

Introduction:

Annually, roughly 32,000 Australians suffer cardiac arrest, 80% of those happen outside of a hospital and only 10% of those people survive. In many cases there are no noticeable warning signs and early detection of heart problems can save thousands of lives. With technology becoming more accurate and accessible, making a bracelet that monitors heart rate can assist in detecting anomalies and alert users to seek medical attention which can prevent serious medical emergencies at a much cheaper price than other products such as the Apple Watch or Fitbit.

Objective:

This project aims to design a wearable system that uses a pulse sensor and software integration to monitor heart rate, detect abnormal readings, and deliver real time health alerts through a mobile app. The goal is to assess whether low cost technology can provide reliable early detection of irregular heart rhythms.

Motivation:

I want to make an affordable heart rate tracking medical bracelet to help people stay safe, healthy and additionally catch heart problems early. Many people don't know they have a heart issue until it's too late. This device could alert them in time to get help and possibly save the lives of many people who would have died without the correct medical attention/advice.

Software/Programming languages used:

Arduino- C++ (Reads heart rate through the analog input and sends it over serial)

IDLE- *Python* (Reads the serial data and sends it through a WebSocket server)

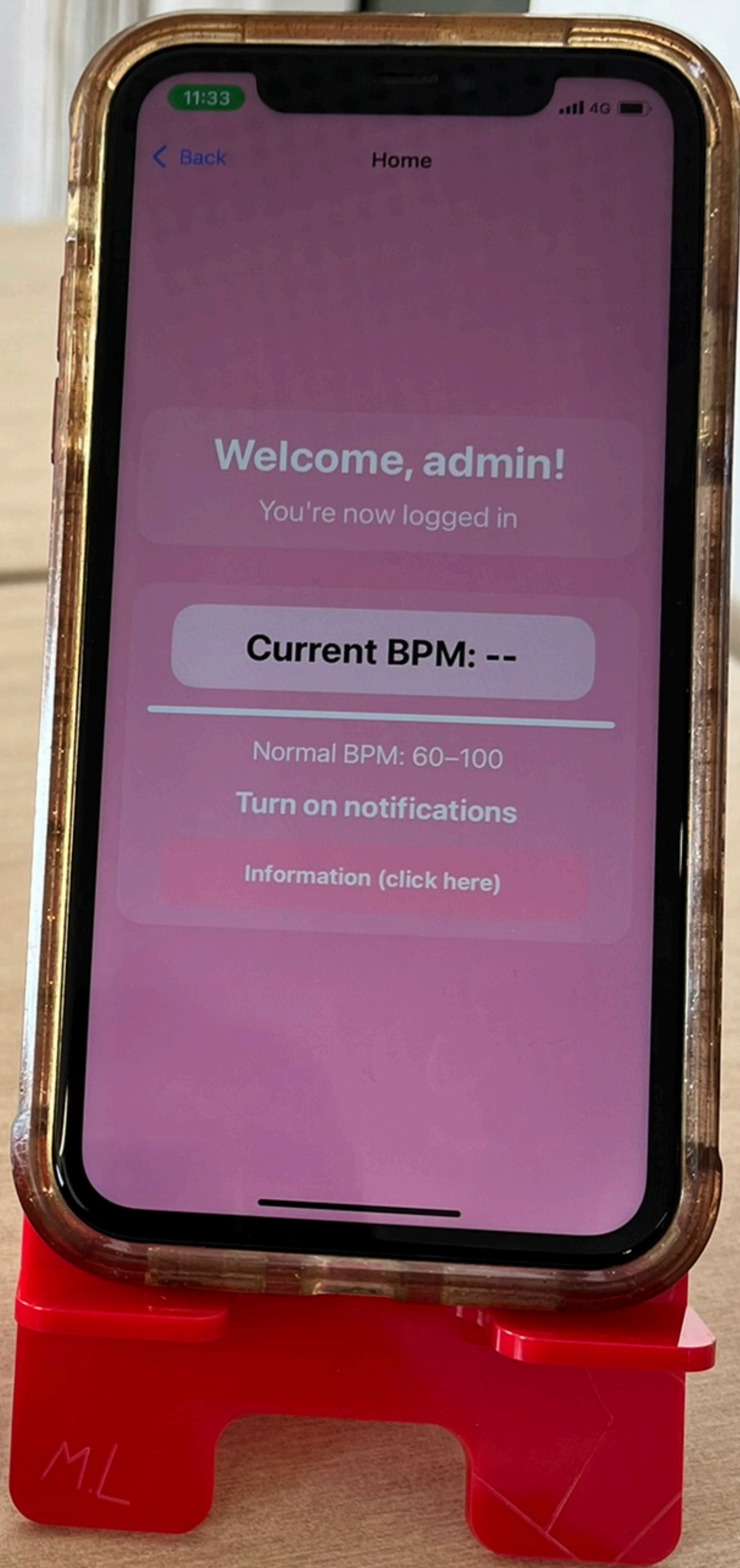
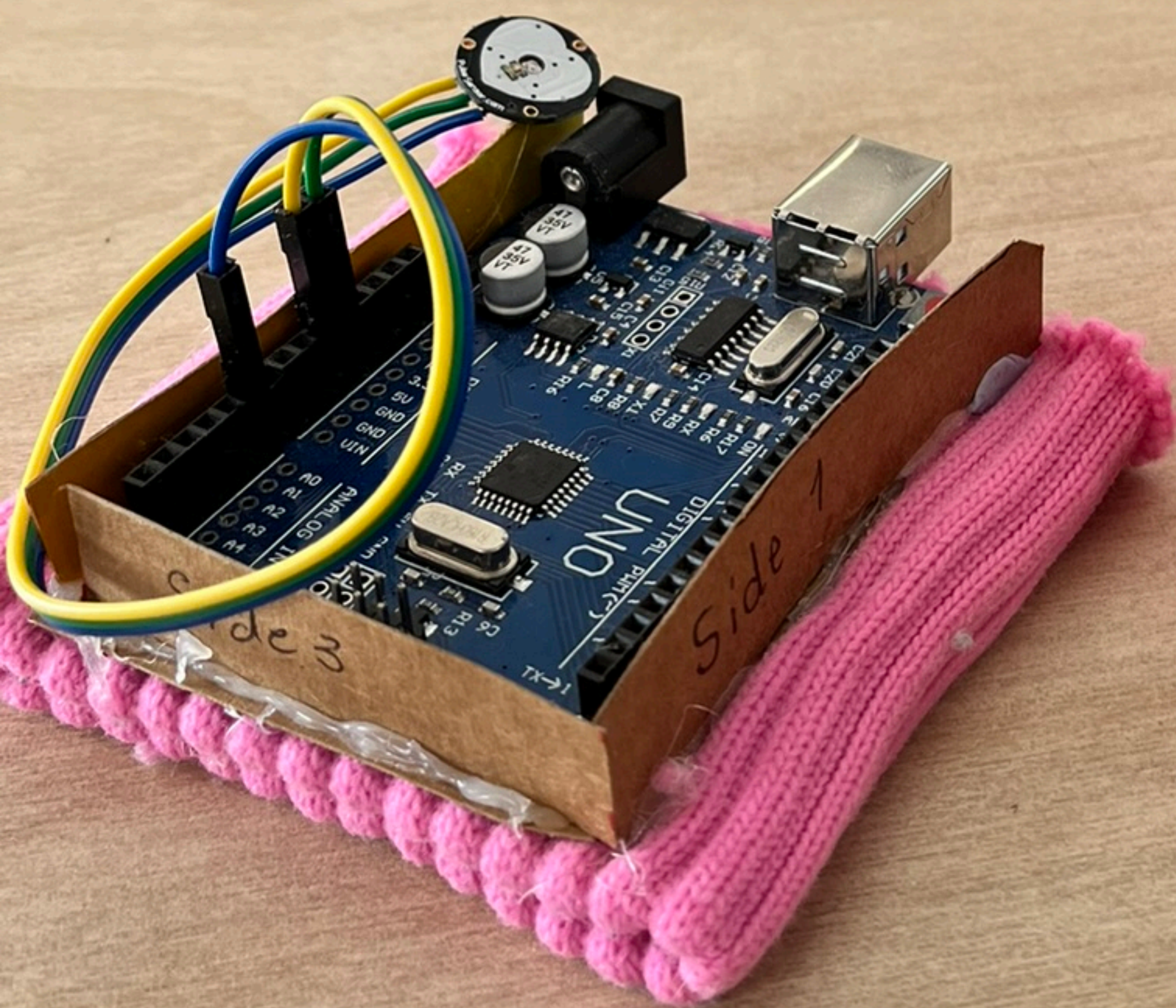
X Code- *Swift* (Receives data and outputs it for the users.)

