Oximeter	SunFounder	4411-ST0244-ND
MCP9808 Temperature Sensor Breakout board	Adafruit Industries LLC	1528-1032-ND
3.7 V 500mAh Rechargeable Lithium-ion Polymer Battery	YDL	702030

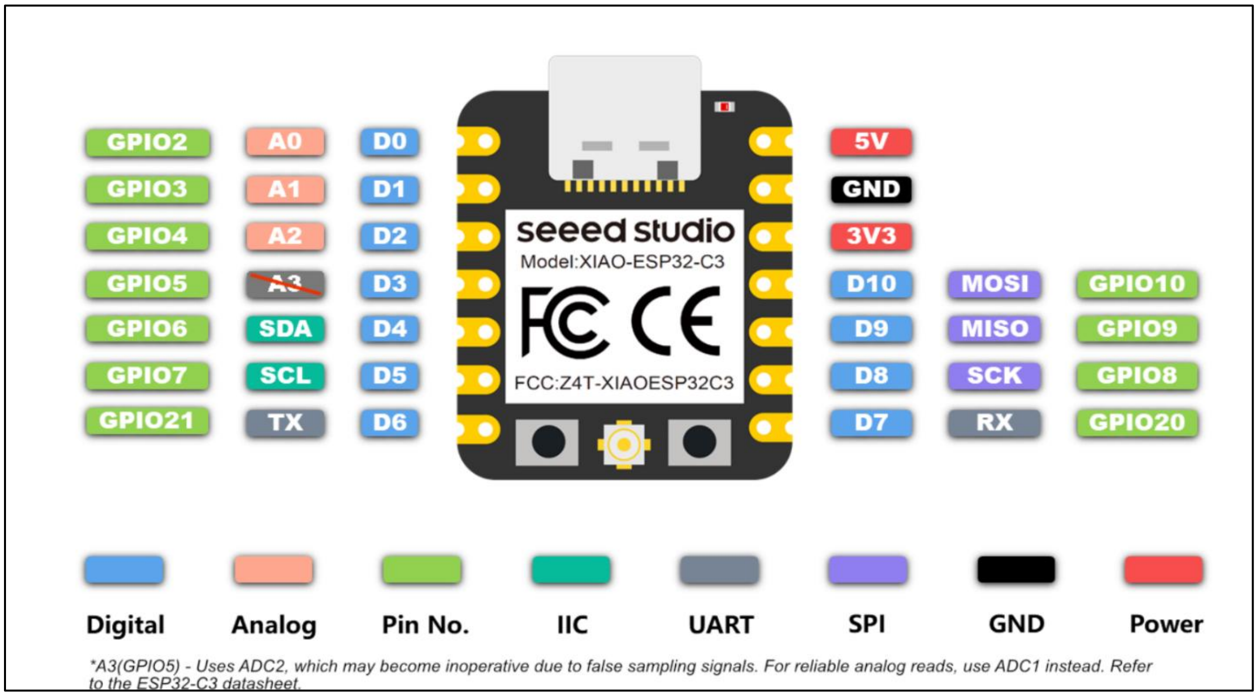
Appendix F: Software Components Used in Project

Name	URL	Software Version	Purpose
Arduino IDE	https://www.arduino.cc/en/software/	Arduino IDE 2.3.6	Program used to code hardware devices
Microsoft Visual Studio Code	https://code.visualstudio.com/download	1.105.1	Interface that combines code from hardware to application

Expo Development	https://expo.dev/signup	54.0.1	Development build for mobile application
GitHub	https://github.com/friyiajr/BLESampleExpo/tree/main	N/A	Reference for application design
Cirkit Designer	https://app.cirkitdesigner.com/	N/A	Reference for sensor wiring layout

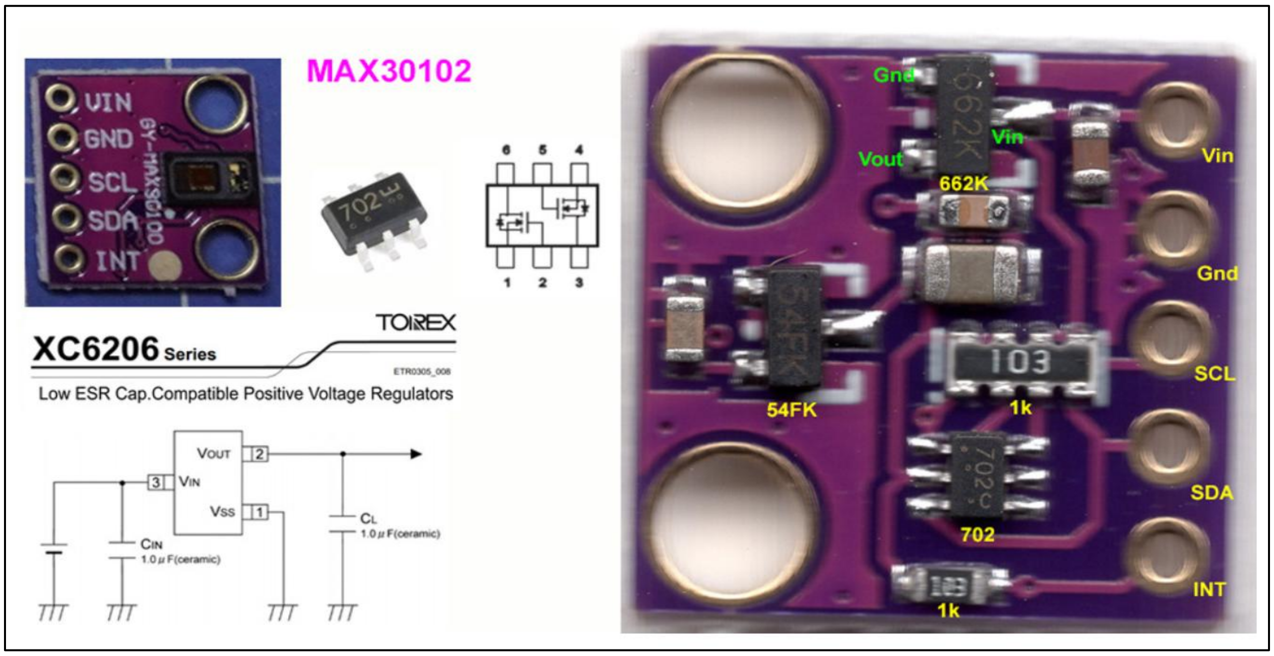
Appendix G: Part Schematics and Wiring Diagrams

