Use Cases & Prototype 1

StressLess Group Project

User Cases – Neha

We have previously stated that within our project, we are targeting students enrolled in higher education, as statistically they have higher stress levels. Within this subsection, I will expand on some situations in which this target user may use our product.

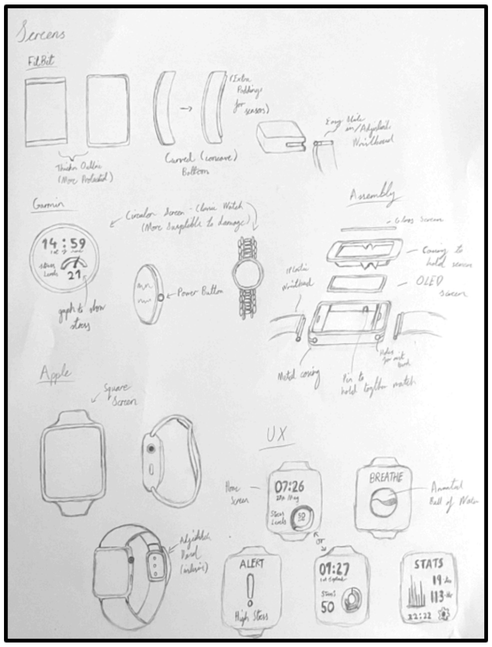
1) Exam periods and assessment periods are often time periods where students feel high levels of stress due to fears of helplessness and an inability to believe that their actions will not correlate with their academic performance, even if they prepare using revision or complete coursework. Therefore, we would suggest the use of a stress monitoring device, which can aid them with methods to reduce stress instantaneously. Reducing stress here is important, as it can prevent students from disengaging with work, which could further affect students' performance and grades. Our product's feature as a wearable means that it is accessible to the user at all times.
2) Situations such as lectures can present high-stress environments for students, as they may be overwhelmed with a large amount of information at once or the high noise levels this space tends to have (Transkriptor, 2023). Though students may feel the urge to leave due to their high stress levels, anxiety around missing content, mandatory attendance and social anxiety regarding leaving in front of a large group of people may prevent them from leaving. Here, it proves important to have an available and subtle aid to help deal with reducing stress. The placement of our wearable on the user's wrist is discreet, allowing use without students drawing attention to themselves.

User Review

Users have responded positively to the wearable bracelet, citing portability as a major advantage. Many people love how light and tiny the device is, making it easy to wear all day without feeling bulky or intrusive. The thin design means it won't get in the way of daily work, exercise, or sleep, allowing consumers to easily include it in their routine. For people who are frequently on the go, the bracelet provides a practical way to manage their health data—such as temperature or stress levels—without having to carry larger gadgets or continuously check their phone. Users have also stated that the unobtrusive form factor allows for simple wear in both informal and business settings. Overall, one of the wristband's most valuable characteristics is its portability, which allows users to stay connected to their well-being no matter where they are.

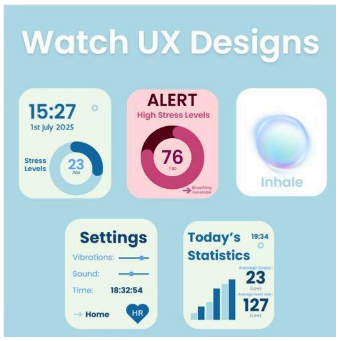
Prototype 1 – Nana

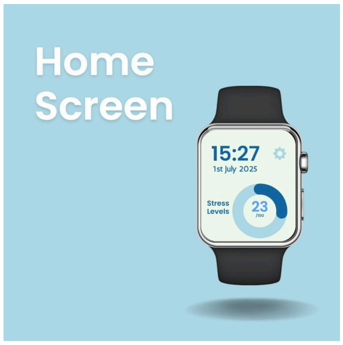
Our first prototype for the project was wireframes of the watch's user interface. We began by making sketches of the possible hardware we could use, starting with the screens that are shapes we might have: circular, square and rectangular; we picked square screens, as they seemed to be the most ergonomic and easily accessible to us. We then sketched out the assembly process of the watch if we were to use a square screen, like we wanted to have a rough idea of the components we'd need for the device.