Required parts and software to run the app:



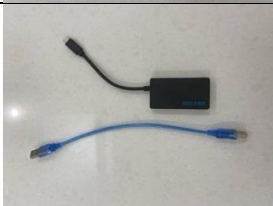


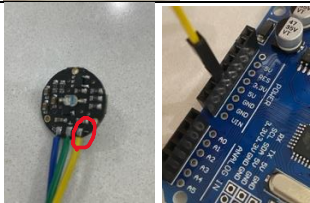
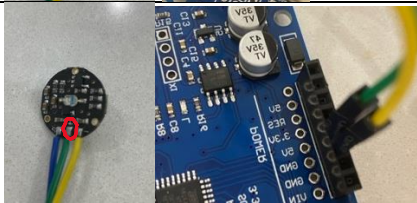
Python IDLE <https://www.python.org/downloads/>

Arduino <https://www.arduino.cc/en/software/>

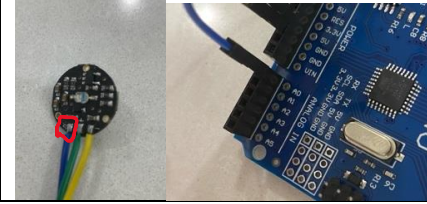
X Code <http://apps.apple.com/us/app/xcode/id497799835?mt=12/>



**This is just a prototype of what the bracelet's intended function is and not a final product.*

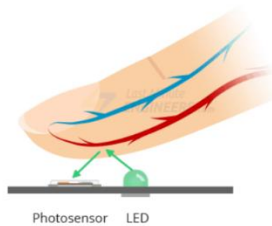
Part	Image
Circuit board (Arduino Uno)	
Pulse Sensor Analog Signal Output	
Cords (To connect to laptop without Bluetooth)	
Laptop (To run the code, collect data, and act as a server)	
Phone (to run the app)	*No photo
Wiring	Image
This sensor has three wires that need to be connected in specific places to work.	
The first wire has a (-) next to it. Which is the yellow wire. This one needs to be connected in GND which is short for ground. Which serves as a zero-voltage reference point in a circuit.	
The second wire has a (+) next to it. (Which is the green one) This wire requires to be connected in 5V which is a controlled 5-volt power source that can be used to power the board and external components.	