XIAO Seeed Studio ESP32-C3 Development Board

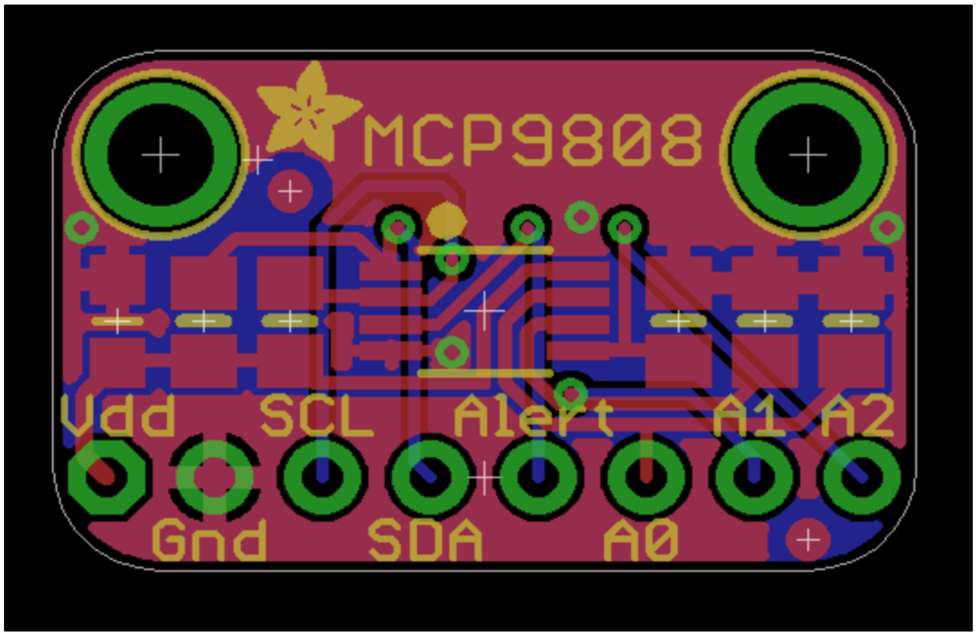


*A3(GPIO5) - Uses ADC2, which may become inoperative due to false sampling signals. For reliable analog reads, use ADC1 instead. Refer to the ESP32-C3 datasheet.

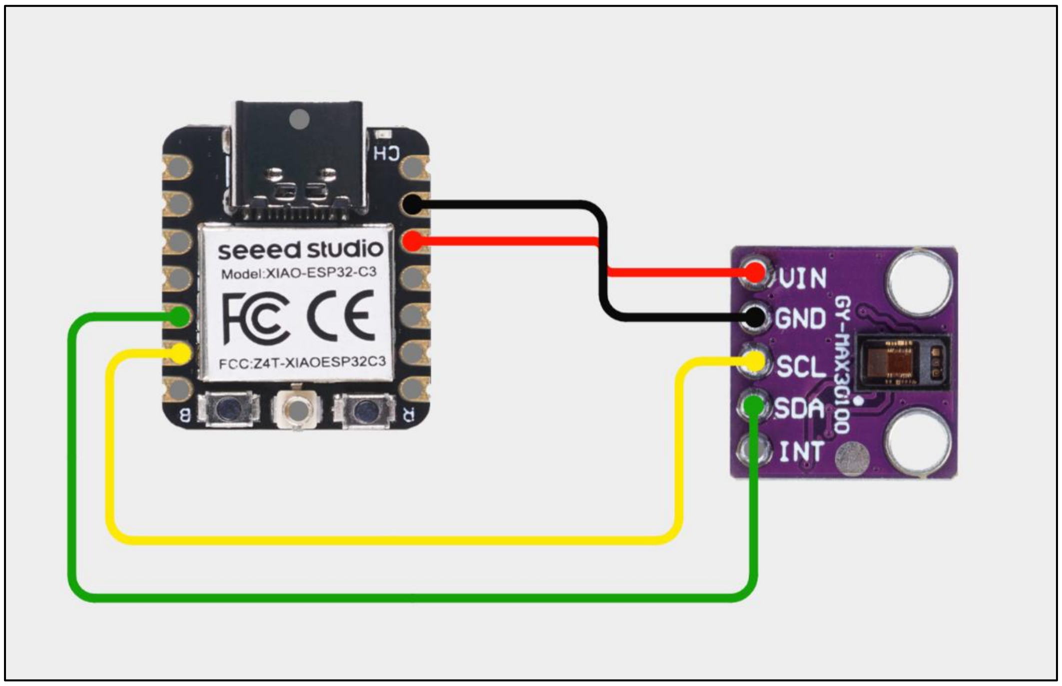
MAX30102 Heart Rate and Pulse Oximeter



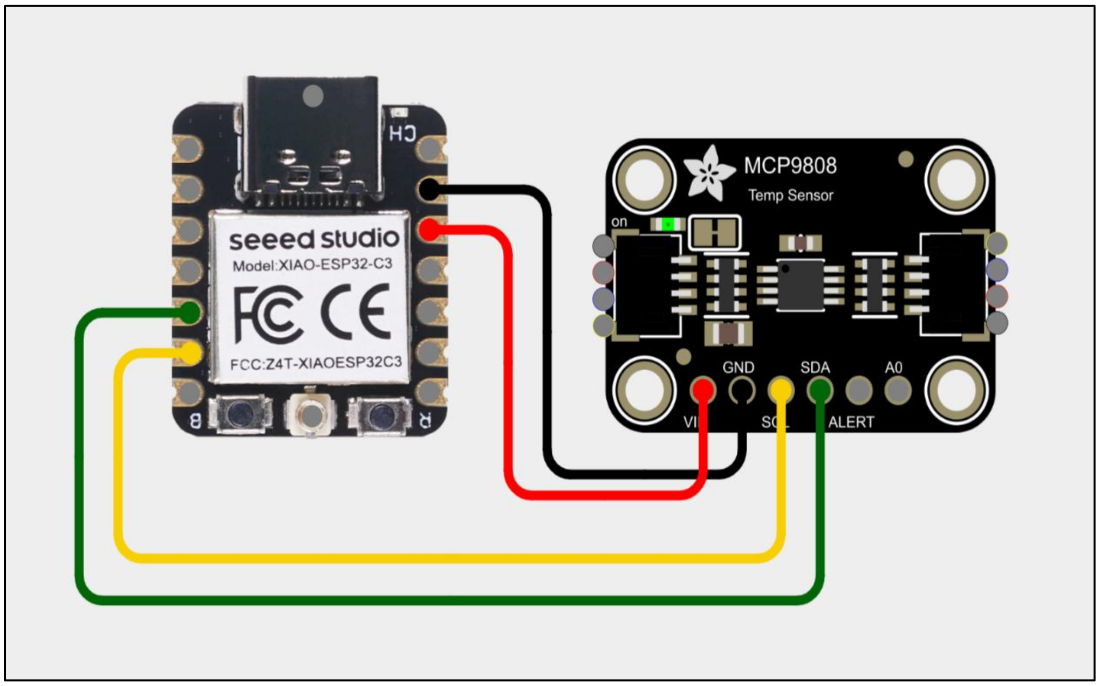
MCP9808 Temperature Sensor Breakout Board