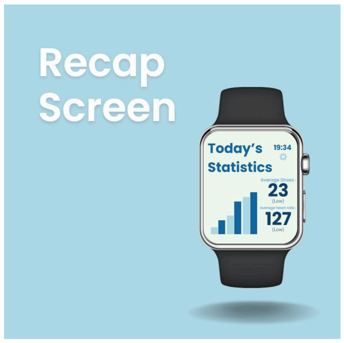


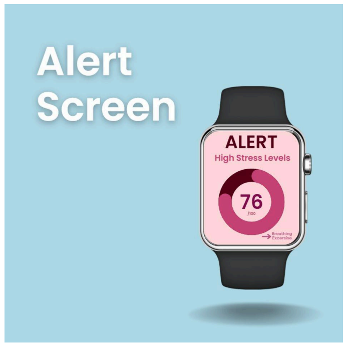
Next, we drew out sketches of the UX screens for possible home pages, an overview stats page, guided breathing, and an alert page. For all these designs, we tried to keep them as simplistic as possible as well as informative. We decided to recreate these sketches digitally so we could adjust the colour schemes and refine our ideas.

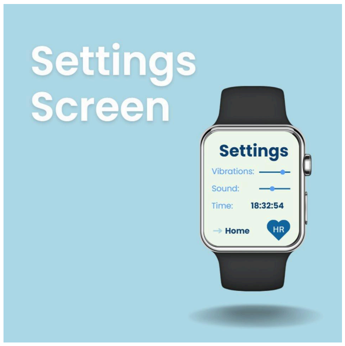
The first UX we made digitally was the **home screen**, following the blue colour scheme we'd be using throughout the project; we wanted the time to be displayed, as our wristband would take up the space where a watch would usually sit, and we did not want our users to be inconvenienced. We then obviously wanted the users' stress level to be readily accessible, so putting it on the main page would allow for this. Since we wanted our design to be minimalistic, we felt nothing else should be put on this page and left it at that. The next pages we made were the **recap** and **setting pages** that followed the same principle, only the necessary information was displayed. Our **alert screen** uses a red/pink colour screen to alert danger and a significant change, alongside the stress level. At the bottom of this page, there's an arrow to let the user go to the **guided breathing screen –** a small blue animated sphere with breathing indicators.