This last wire has an (S) next to it. (Which is the blue one) This wire connects to A0 which is the first analogue input pin meaning it can measure voltage between 0-5 volts on the Arduino UNO.

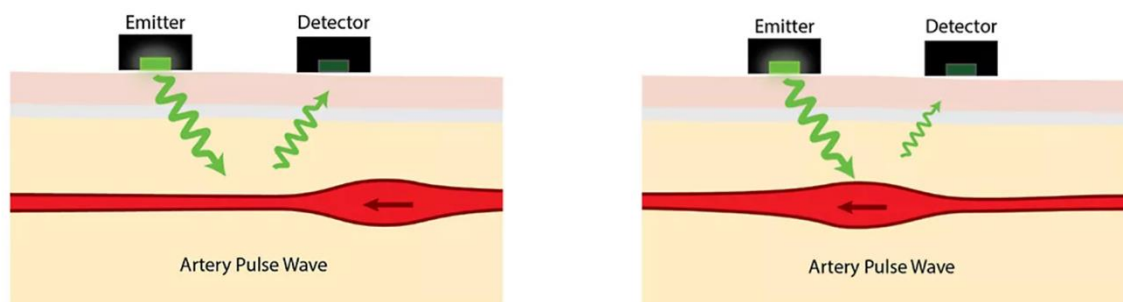


Photoplethysmography:

The pulse sensor used in this project is based on a technology called Photoplethysmography (PPG) which is a non-invasive optical method used to detect blood volume changes in the microvascular tissue.



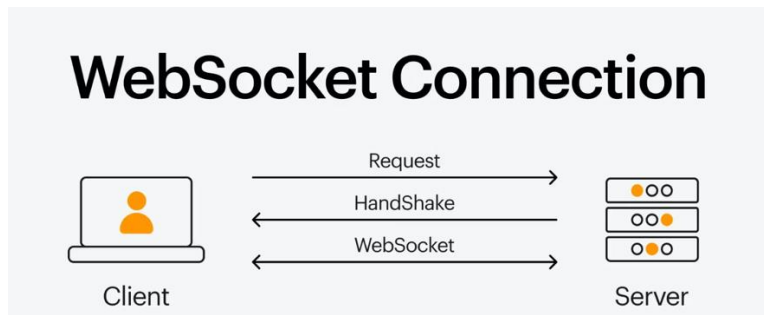
It works by shining a small light-emitting diode (LED) into the skin. As the heart pumps blood, the volume of blood in the capillaries changes with each heartbeat. The photodetector next to the LED measures how much light is absorbed or reflected by the blood. More blood means more light absorbed while less blood means more light reflected. PPG devices use green LEDs because green light is absorbed well by blood, especially in the skin's tiny blood vessels. This helps the device easily detect a heartbeat by measuring changes in blood flow. Also, green light works better on the wrist and gives a clearer signal, even when you move.



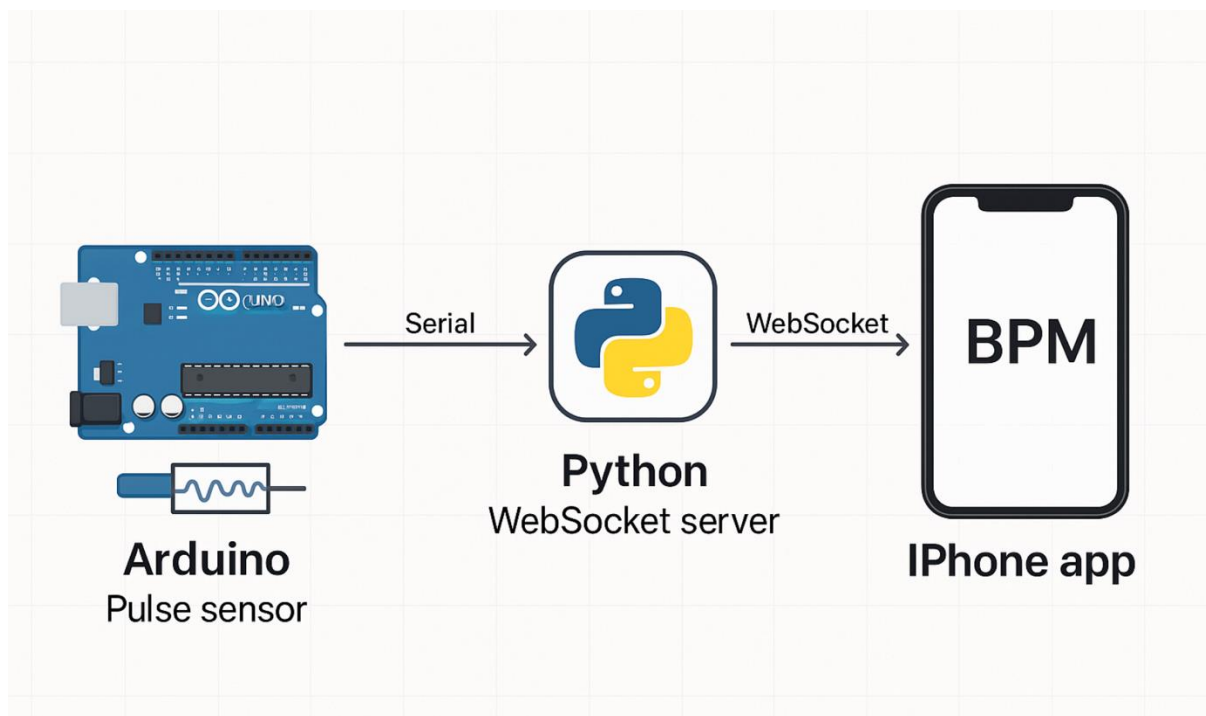
These changes produce a waveform that matches the rhythm of your heartbeat. The Arduino reads this signal as an analog signal, and with proper processing the time between each pulse (Inter-Beat Interval) can be used to calculate Beats Per Minute (BPM). This type of sensor is used in smartwatches, medical monitors, and fitness devices, and is effective because it's simple and inexpensive.