Wiring Layout for Heart Rate/Pulse Oximeter Sensor



Wiring Layout for Temperature Sensor



Appendix H: Vital Signs Testing Data

Time (s)	Reference Equipment			SynkLinx		
	Heart Rate (bpm)	Oxygen Saturation (%)	Core Temperature (C)	Heart Rate (bpm)	Oxygen Saturation (%)	Core Temperature (C)
1	79.00	96.00	38.00	51.00	97.00	36.10
2	79.10	96.00	38.00	51.00	97.00	36.10
3	79.20	96.00	37.94	62.00	97.00	36.10
4	76.00	96.00	37.94	62.00	97.00	36.00
5	76.10	94.00	37.94	62.00	97.00	35.90
6	57.00	94.00	37.94	62.00	98.00	36.00
7	56.00	95.00	37.83	62.00	97.00	36.10
8	56.10	95.00	37.83	62.00	96.00	36.20
9	56.20	95.00	37.11	62.00	96.00	36.30
10	55.00	96.00	37.11	62.00	96.00	36.30
11	55.10	96.00	37.28	64.00	96.00	36.30
12	54.00	96.00	37.28	64.00	96.00	36.30
13	54.10	96.00	37.28	62.00	96.00	36.30
14	54.20	96.00	37.28	62.00	96.00	36.30
15	53.00	96.00	37.28	62.00	96.00	36.30
16	53.10	96.00	37.28	62.00	96.00	36.40
17	52.00	96.00	37.28	62.00	96.00	36.30
18	52.10	97.00	37.28	62.00	96.00	36.30
19	52.20	97.00	37.28	62.00	96.00	36.30
20	51.00	97.00	37.28	62.00	95.00	36.30
21	51.10	97.00	37.22	62.00	95.00	36.30
22	51.20	97.00	37.22	62.00	95.00	36.30
23	52.30	97.00	37.17	62.00	96.00	36.40
24	52.40	97.00	37.17	62.00	96.00	36.30
25	52.50	97.00	37.17	62.00	97.00	36.30
26	53.20	97.00	37.17	62.00	97.00	36.20
27	53.30	97.00	37.11	62.00	97.00	36.10
28	54.30	97.00	37.11	62.00	97.00	36.20
29	56.30	97.00	37.11	62.00	97.00	36.30
30	57.10	97.00	37.11	62.00	97.00	36.40

Appendix I: Device Specifications

Prototype Specifications					
Component	Function	Data Retrieval/Sampling	Dimensions(mm)	Weight	Sensor Range/Accuracy
Battery: 3.7 Volt LiPo 500mAh	Powers wearable smart necklace	N/A	52 x 20 x 5 (L x W x T)	10g	Battery life ranges up to 4 hours.
XIAO ESP32-C3	MCU: BLE, Wi-Fi, sensor comm	I2C up to 400 kHz	21 X 17.5 x 3.5	~3g	N/A
MAX30102 Pulse Oximeter / HR Sensor	Heart Rate & SpO_2	Configurable up to 3200 samples/sec	~12 x 11 x 3	~1g	HR: 30–250 BPM SpO_2: 0–100% Accuracy: HR ±1–2 BPM, SpO_2 ±2–3%
MCP9808 Temperature Sensor	Body/ Ambient Temperature	0.25 °C resolution, ~250 ms conversion	4.5 x 4.5 x 1.5	~1g	-40 °C to +125 °C Accuracy ±0.5 °C
Wiring/SynkLinx Housing	Interconnect and wearable housing	N/A	558 x 25 x 10	~20g	N/A

Appendix J: Custom Software Coding

Arduino Code

Firmware.cpp

```
//  - SparkFun MAX30102 PBA Heart Rate
//  - SparkFun Maxim SpO2 Algorithm
//  - Adafruit MCP9808 Temperature
//  - BLE notify using UUIDs

#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "spo2_algorithm.h"
#include <Adafruit_MCP9808.h>

#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

// BLE UUIDs
#define SYNKLINX_SERVICE "e5df6019-cf42-49f6-a418-346db96363f6"
#define HEART_RATE_CHAR  "3c1f4fe4-c7ce-4c09-a38a-ba166fce06c6"
#define O2_CHAR          "9749ccd9-940e-4ad6-bce9-d246a8d30dca"
#define TEMP_CHAR        "69b82322-1c56-423e-820e-eb08c3f030d4"

// SENSORS
MAX30105 sensor;
Adafruit_MCP9808 tempsensor;

// HR (SparkFun PBA)
const byte RATE_SIZE = 4;
```

```
byte rates[RATE_SIZE];
byte rateSpot = 0;
long lastBeat = 0;

float beatsPerMinute = 0;
int beatAvg = 0;

// SpO2 (SparkFun Maxim)
#define BUFFER_SIZE 100
uint32_t irBuffer[BUFFER_SIZE];
uint32_t redBuffer[BUFFER_SIZE];
bool bufferFilled = false;
int bufferIndex = 0;

float spo2Filtered = 98.0;

// Temperature
float coreTemp = 36.5;

// BLE
BLECharacteristic *hrChar;
BLECharacteristic *spo2Char;
BLECharacteristic *tempChar;

void setup() {
  Serial.begin(115200);
  delay(200);

  Wire.begin(4, 5, 400000);

  // MAX30102 INIT
  Serial.println("Initializing MAX30102...");
  if (!sensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 NOT found!");
    while (1);
  }
  sensor.setup();
  sensor.setPulseAmplitudeRed(0x2A);
  sensor.setPulseAmplitudeIR(0x2A);
  sensor.setPulseAmplitudeGreen(0);

  // MCP9808 INIT
  Serial.println("Initializing MCP9808...");
  if (!tempsensor.begin(0x18)) {
    Serial.println("MCP9808 NOT found!");
    while (1);
  }
  tempsensor.setResolution(3);

  // BLE INIT
  BLEDevice::init("SynkLinx");
  BLEServer *server = BLEDevice::createServer();
  BLEService *service = server->createService(SYNKLINX_SERVICE);

  hrChar   = service->createCharacteristic(HEART_RATE_CHAR,
BLECharacteristic::PROPERTY_NOTIFY);
  spo2Char = service-
>createCharacteristic(O2_CHAR,         BLECharacteristic::PROPERTY_NOTIFY);
  tempChar = service-
>createCharacteristic(TEMP_CHAR,       BLECharacteristic::PROPERTY_NOTIFY);

  hrChar->addDescriptor(new BLE2902());
  spo2Char->addDescriptor(new BLE2902());
  tempChar->addDescriptor(new BLE2902());

  service->start();
  server->getAdvertising()->start();

  Serial.println("BLE Advertising: SynkLinx");
}

void loop() {
```

```
if (!sensor.available()) {
  sensor.check();
  return;
}

uint32_t ir = sensor.getIR();
uint32_t red = sensor.getRed();
sensor.nextSample();

if (checkForBeat(ir)) {
  long now = millis();
  long delta = now - lastBeat;
  lastBeat = now;

  float bpm = 60.0 / (delta / 1000.0);

  if (bpm >= 20 && bpm <= 220) {
    beatsPerMinute = bpm;

    rates[rateSpot++] = (byte)bpm;
    rateSpot %= RATE_SIZE;

    int sum = 0;
    for (byte i = 0; i < RATE_SIZE; i++) sum += rates[i];
    beatAvg = sum / RATE_SIZE;
  }
}

irBuffer[bufferIndex] = ir;
redBuffer[bufferIndex] = red;

bufferIndex++;
if (bufferIndex >= BUFFER_SIZE) {
  bufferIndex = 0;
  bufferFilled = true;
}
```

```
if (bufferFilled) {
  int32_t spo2, hrDummy;
  int8_t validSpo2, validHR;

  maxim_heart_rate_and_oxygen_saturation(
    irBuffer, BUFFER_SIZE,
    redBuffer,
    &spo2, &validSpo2,
    &hrDummy, &validHR
  );

  if (validSpo2 && spo2 >= 70 && spo2 <= 100) {
    spo2Filtered = spo2Filtered * 0.7 + spo2 * 0.3;
  }
}

float skinC = tempsensor.readTempC();
coreTemp = skinC + 5.2;  // simple estimate



// BLE Notify
if (BLEDevice::getInitialized()) {

  uint8_t hrOut  = (uint8_t)beatAvg;
  uint8_t spOut  = (uint8_t)spo2Filtered;
  int16_t tempOut = (int16_t)(coreTemp * 100);

  hrChar->setValue(&hrOut, 1);
  spo2Char->setValue(&spOut, 1);
  tempChar->setValue((uint8_t *)&tempOut, 2);

  hrChar->notify();
  spo2Char->notify();
  tempChar->notify();
}

// Debugging print
```

```
  Serial.printf("IR:%lu | HR:%d | SpO2:%.1f | Core:%.2f°C\n",
    ir, beatAvg, spo2Filtered, coreTemp
  );
}
```

React Native Frameworks

App.jsx

```
import React from "react";
import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import FlashMessage from "react-native-flash-message";

import HomePage from "./screens/HomePage";
import DeviceModal from "./screens/DeviceModal";
import Dashboard from "./screens/Dashboard";

import { BLEProvider } from "./BLEProvider";

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <BLEProvider>
      <NavigationContainer>
        <Stack.Navigator
          initialRouteName="Home"
          screenOptions={{ headerShown: false, animation: "slide_from_right" }}
        >
          <Stack.Screen name="Home" component={HomePage} />
          <Stack.Screen name="DeviceModal" component={DeviceModal} />
          <Stack.Screen name="Dashboard" component={Dashboard} />
        </Stack.Navigator>

        <FlashMessage position="top" floating />
      </NavigationContainer>
    </BLEProvider>
  );
}
```

HomePage.jsx

```
import React, { useRef, useEffect } from "react";
import {
  SafeAreaView,
  Text,
  TouchableOpacity,
  Image,
  Animated,
  StatusBar,
  StyleSheet,
  View,
} from "react-native";
import { useNavigation } from "@react-navigation/native";
import Logo from "../assets/Logos/SynkLinx_Logo.png";

const HomePage = () => {
  const navigation = useNavigation();

  // Animations for each section
  const logoFade = useRef(new Animated.Value(0)).current;
  const titleFade = useRef(new Animated.Value(0)).current;
  const subtitleFade = useRef(new Animated.Value(0)).current;
  const buttonFade = useRef(new Animated.Value(0)).current;

  const logoSlide = useRef(new Animated.Value(20)).current;
  const textSlide = useRef(new Animated.Value(20)).current;

  useEffect(() => {
    Animated.sequence([
      // Logo appears first
      Animated.parallel([
        Animated.timing(logoFade, { toValue: 1, duration: 800, useNativeDriver:
true }),
        Animated.timing(logoSlide, { toValue: 0, duration: 800, useNativeDriver:
true }),
      ]),

      // Title + subtitle together
      Animated.parallel([
        Animated.timing(titleFade, { toValue: 1, duration: 600, useNativeDriver:
true }),
        Animated.timing(subtitleFade, { toValue: 1, duration: 600,
useNativeDriver: true }),
        Animated.timing(textSlide, { toValue: 0, duration: 600, useNativeDriver:
true }),
      ]),

      // Button fades last
      Animated.timing(buttonFade, { toValue: 1, duration: 600, useNativeDriver:
true }),
    ]).start();
  }, []);


  return (
    <SafeAreaView style={styles.container}>
      <StatusBar barStyle="dark-content" backgroundColor="#fff" />

      <View style={styles.centerContent}>
        {/* Logo Animation */}
        <Animated.View
          style={{ opacity: logoFade, transform: [{ translateY: logoSlide }] }}
        >
          <Image source={Logo} style={styles.logo} />
        </Animated.View>

        {/* Text Animation */}
        <Animated.View
          style={{ opacity: titleFade, transform: [{ translateY: textSlide }] }}
        >
          <Text style={styles.title}>The World's First Smart Necklace</Text>
        </Animated.View>

        <Animated.View
          style={{ opacity: subtitleFade, transform: [{ translateY: textSlide }]
}}
        >
          <Text style={styles.subtitle}>Click Start to begin!</Text>
        </Animated.View>
      </View>

      {/* Button stays bottom-ish and fades in */}
      <Animated.View style={[styles.buttonWrapper, { opacity: buttonFade }]}>
        <TouchableOpacity style={styles.button} onPress={() =>
navigation.navigate("DeviceModal")}
>
          <Text style={styles.buttonText}>Start</Text>
        </TouchableOpacity>
      </Animated.View>
    </SafeAreaView>
  );
};

export default HomePage;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#fff",
  },

  // Ensures proper centering on all screen sizes
  centerContent: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    paddingHorizontal: 20,
  },

  logo: {
    width: 260,
    height: 260,
    resizeMode: "contain",
    marginBottom: 15,
  },

  title: {
    fontSize: 24,
    fontWeight: "bold",
    textAlign: "center",
    marginBottom: 10,
  },

  subtitle: {
    fontSize: 16,
    textAlign: "center",
    marginBottom: 20,
  },

  buttonWrapper: {
    alignItems: "center",
    marginBottom: 40,
  },

  button: {
    backgroundColor: "#FF7A00",
    paddingVertical: 20,
    paddingHorizontal: 60,
    borderRadius: 14,
  },
  buttonText: {
    color: "#fff",
    fontSize: 20,
    fontWeight: "bold",
  },
});
```