WebSocket:

From the sensor to the app all of this functions through the use of a WebSocket. A WebSocket is a way for the client and the server to communicate to each other in real time. Unlike a normal Hypertext Transfer Protocol (HTTP) which tends to be short-lived and one way, WebSockets keep the connection open to send or receive messages or inputs from both sides as long as the program is running.



In this scenario the Arduino reads the pulse sensor and sends the value over serial (USB). Then the python script running on the computer reads the serial data and acts as a WebSocket server. The X-Code app connects to the WebSocket server using Swifts *URLSessionWebSocketTask*. So, when the Arduino sends over a sensor value, the Python server relays the data to the app in real time through WebSocket. Then the app will display and trigger alerts if the detected BPM is abnormal.



*Image generated by ChatGPT

Testing:

The C++ code for the sensor was developed to verify the hardware is functioning properly. It began with a simple program designed to read the sensor's output. The output reflects what the sensor is detecting in real time. This basic code provides a straightforward way to confirm that the sensor is working correctly.

Code is below

// means annotations

```
#define sensor A0 // Pulse sensor is connected to the analog pin A0 of the Arduino

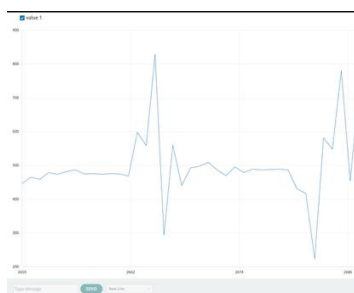
void setup() {
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate for data transfer
  delay(1000); // Wait for 1000 milliseconds (1 second) to ensure the Serial Plotter is ready
}

void loop() {
  int Svalue = analogRead(sensor); // Read the current analog value from the pulse sensor on pin A0
  Serial.println(Svalue); // Print the actual sensor value (between 0 and 1023) to the Serial Plotter
  delay(500); // Wait for 500 milliseconds before reading the sensor value again
}
```

This code allows the monitoring of raw analog values from the pulse sensor helping observe the sensor's response to a pulse. By these values being displayed in the Serial plotter, it makes it easy to analyse the sensor's behaviour in real time. The delay at the end of the code ensures that the data is not printed too quickly.

Example output:

**The output is not perfect due to background noise the sensor could pick up.*



Converting Sensor Value to Heart Rate:



Prize Winner

Programming, Apps & Robotics Year 9-10

Madi Lucey

**Pembroke School - Middle
School**



At the moment there is a code that gives the sensor value but not the heart rate. To convert the sensor value to a BPM value this equation has to be plugged into the code.

$$\text{BPM} = \frac{60,000}{\text{IBI (ms)}}$$

The IBI is the time in milliseconds between two heart beats and there are 60,000 milliseconds in a minute. So, by dividing 60,000 by the time between two heart beats you would get the average beats per minute.

Updated Arduino code with BPM conversions:

```
#define sensor A0 // Pulse sensor connected to analog pin A0

// Variables for beat detection
int threshold = 550; // Signal level threshold to detect a pulse
int sensorValue = 0; // Current analog reading from the pulse sensor
int lastSensorValue = 0; // Previous sensor reading, used to detect rising edges
bool pulseDetected = false; // Flag to make sure each pulse is counted once

unsigned long lastBeatTime = 0; // Timestamp of the last detected beat in milliseconds
unsigned long currentTime = 0; // Current time (in milliseconds)
int bpm = 0; // Calculated Beats Per Minute (BPM)

void setup() {
  Serial.begin(9600); // Initialize serial communication at 9600 baud for debugging and output
  delay(1000); // Short delay to allow the sensor to stabilize
}

void loop() {
  sensorValue = analogRead(sensor); // Read the current sensor value from analog pin A0

  currentTime = millis(); // Get the current elapsed time since program started in milliseconds

  // Detect rising edge when sensor signal crosses the threshold from below (start of a heartbeat)
  if (sensorValue > threshold && lastSensorValue <= threshold && !pulseDetected) {
    // Calculate the time interval between this beat and the previous beat
    unsigned long beatInterval = currentTime - lastBeatTime;
    lastBeatTime = currentTime; // Update last beat timestamp to now

    // Filter out unrealistic intervals to avoid false BPM calculations
    if (beatInterval > 300 && beatInterval < 2000) {
      bpm = 60000 / beatInterval; // Convert interval (ms) to beats per minute
    }

    pulseDetected = true; // Mark pulse as detected to avoid multiple counts for same beat
  }
}
```

```

// Reset pulseDetected flag when sensor signal drops below threshold (end of heartbeat)
if (sensorValue < threshold) {
    pulseDetected = false;
}

lastSensorValue = sensorValue; // Store current sensor value for next loop iteration

Serial.println(bpm); // Output BPM value to serial monitor

delay(100); // Delay to read sensor approximately 10 times per second (for better time resolution)
}

```

Python Server: # means annotations

The purpose of the python server is to act as a bridge between the Arduino and X-Code app through serial port.