DeviceModal.jsx

```
import React, { useEffect, useState } from "react";
import {
  SafeAreaView,
  View,
  Text,
  TouchableOpacity,
  FlatList,
  Pressable,
  StyleSheet,
  ActivityIndicator,
} from "react-native";
import Animated, {
  FadeIn,
  FadeOut,
  SlideInUp,
  SlideOutDown,
  withRepeat,
  withTiming,
  useSharedValue,
  useAnimatedStyle,
} from "react-native-reanimated";
import { useNavigation } from "@react-navigation/native";
import { useBLEContext } from "../BLEProvider";

const DeviceModal = () => {
  const navigation = useNavigation();
  const { devices, connectToDevice, scanning, scanForPeripherals, stopScanning }
=
    useBLEContext();
  const [connectingId, setConnectingId] = useState(null);

  const glow = useSharedValue(0.4);

  // Pulse animation
  useEffect(() => {
    glow.value = scanning
      ? withRepeat(withTiming(1, { duration: 900 }), -1, true)
      : withTiming(0.4, { duration: 300 });
  }, [scanning]);

  const glowStyle = useAnimatedStyle(() => ({ opacity: glow.value }));

  // Auto scan on mount
  useEffect(() => {
    scanForPeripherals({ autoRetry: true });
    return () => stopScanning();
  }, []);

  // Connect
  const handleConnect = async (device) => {
    setConnectingId(device.id);
    try {
      await connectToDevice(device);
      navigation.navigate("Dashboard");
    } catch (error) {
      console.log("Connection failed:", error);
      alert(`Failed to connect to ${device.name || "device"}.`);
    } finally {
      setConnectingId(null);
    }
  };

  const renderItem = ({ item }) => {
    const isConnecting = connectingId === item.id;

    return (
      <Pressable
        style={({ pressed }) => [
          styles.deviceItem,
          pressed && { backgroundColor: "rgba(0,0,0,0.05)" },
          isConnecting && { opacity: 0.5 },
        ]}
        onPress={() => handleConnect(item)}
        disabled={isConnecting}
      >
        <View style={{ flex: 1 }}>
          <Text style={styles.deviceName}>{item.name}</Text>
        </View>
        {isConnecting && <ActivityIndicator size="small" color="#12ed3e" />}
      </Pressable>
    );
  };

  return (
    <SafeAreaView style={styles.container}>
      <Animated.View entering={FadeIn} exiting={FadeOut}
style={styles.fullscreen}>
        <Animated.View
          entering={SlideInUp.springify().damping(12).stiffness(120)}
          exiting={SlideOutDown}
          style={styles.contentContainer}
        >
          <Text style={styles.title}>Available Devices</Text>

          {scanning && (
            <View style={styles.scanningRow}>
              <Animated.View style={[styles.scanDot, glowStyle]} />
              <Text style={styles.scanningText}>Scanning…</Text>
            </View>
          )}
          {devices.length === 0 && !scanning ? (
            <Text style={styles.empty}>
              No devices found. Please make sure your SynkLinx device is on.
            </Text>
          ) : (
            <FlatList
              data={devices}
              keyExtractor={(item) => item.id}
              renderItem={renderItem}
              contentContainerStyle={{ paddingBottom: 20 }}
              style={{ width: "100%" }}
              showsVerticalScrollIndicator={false}
            />
          )}

          <View style={styles.buttonRow}>
            <TouchableOpacity
              style={styles.scanButton}
              onPress={() => scanForPeripherals({ autoRetry: true })}
              disabled={scanning}
            >
              <Text style={styles.scanButtonText}>Scan Again</Text>
            </TouchableOpacity>

            <TouchableOpacity style={styles.closeButton} onPress={() =>
            navigation.navigate("Home")}>
              <Text style={styles.closeButtonText}>Back</Text>
            </TouchableOpacity>
          </View>
        </Animated.View>
      </Animated.View>
    </SafeAreaView>
  );
};

export default DeviceModal;
```

```
const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: "#f5f5f5" },
  fullscreen: { flex: 1, justifyContent: "center", alignItems: "center" },
  contentContainer: {
    width: "90%",
    maxHeight: "90%",
    backgroundColor: "#fff",
    borderRadius: 16,
    padding: 20,
    alignItems: "center",
    justifyContent: "space-between",
  },
  title: { fontSize: 22, fontWeight: "700", color: "#000", marginBottom: 12 },
  scanningRow: { flexDirection: "row", alignItems: "center", marginBottom: 12 },
  scanDot: { width: 10, height: 10, backgroundColor: "#3d3a45ff", borderRadius:
10, marginRight: 8 },
  scanningText: { fontSize: 16, color: "#555" },
  empty: { fontSize: 16, color: "#888", marginVertical: 20, textAlign: "center"
},
  deviceItem: {
    flexDirection: "row",
    alignItems: "center",
    padding: 14,
    width: "100%",
    borderRadius: 10,
    backgroundColor: "rgba(0,0,0,0.03)",
    marginBottom: 10,
  },
  deviceName: { fontSize: 17, fontWeight: "600", color: "#000" },
  buttonRow: { flexDirection: "row", marginTop: 15, marginBottom: 0 },
  scanButton: { backgroundColor: "#12ed3e", paddingVertical: 12,
paddingHorizontal: 22, borderRadius: 10 },
  scanButtonText: { color: "#fff", fontWeight: "700", fontSize: 16 },
  closeButton: { marginLeft: 10, backgroundColor: "#d32121", paddingVertical: 12,
paddingHorizontal: 22, borderRadius: 10 },
  closeButtonText: { color: "#fff", fontWeight: "700", fontSize: 16 },
});
```

Dashboard.jsx

```
import React, { useState, useRef, useEffect } from "react";
import {
  SafeAreaView,
  View,
  Text,
  TouchableOpacity,
  StyleSheet,
  Image,
  Animated,
  Dimensions,
} from "react-native";
import { useNavigation } from "@react-navigation/native";
import { useBLEContext } from "../BLEProvider";
import PulseIndicator from "../PulseIndicator";
import { runVitalAlerts } from "../dataAlert";

import Heart_Rate from "../assets/Heart_Rate.png";
import O2_Sat from "../assets/O2_Sat.png";
import Body_Temp from "../assets/Body_Temp.png";

const { height: SCREEN_HEIGHT } = Dimensions.get("window");
const PULSE_SIZE = SCREEN_HEIGHT / 8;

const Dashboard = () => {
  const navigation = useNavigation();
  const {
    connectedDevice,
    disconnect,
    heartRate = 0,
    O2Sat = 0,
    temperature = 0,
  } = useBLEContext();
  const [isCelsius, setIsCelsius] = useState(true);

  // Animated button pulse
  const pulseAnim = useRef(new Animated.Value(1)).current;
  useEffect(() => {
    Animated.loop(
      Animated.sequence([
        Animated.timing(pulseAnim, { toValue: 1.05, duration: 800,
useNativeDriver: true }),
        Animated.timing(pulseAnim, { toValue: 1, duration: 800, useNativeDriver:
true }),
      ])
    ).start();
  }, []);

  // Run vital alerts
  useEffect(() => {
    if (connectedDevice) {
      runVitalAlerts({ heartRate, O2Sat, temperature, isCelsius });
    }
  }, [heartRate, O2Sat, temperature, connectedDevice]);

  // Display helper
  const displayValue = (val, unit) =>
    val === null || val === undefined || val === 0 || isNaN(val) ? "--" :
`${val}${unit}`;

  // Pulse Speeds
  const getHeartRateSpeed = (hr) => {
    if (!hr || isNaN(hr)) return 0.5;
    if (hr <= 59) return 1;      // slow
    if (hr <= 90) return 2;      // moderate
    if (hr <= 130) return 4;     // fast
    return 6;                     // super-fast
  };

  const getO2Speed = (o2) => {
    if (!o2 || isNaN(o2) || o2 < 50) return 0.5;
    return 1 + ((Math.min(Math.max(o2, 50), 100) - 50) / 50); // 1-2 range
  };

  const getO2Color = (o2) => {
    if (!o2 || isNaN(o2)) return "rgb(210,180,255)";
    const normalized = Math.min(Math.max((o2 - 50) / 50, 0), 1);
    const r = 200 - normalized * 100;
    const g = 150 - normalized * 150;
    const b = 255 - normalized * 55;
    return `rgb(${r},${g},${b})`;
  };

  const convertTemp = (temp) => {
    if (!temp || isNaN(temp) || temp === 0) return "--";
    return isCelsius ? temp.toFixed(1) : ((temp * 9) / 5 + 32).toFixed(1);
  };

  const getTempColor = (temp) => {
    if (!temp || isNaN(temp) || temp === 0) return "rgba(168, 255, 255, 1)";
    if (temp <= 34.4) return "rgba(0, 255, 255, 1)";
    if (temp <= 37.2) return "rgba(62, 173, 37, 1)";
    if (temp <= 38.3) return "rgba(220, 111, 10, 1)";
    return "red";
  };

  const getTempSpeed = (temp) => {
    if (!temp || isNaN(temp) || temp === 0) return 0.5;
    if (temp <= 34.4) return 0.5;
    if (temp <= 37.2) return 2;
    if (temp <= 38.3) return 4;
    return 8;
  };

  // Render Metric
  const renderMetric = (pulseSize, speed, color, icon, label, valueContent,
toggle = false) => (
    <View style={styles.metricWrapper}>
      <View style={{ justifyContent: "center", alignItems: "center" }}>
        <PulseIndicator
          size={pulseSize}
          speedFactor={speed}
          color={color}
          intensity={0.3}
          darkenAmount={0.25}
          opacityRange={[0.2, 0.8]}
        />
        <Image
          source={icon}
                style={{ width: pulseSize * 0.75, height: pulseSize * 0.75,
      position: "absolute" }}
        />
      </View>
      <View style={styles.metricTextWrapper}>
        <Text style={styles.label}>{label}</Text>
        <Text style={styles.value}>{valueContent}</Text>
        {toggle && (
          <TouchableOpacity onPress={() => setIsCelsius(!isCelsius)}
style={styles.singleToggleButton}>
            <Text style={styles.singleToggleText}>Show {isCelsius ? "°F" :
"°C"}</Text>
          </TouchableOpacity>
        )}
      </View>
    </View>
  );

  return (
    <SafeAreaView style={styles.container}>
      <View style={styles.dataContainer}>
        <View style={styles.dataWrapper}>
          {connectedDevice ? (
            <>
              {renderMetric(
              PULSE_SIZE,
              getHeartRateSpeed(heartRate),
              "rgba(240, 20, 105, 0.9)",
              Heart_Rate,
              "Heart Rate",
              displayValue(heartRate, " bpm")
            )}
            {renderMetric(
              PULSE_SIZE,
              getO2Speed(O2Sat),
              getO2Color(O2Sat),
              O2_Sat,
              "O₂ Saturation",
              displayValue(O2Sat, "%")
            )}
            {renderMetric(
              PULSE_SIZE,
              getTempSpeed(temperature),
              getTempColor(temperature),
              Body_Temp,
              "Body Temperature",
              `${convertTemp(temperature)}°${isCelsius ? "C" : "F"}`,
              true
            )}
          </>
        ) : (
                <Text style={styles.value}>Disconnected. Please connect a
      device.</Text>
        )}
      </View>
    </View>

    <Animated.View style={{ transform: [{ scale: pulseAnim }] }}>
      <TouchableOpacity
        style={[styles.button, { backgroundColor: connectedDevice ? "red" :
"limegreen" }]}
        onPress={() => {
          if (connectedDevice) disconnect();
          navigation.navigate("DeviceModal");
        }}
      >
        <Text style={styles.buttonText}>{connectedDevice ? "Disconnect" :
"Search Devices"}</Text>
      </TouchableOpacity>
    </Animated.View>
  </SafeAreaView>
);
};

const styles = StyleSheet.create({
container: { flex: 1, backgroundColor: "white" },
dataContainer: { flex: 1, justifyContent: "center", alignItems: "center",
paddingVertical: 10 },
dataWrapper: { flex: 1, justifyContent: "space-evenly", alignItems: "center",
width: "100%" },
metricWrapper: { alignItems: "center", justifyContent: "center", width: "100%"
},
metricTextWrapper: { marginTop: 10, alignItems: "center" },
label: { fontSize: 20, fontWeight: "500", textAlign: "center" },
value: { fontSize: 28, fontWeight: "bold", textAlign: "center", marginTop: 2 },
button: {
  justifyContent: "center",
  alignItems: "center",
  height: 50,
  marginHorizontal: 20,
  marginBottom: 20,
  borderRadius: 10,
},
buttonText: { fontSize: 18, color: "white", fontWeight: "bold" },
singleToggleButton: {
  marginTop: 8,
  paddingVertical: 4,
  paddingHorizontal: 10,
  backgroundColor: "#ffbb00ff",
  borderRadius: 5,
},
singleToggleText: { color: "#fff", fontWeight: "bold", fontSize: 14 },
});

export default Dashboard;
```