Code below:

```

import asyncio # Imports Python's asyncio library for asynchronous I/O operations
import serial # Imports pySerial to communicate with serial ports (Arduino)
import websockets # Imports the websockets library for handling WebSocket connections

SERIAL_PORT = '/dev/cu.usbserial-120' # The serial port connected to the Arduino
BAUDRATE = 9600 # The baud rate for serial communication
PORT = 6789 # Port number for the WebSocket server

# Coroutine to handle data from serial to websocket
async def serial_to_websocket(websocket, path):
    print(f"Client connected: {path}") # Logs the WebSocket client connection

    try:
        ser = serial.Serial(SERIAL_PORT, BAUDRATE, timeout=1) # Open the serial port

        print(f"Serial port opened: {SERIAL_PORT}")

        while True:
            if ser.in_waiting > 0: # If data is waiting in the serial buffer:
                line = ser.readline().decode('utf-8').strip() # Read and decode the line
                await websocket.send(line) # Send the line to the WebSocket client
                await asyncio.sleep(0.1) # Small delay to avoid hogging CPU
    except Exception as e:

```

```

    print(f"Error: {e}") # Print any errors that occur

finally:

    ser.close() # Ensure the serial port is closed

# Main coroutine to start the WebSocket server

async def main():

    async with websockets.serve(serial_to_websocket, "0.0.0.0", PORT): # Bind server to all interfaces

        print(f"WebSocket server running on port {PORT}")

        await asyncio.Future() # Run forever (acts like while True but async-safe)

# Run the main function if the script is executed directly

if __name__ == "__main__":

    asyncio.run(main()) # Launch the asyncio event loop and run the main function

```

X-Code:

After the code for the hardware to function is developed, I began creating the app. To start I made a simplistic login page with the help of books and YouTube tutorials (found in the sources.) The login page leads the user to the main app after correct credentials are put in.

Login Page (ContentView) // means annotations

```

import SwiftUI // Imports the SwiftUI framework for building user interfaces.

struct ContentView: View { // Defines the main view of the application
    // State variables that SwiftUI monitors for changes
    @State private var username: String = "" // Holds the username
    @State private var password: String = "" // Holds the password
    @State private var isLoggedIn = false // Tracks whether the user is successfully logged in
    @State private var showAlert = false // Controls the display of the alert dialog
    @State private var alertMessage = "" // Message to display in the alert

    var body: some View {
        NavigationStack { // A container that manages navigation between views
            VStack(spacing: 20) { // Vertically stacks elements with 20 points spacing

                // The title of the login page
                Text("Login")
                    .font(.largeTitle) // Sets the font size to large
                    .fontWeight(.bold) // Makes the text bold

                // Username input field
                TextField("Username", text: $username) // A text input field connected to the username variable
                    .autocapitalization(.none) // Prevents auto-capitalization
                    .padding() // Adds padding inside the text field
                    .background(Color(.secondarySystemBackground)) // Makes the background light grey
                    .cornerRadius(8) // Rounds the corners of the box
            }
        }
    }
}

```

```

// Password input field
SecureField("Password", text: $password) // A secure text field that hides the input
    .padding()
    .background(Color(.secondarySystemBackground))
    .cornerRadius(8)

// Login button
Button(action: {
    loginUser(username: username, password: password) // Calls loginUser when clicked
}) {
    Text("Enter")
        .frame(maxWidth: .infinity) // Makes the button expand horizontally
        .padding()
        .background(Color(.sRGB, red: 1.0, green: 0.62, blue: 0.72, opacity: 1.0)) // light pink
        .foregroundColor(.white) // White text color
        .cornerRadius(8)
}

// Hidden navigation link that activates when isLoggedIn becomes true
NavigationLink(
    destination: DashboardView(), // Navigates to the DashboardView upon a successful login
    isActive: $isLoggedIn,
    label: { EmptyView() } // Doesn't display a visual element
)

Spacer() // Pushes everything up to the top
}
.padding() // Adds padding around the whole VStack
.alert(isPresented: $showingAlert) { // Shows an alert if showingAlert is true
    Alert(
        title: Text("Login Failed"),
        message: Text(alertMessage),
        dismissButton: .default(Text("OK"))
    )
}
}
}

// Function to handle login logic
func loginUser(username: String, password: String) {
    if username == "admin" && password == "test123" {
        isLoggedIn = true // Navigates to DashboardView if credentials match
    } else {
        alertMessage = "Invalid username or password." // Shows error message
        showingAlert = true // Triggers alert display
    }
}
}
}

```

Main App (Dashboard View)

```

import SwiftUI // Imports SwiftUI for building UI components
import UserNotifications // Imports framework for handling notification permissions

// Main view for displaying a heart rate monitoring dashboard
struct DashboardView: View {
    // Observed object that manages WebSocket connection and BPM data updates
    @StateObject private var socketManager = WebSocketManager()

    var body: some View {
        VStack {
            // Sets a pink background color that covers the entire screen
            Color(red: 1.0, green: 0.62, blue: 0.72)
                .ignoresSafeArea() // Ensures background goes under system UI elements

            VStack(spacing: 30) { // Vertical layout with spacing between components
                Spacer() // Pushes the content downward for layout balance

                // Welcome header section
                VStack(spacing: 10) {
                    Text("Welcome, admin!") // Greeting text
                        .font(.largeTitle)
                        .fontWeight(.bold)
                        .foregroundColor(.white) // White text for visibility

                    Text("You're now logged in") // Subtext under greeting
                        .font(.title2)

```

```

        .foregroundColor(.white.opacity(0.8)) // Slightly transparent
    }
    .padding() // Adds padding inside the greeting box
    .frame(maxWidth: .infinity) // Stretches full width
    .background(RoundedRectangle(cornerRadius: 20).fill(Color.white.opacity(0.1))) // Semi-transparent background with rounded corners
    .padding(.horizontal) // Padding on the sides

// BPM and health suggestion section
VStack(spacing: 15) {
    // Displays the current BPM value from WebSocket
    Text("Current BPM: \${socketManager.bpm}")
        .font(.title)
        .foregroundColor(.black)
        .fontWeight(.bold)
        .padding()
        .frame(maxWidth: .infinity)
        .background(RoundedRectangle(cornerRadius: 20).fill(Color.white.opacity(0.5))) // Light background behind BPM
        .padding(.horizontal)

    // Horizontal visual separator
    RoundedRectangle(cornerRadius: 25)
        .fill(Color.white)
        .frame(height: 5)

    Text("Normal BPM: 60–100") // Reference range
        .font(.title3)
        .foregroundColor(.white)

    // Optional health advice based on BPM value
    if let bpmInt = Int(socketManager.bpm) {
        if bpmInt < 40 {
            // If BPM is below 40
            Text("Low Heart Rate") // Warning messages for low BPM
            Text("Suggestion: Sit or lay down and drink water")
            Text("If still feeling dizzy consult a medical professional.")
                .font(.title3)
                .foregroundColor(.red)
                .fontWeight(.semibold)
        } else if bpmInt > 110 {
            // If BPM is above 110
            Text("High Heart Rate") // Warning message for high BPM
            Text("Suggestion: Sit or lay down")
            Text("If still experiencing chest pain or shortness of breath consult a medical professional")
                .font(.title3)
                .foregroundColor(.red)
                .fontWeight(.semibold)
        } else {
            // If BPM is in a normal range
            Text("BPM should be on the lower side while resting.")
            Text("And higher while exercising")
                .font(.caption)
                .foregroundColor(.white)
        }
    }

    // Notification prompt text
    Text("Turn on notifications")
        .font(.title2)
        .fontWeight(.bold)
        .foregroundColor(.white)

    // Navigation to an additional info view
    NavigationLink(destination: InfoView()) {
        Text("Information (click here)")
            .padding()
            .frame(maxWidth: .infinity)
            .background(Color(red: 2.0, green: 0.62, blue: 0.72)) // Brighter pink
            .foregroundColor(.white)
            .fontWeight(.bold)
            .cornerRadius(10)
    }
    .padding(.horizontal)
}
.padding() // Padding around the entire BPM box
.frame(maxWidth: .infinity)
.background(RoundedRectangle(cornerRadius: 20).fill(Color.white.opacity(0.1))) // Semi-transparent rounded background
.padding(.horizontal)

Spacer() // Pushes everything upward for symmetry

```

```

    }
}
.navigationTitle("Home") // Title in the navigation bar
.onAppear {
    // Requests notification permissions on first load
    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound]) { granted, _ in
        print("Notification permission: \(granted)")
    }
    socketManager.connect() // Connect to WebSocket for real-time updates
}
.onDisappear {
    socketManager.disconnect() // Clean up connection when view disappears
}
}
}
}

```

WebSocket Manager:

```

import Foundation
import UserNotifications // For sending local notifications

// ObservableObject allows this class to be used with SwiftUI views that observe it
class WebSocketManager: ObservableObject {
    @Published var bpm: String = "--" // Publishes changes to the heart rate string to update UI

    private var task: URLSessionWebSocketTask? // Manages the WebSocket connection
    private var isAbnormal = false // Tracks whether the BPM is outside the normal range
    private var alertTimer: Timer? // Timer that periodically checks for abnormal BPM

    // Establishes a WebSocket connection
    func connect() {
        // Ensure the WebSocket URL is valid
        guard let url = URL(string: "ws://*****:6789") else { // censored IP for security reasons
            print("Invalid WebSocket URL")
            return
        }

        // Create and start a WebSocket task
        let session = URLSession(configuration: .default)
        task = session.webSocketTask(with: url)
        task?.resume()

        print("WebSocket connected to \(url)")

        startAlertTimer() // Start periodic abnormality check
        receive() // Begin listening for incoming messages
    }

    // Disconnects the WebSocket and stops the timer
    func disconnect() {
        task?.cancel(with: .goingAway, reason: nil) // Close the connection
        alertTimer?.invalidate() // Stop the alert timer
        print("WebSocket disconnected")
    }

    // Continuously receives messages from the WebSocket server
    private func receive() {
        task?.receive { [weak self] result in
            DispatchQueue.main.async {
                switch result {
                case .success(let message):
                    switch message {
                    case .string(let text): // Received a text message
                        print("Received text: \(text)")
                        // Show "Connecting..." while no real data is coming in
                        self?.bpm = text == "0" ? "Connecting..." : text
                        self?.checkAbnormal(bpm: text) // Check if the BPM is abnormal
                    case .data(let data): // Received binary data (not used)
                        print("Received binary data: \(data)")
                    @unknown default: // Fallback for future message types
                        print("Unknown message type received")
                    }
                case .failure(let error): // WebSocket error
                    print("WebSocket error: \(error)")
                }
            }
        }

        // Recursively listen for the next message
        self?.receive()
    }
}