dataAlert.jsx

```
import { showMessage } from "react-native-flash-message";

// Throttle timers
const ALERT_THROTTLE = 15000; // 15 seconds

// Last alert times
const lastAlertTime = {
  heartRate: 0,
  O2Sat: 0,
  temperature: 0,
};

const thresholds = {
  heartRate: {
    criticallyHigh: 180,
    high: 120,
    low: 50,
    criticallyLow: 30,
  },
  O2Sat: {
    low: 95,
    criticallyLow: 90,
  },
  tempC: {
    high: 38,
    low: 35,
  },
  tempF: {
    high: 100.4,
    low: 95,
  },
};

export function runVitalAlerts({ heartRate, O2Sat, temperature, isCelsius }) {
  const now = Date.now();

  // O2 Saturation
  if (O2Sat !== null && !isNaN(O2Sat)) {
    if (now - lastAlertTime.O2Sat > ALERT_THROTTLE) {
      if (O2Sat <= thresholds.O2Sat.criticallyLow) {
        showMessage({
          message: "Critically Low O₂",
          description: `SpO₂: ${O2Sat}%`,
          type: "danger",
          duration: 3000,
          icon: "danger",
        });
      } else if (O2Sat < thresholds.O2Sat.low) {
        showMessage({
          message: "Low O₂",
          description: `SpO₂: ${O2Sat}%`,
          type: "warning",
          duration: 3000,
          icon: "warning",
        });
      }
      lastAlertTime.O2Sat = now;
    }
  }

  //  Heart Rate
  if (heartRate !== null && !isNaN(heartRate)) {
    if (now - lastAlertTime.heartRate > ALERT_THROTTLE) {
      if (heartRate >= thresholds.heartRate.criticallyHigh) {
        showMessage({
```

```
        message: "Critically High HR",
        description: `Heart Rate: ${heartRate} bpm`,
        type: "danger",
        duration: 3000,
        icon: "danger",
      });
    } else if (heartRate >= thresholds.heartRate.high) {
      showMessage({
        message: "High HR",
        description: `Heart Rate: ${heartRate} bpm`,
        type: "warning",
        duration: 3000,
        icon: "warning",
      });
    } else if (heartRate < thresholds.heartRate.criticallyLow) {
      showMessage({
        message: "Critically Low HR",
        description: `Heart Rate: ${heartRate} bpm`,
        type: "danger",
        duration: 3000,
        icon: "danger",
      });
    } else if (heartRate < thresholds.heartRate.low) {
      showMessage({
        message: "Low HR",
        description: `Heart Rate: ${heartRate} bpm`,
        type: "warning",
        duration: 3000,
        icon: "warning",
      });
    }
    lastAlertTime.heartRate = now;
  }
}

//  Temperature
if (temperature !== null && !isNaN(temperature)) {
    const tempVal = isCelsius ? temperature : (temperature * 9) / 5 + 32;
    const tempThresholds = isCelsius ? thresholds.tempC : thresholds.tempF;

    if (now - lastAlertTime.temperature > ALERT_THROTTLE) {
      if (tempVal >= tempThresholds.high) {
        showMessage({
          message: "High Body Temperature",
          description: `Temperature: ${tempVal.toFixed(1)}°${isCelsius ? "C" :
"F"}`,
          type: "danger",
          duration: 3000,
          icon: "danger",
        });
      } else if (tempVal <= tempThresholds.low) {
        showMessage({
          message: "Low Body Temperature",
          description: `Temperature: ${tempVal.toFixed(1)}°${isCelsius ? "C" :
"F"}`,
          type: "danger",
          duration: 3000,
          icon: "danger",
        });
      }
      lastAlertTime.temperature = now;
    }
  }
}
```

PulseIndicator.jsx

```
import React, { useEffect, useRef } from "react";
import { Animated, StyleSheet, Easing } from "react-native";

const PulseIndicator = ({
  size = 50,
  speedFactor = 1,
  color = "red",
  intensity = 0.3,
  darkenAmount = 0.25,
  opacityRange = [0.25, 1]
}) => {
  const scaleAnim = useRef(new Animated.Value(1)).current;
  const colorAnim = useRef(new Animated.Value(0)).current;

  useEffect(() => {
    const duration = 1000 / speedFactor;

    Animated.loop(
      Animated.sequence([
        Animated.parallel([
          Animated.timing(scaleAnim, {
            toValue: 1 + intensity,
            duration,
            easing: Easing.inOut(Easing.ease),
            useNativeDriver: false, // scale uses native driver
          }),
          Animated.timing(colorAnim, {
            toValue: 0,
            duration,
            easing: Easing.inOut(Easing.ease),
            useNativeDriver: false, // color cannot use native driver
          }),
        ]),
        Animated.parallel([
          Animated.timing(scaleAnim, {
            toValue: 1,
            duration,
            easing: Easing.inOut(Easing.ease),
            useNativeDriver: false,
          }),
          Animated.timing(colorAnim, {
            toValue: 1,
            duration,
            easing: Easing.inOut(Easing.ease),
            useNativeDriver: false,
          }),
        ]),
      ])
    ).start();
  }, [speedFactor, intensity]);

  const animatedColor = colorAnim.interpolate({
    inputRange: [0, 1],
    outputRange: [color, darkenColor(color, darkenAmount)]
  });

  const animatedOpacity = colorAnim.interpolate({
    inputRange: [0, 1],
    outputRange: [opacityRange[1], opacityRange[0]] // fades as it expands
  });

  return (
    <Animated.View
      style={[
        styles.circle,
        {
          width: size,
          height: size,
          borderRadius: size / 2,
          backgroundColor: animatedColor,
          transform: [{ scale: scaleAnim }],
          opacity: animatedOpacity,
        },
      ]}
    />
  );
};

// Helper to darken an RGB or named color
function darkenColor(baseColor, amount = 0.2) {
  const match = baseColor.match(/\d+/g);
  if (!match) return baseColor;
  const [r, g, b] = match.map(Number);

  const newR = Math.max(0, r * (1 - amount));
  const newG = Math.max(0, g * (1 - amount));
  const newB = Math.max(0, b * (1 - amount));

  return `rgb(${newR}, ${newG}, ${newB})`;
}

export default PulseIndicator;

const styles = StyleSheet.create({
  circle: {
    marginVertical: 5,
    justifyContent: "center",
    alignItems: "center",
  },
});
```

useBLE.jsx

```
import { useState, useEffect, useRef, useCallback } from "react";
import { Alert, Vibration } from "react-native";
import { BleManager } from "react-native-ble-plx";
import { Buffer } from "buffer";

const SYNKLINX_SERVICE = "e5df6019-cf42-49f6-a418-346db96363f6";
const HEART_RATE_CHAR = "3c1f4fe4-c7ce-4c09-a38a-ba166fce06c6";
const O2_CHAR = "9749ccd9-940e-4ad6-bce9-d246a8d30dca";
const TEMP_CHAR = "69b82322-1c56-423e-820e-eb08c3f030d4";

export default function useBLE() {
  // BLE Manager
```

```
const manager = useRef(new BleManager()).current;

// States
const [devices, setDevices] = useState([]);
const devicesRef = useRef([]);
const [connectedDevice, setConnectedDevice] = useState(null);
const [heartRate, setHeartRate] = useState(0);
const [O2Sat, setO2Sat] = useState(0);
const [temperature, setTemperature] = useState(0);
const [scanning, setScanning] = useState(false);

// Refs
const subs = useRef([]);
const retryTimeout = useRef(null);
const disconnectDebounce = useRef(false);
const scanLock = useRef(false);

// Cleanup
useEffect(() => {
  return () => {
    subs.current.forEach((s) => { try { s?.remove?.(); } catch {} });
    if (retryTimeout.current) clearTimeout(retryTimeout.current);
    try { manager.destroy(); } catch {}
  };
}, []);

// Parsing
const parseUInt8 = (char, setter, min = 0, max = 255) => {
  if (!char?.value) return;
  const buf = Buffer.from(char.value, "base64");
  if (buf.length < 1) return;
  const value = buf.readUInt8(0);
  if (value >= min && value <= max) setter(value);
};

const parseInt16Temp = (char, setter, min = 30, max = 45) => {
  if (!char?.value) return;
```

```
    const buf = Buffer.from(char.value, "base64");
    if (buf.length < 2) return;
    const value = buf.readInt16LE(0) / 100;
    if (value >= min && value <= max) setter(value);
  };

  // Streaming
  const startStreaming = useCallback((device) => {
    subs.current.forEach((s) => { try { s?.remove?.(); } catch {} });
    subs.current = [];

    try {
      const hrSub = device.monitorCharacteristicForService(
        SYNKLINX_SERVICE,
        HEART_RATE_CHAR,
        (err, char) => { if (!err) parseUInt8(char, setHeartRate, 40, 200); }
      );

      const o2Sub = device.monitorCharacteristicForService(
        SYNKLINX_SERVICE,
        O2_CHAR,
        (err, char) => { if (!err) parseUInt8(char, setO2Sat, 50, 100); }
      );

      const tempSub = device.monitorCharacteristicForService(
        SYNKLINX_SERVICE,
        TEMP_CHAR,
        (err, char) => { if (!err) parseInt16Temp(char, setTemperature, 30, 45);
}
      );

      subs.current.push(hrSub, o2Sub, tempSub);
    } catch (err) {
      console.log("Streaming error:", err);
      Alert.alert("Streaming Error", err.message || "Could not start
streaming.");
    }
  }, []);

  // Scan
  const scanForPeripherals = useCallback((options = { autoRetry: true, timeout:
8000 }) => {
    if (scanning || scanLock.current) return;
    scanLock.current = true;

    setDevices([]);
    devicesRef.current = [];
    setScanning(true);

    manager.startDeviceScan(null, null, (error, device) => {
      if (error) {
        console.log("Scan error:", error);
        setScanning(false);
        scanLock.current = false;
        return;
      }

      if (!device || !device.name) return;

      const name = device.name.trim().toLowerCase();
      if (name.includes("synklinx")) {
        if (!devicesRef.current.some(d => d.id === device.id)) {
          devicesRef.current.push(device);
          setDevices(prev => [...prev.filter(d => d.id !== device.id), { id:
device.id, name: device.name }]);
        }
      }
    });

    retryTimeout.current = setTimeout(() => {
      manager.stopDeviceScan();
      setScanning(false);
      scanLock.current = false;
      if (options.autoRetry && devicesRef.current.length === 0) {
        console.log("No devices found, retrying scan...");
        scanForPeripherals(options);
      }
    }, options.timeout);
  }, [scanning, manager]);

  const stopScanning = useCallback(() => {
    if (retryTimeout.current) clearTimeout(retryTimeout.current);
    try { manager.stopDeviceScan(); } catch {}
    setScanning(false);
    scanLock.current = false;
  }, [manager]);

  // Connect
  const connectToDevice = useCallback(async (device) => {
    stopScanning();
    const realDevice = devicesRef.current.find(d => d.id === device.id);
    if (!realDevice) return;

    try {
      const conn = await manager.connectToDevice(realDevice.id, { autoConnect:
false });
      await conn.discoverAllServicesAndCharacteristics();
      await new Promise(r => setTimeout(r, 500));

      setConnectedDevice(conn);
      setHeartRate(0);
      setO2Sat(0);
      setTemperature(0);

      await startStreaming(conn);

      const dis = manager.onDeviceDisconnected(realDevice.id, () =>
handleDisconnect(realDevice.id));
      subs.current.push(dis);
    } catch (err) {
      console.log("Connection failed:", err);
      Alert.alert("Error", "Failed to connect to SynkLinx device.");
    }
  }, [manager, stopScanning, startStreaming]);

  // Disconnect
  const handleDisconnect = useCallback((deviceId) => {
    if (disconnectDebounce.current) return;
    disconnectDebounce.current = true;

    Vibration.vibrate([500, 200, 500]);
    Alert.alert("Disconnected", "SynkLinx device lost connection.");

    setConnectedDevice(null);
    setHeartRate(0);
    setO2Sat(0);
    setTemperature(0);

    subs.current.forEach(s => { try { s?.remove?.(); } catch {} });
    subs.current = [];

    setTimeout(() => {
      disconnectDebounce.current = false;
      scanForPeripherals();
    }, 2000);
  }, [scanForPeripherals]);

  const disconnect = useCallback(async () => {
    subs.current.forEach(s => { try { s?.remove?.(); } catch {} });
    subs.current = [];

    if (connectedDevice) {
      try { await manager.cancelDeviceConnection(connectedDevice.id); } catch {}
      setConnectedDevice(null);
      setHeartRate(0);
      setO2Sat(0);
      setTemperature(0);
    }
  }, [connectedDevice, manager]);

  return {
    scanForPeripherals,
    stopScanning,
    connectToDevice,
    disconnect,
    devices,
    connectedDevice,
    heartRate,
    O2Sat,
    temperature,
    scanning,
  };
}
```

BLEProvider.jsx

```
import React, { createContext, useContext } from "react";
import useBLE from "./useBLE"; // your existing hook

const BLEContext = createContext(null);

export const BLEProvider = ({ children }) => {
  const ble = useBLE();
  return <BLEContext.Provider value={ble}>{children}</BLEContext.Provider>;
};
export const useBLEContext = () => {
  const context = useContext(BLEContext);
  if (!context) throw new Error("useBLEContext must be used within a
BLEProvider");
  return context; };
```