```

```

    }
}
}

// Determines whether the current BPM is abnormal (too low or too high)
private func checkAbnormal(bpm: String) {
    guard let value = Int(bpm) else { return }
    isAbnormal = value < 40 || value > 100 // Define abnormal range
}

// Starts a timer that checks the BPM every 30 seconds and sends a notification if abnormal
private func startAlertTimer() {
    alertTimer = Timer.scheduledTimer(withTimeInterval: 30.0, repeats: true) { [weak self] _ in
        guard let self = self else { return }
        if self.isAbnormal, let currentBpm = Int(self.bpm) {
            self.sendNotification(title: "BPM Alert", body: "Heart rate is \(currentBpm) BPM") // Notification message
        }
    }
}

// Sends a local notification using the UserNotifications framework
private func sendNotification(title: String, body: String) {
    let content = UNMutableNotificationContent()
    content.title = title
    content.body = body
    content.sound = .default // Use default system sound

    // Trigger notification immediately
    let request = UNNotificationRequest(
        identifier: UUID().uuidString,
        content: content,
        trigger: nil
    )

    // Schedule the notification
    UNUserNotificationCenter.current().add(request) { error in
        if let error = error {
            print("Notification error: \(error)")
        } else {
            print("Notification scheduled")
        }
    }
}
}
}

```

Information Page (infoview)

import SwiftUI // Imports the SwiftUI framework for building user interfaces.

```

struct InfoView: View { // Main view for the information page
    @Environment(\.presentationMode) var presentationMode

```

```

    var body: some View {
        ZStack {
            Color(red: 1.0, green: 0.62, blue: 0.72) // pink colour
                .ignoresSafeArea() // makes the whole background pink

            VStack(spacing: 20) {
                Text("About to BeatSync") // title
                    .font(.largeTitle) // font
                    .foregroundColor(.white) // text colour
                    .bold()
                    .padding()
            }
        }
    }
}

```

Text("Each year, approximately 32,000 Australians suffer a cardiac arrest. Alarminglly, 80% of these events occur outside of hospital environments, and only about 10% of those affected survive. In many cases, there are no warning signs, cardiac arrest can strike without notice. Early detection of heart irregularities is crucial and could save thousands of lives. As wearable technology becomes more accurate and accessible, it presents an opportunity to offer a life-saving tool at a fraction of the cost of premium products like the Apple Watch or Fitbit.") // information paragraph 1

```

        .multilineTextAlignment(.center) // aligns text to the centre
        .foregroundColor(.white)
        .padding()
        .frame(maxWidth: .infinity)
        .background(RoundedRectangle(cornerRadius: 20).fill(Color.white.opacity(0.1))) // Rounded box

```

Text("This project aims to design and build an affordable medical bracelet that connects to a mobile application. The bracelet will measure heart rate in real time, detect abnormalities, and provide immediate feedback to the user. By identifying unusual heart activity early, the device can prompt users to seek medical attention before the situation becomes critical. This proactive approach to heart health could significantly reduce emergency cases and improve survival rates.") // information paragraph 2

```

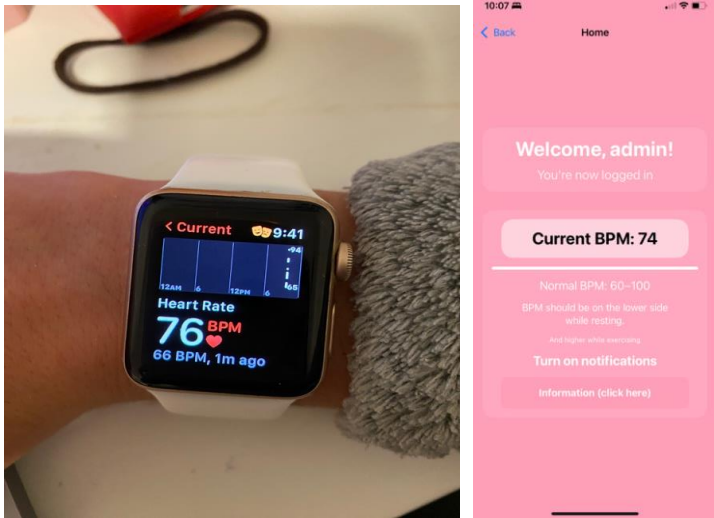
        .multilineTextAlignment(.center)

```

Testing:



Trials	Apple watch BPM	BeatSync BPM
1	78	72
2	76	74
3	86	99



The results show that BeatSync produced readings within a close range of the Apple Watch in two out of three trials, with a larger deviation in the third. This variation could be due to sensor noise, movement, or differences in signal processing. Overall, the results suggest that the prototype is capable of providing reasonably accurate BPM measurements, supporting its use as a low-cost heart monitoring solution.

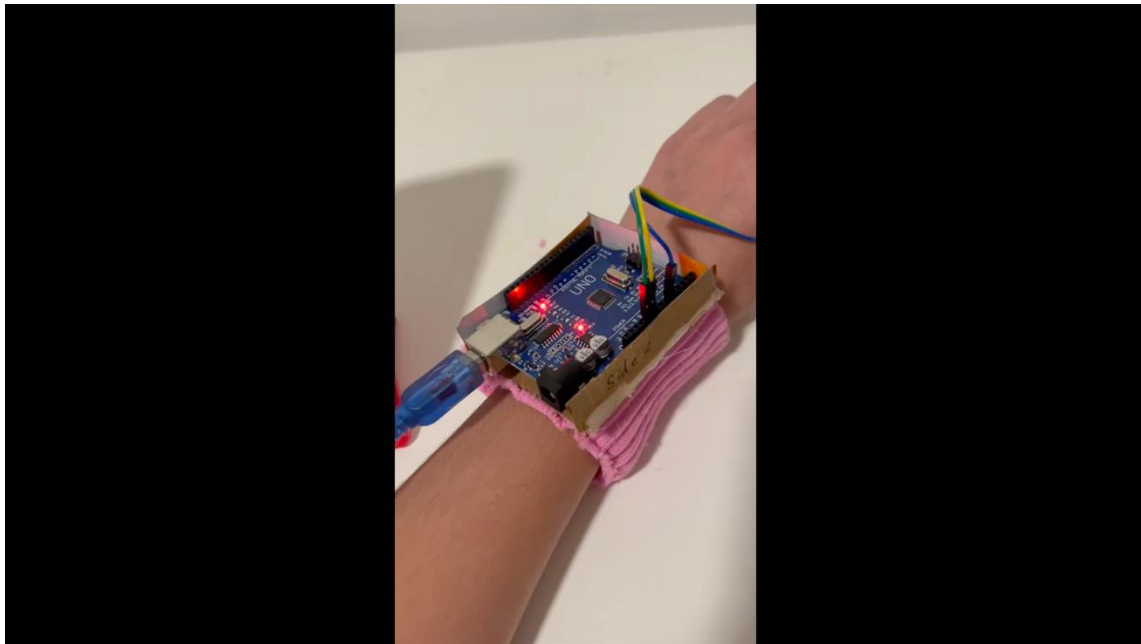
Summary:

This project, BeatSync is an engineered solution to the real-world problem of undetected cardiac events. It combines hardware and software systems to create an affordable, wearable heart rate monitor that detects abnormal pulse patterns and delivers real-time alerts through a mobile application. The system architecture consists of an Arduino-based pulse sensor, a Python WebSocket server for live data transmission, and a SwiftUI iOS app that processes and displays BPM with health recommendations.

The project applies core engineering principles problem-solving, iterative prototyping, system integration, and testing to deliver a reliable and scalable health technology solution. It showcases the use of threshold-based signal detection, serial communication, and cross-platform software development. *BeatSync* demonstrates strong computational thinking, user-centered design, and a focus on practical outcomes.

The code for the Arduino, Python WebSocket server, and SwiftUI app was scripted by me, without copying full templates or using AI. I used documentation, forums, video tutorials, and past experiences to learn how to build each part.

Video:



Sources

<https://www.heartfoundation.org.au/your-heart/cardiac-arrest>

<https://srituhobby.com/how-to-use-the-heart-pulse-sensor-with-arduino-heart-pulse-monitoring-system/>

[https://learn.sparkfun.com/tutorials/what-is-an-arduino/whats-on-the-board#:~:text=GND%20\(3\)%3A%20Short%20for,used%20to%20ground%20your%20circuit.](https://learn.sparkfun.com/tutorials/what-is-an-arduino/whats-on-the-board#:~:text=GND%20(3)%3A%20Short%20for,used%20to%20ground%20your%20circuit.)

<https://docs.arduino.cc/learn/electronics/power-pins/>

<https://forum.arduino.cc/t/variable-int-with-a0-instead-of-a-number/681553>

<https://lastminuteengineers.com/pulse-sensor-arduino-tutorial/>

<https://www.youtube.com/watch?v=ZOlXMxLRqc>

<https://www.youtube.com/watch?v=BLrHTHujPuw>.

Feiler, J. (2014). *iOS app development for dummies*. Hoboken, Nj: John Wiley & Sons.

<https://forum.arduino.cc/t/arduino-python-websocket/516034>

<https://websockets.readthedocs.io/en/10.1/>

<https://www.youtube.com/watch?v=ZzaPdXTrSb8>

<https://en.wikipedia.org/wiki/Baud>

<https://www.swiftypace.com/blog/swiftui-font-and-texts>

<https://www.youtube.com/watch?v=Fo1A36RsoCI>

<https://pulsesensor.com/pages/getting-advanced>

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

<https://www.ramotion.com/blog/what-is-websocket/>

<https://www.youtube.com/watch?v=8ARodQ4Wlf4>

<https://stackoverflow.com/questions/56443535/swiftui-text-alignment>

<https://en.wikipedia.org/wiki/Photoplethysmogram>

<https://www.ansys.com/blog/modeling-human-skin-and-optical-heart-rate-sensors>

<https://stackoverflow.com/questions/56571349/custom-back-button-for-navigationviews-navigation-bar-in-swiftui>