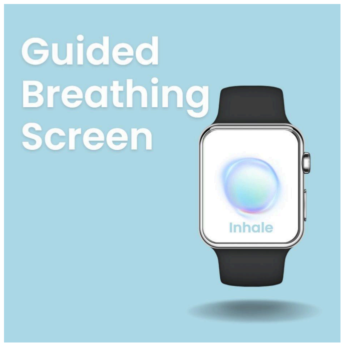


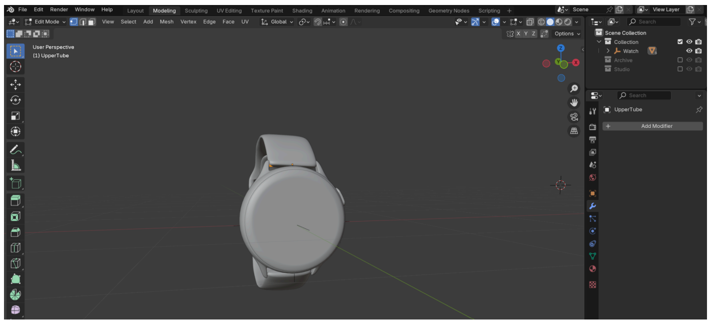


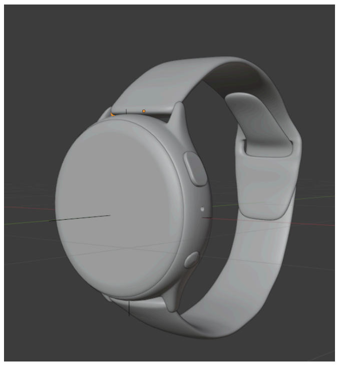


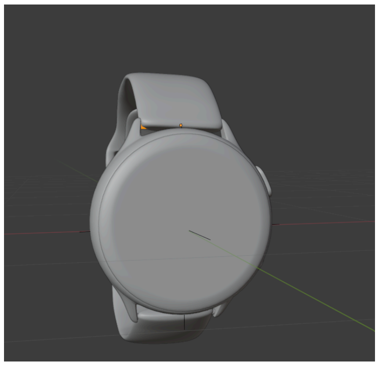






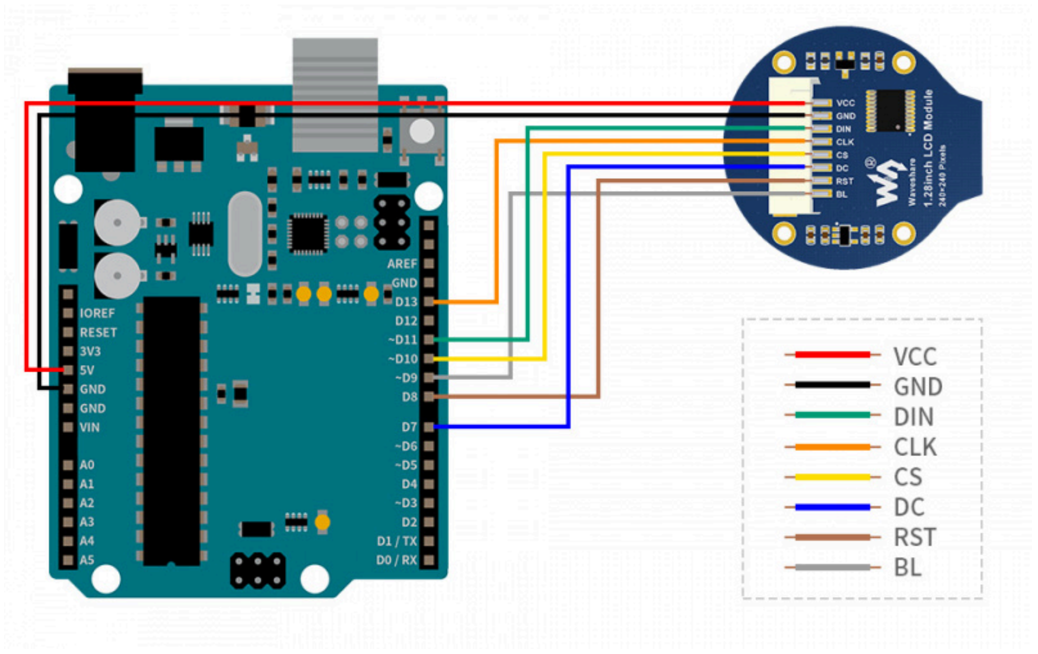






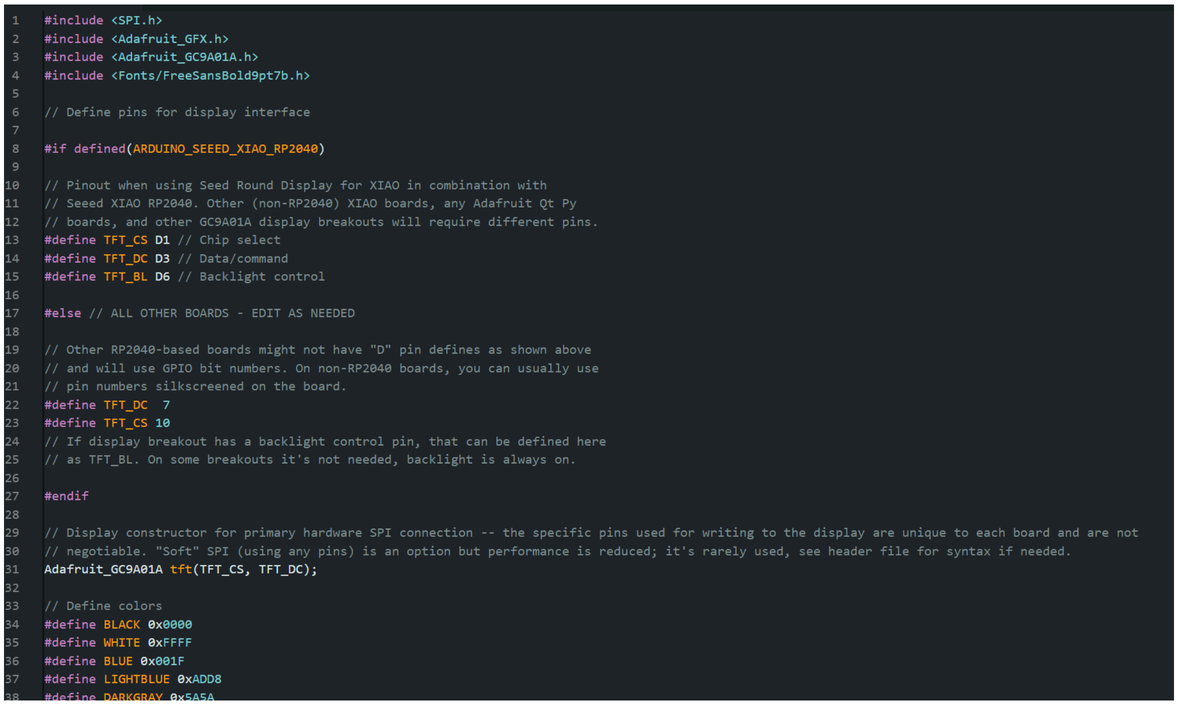
In the images above, we made Blender models of how we'd want the watch to look. We modelled with a circular screen because we had a circular OLED screen that was at our disposal, so we decided to use that rather than increasing expenses.

To make our prototype physical, we used the Waveshare 1.28-inch OLED screen and connected it to an Arduino Uno to try and get a version of the above wireframes onto the screen. We watched a tutorial (DroneBot, 2022) on how to connect the screen to the Uno and find out which libraries are necessary for coding our wireframe onto it. From the video and article, we found this circuit diagram on how to wire up the screen.

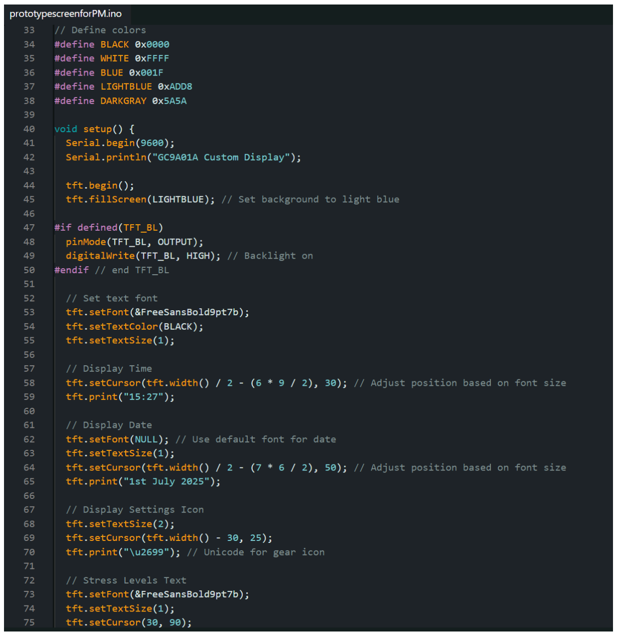


We then opened up our Arduino IDE and downloaded the Waveshare library for the 1.28-inch screen. Additionally, we also downloaded the Adafruit_GC9A01 from their Git repository for the code. Once we'd connected the libraries, we used an example code from the Waveshare library to test our screen and make sure that the screen was connected successfully. The test was successful, so we knew the screen was connected to the board correctly.

We then began to code for our home screen by importing the libraries SPI.h, Adafruit_GFX.h, Adafruit_GC9A01A.h, and Fonts/FreeSansBold9pt7b.h so we could reference them within the code. Then we defined the pin connected to the OLED screen and the colours that we use (using hex colours), and then called them with the code to say what colour we wanted for certain aspects of the screen, i.e., the background colour being a light blue and the font being black. Next, we turned the backlight of the screen on and picked the font for the on-screen text – Free Sans. The next section of code sets the positioning of the on-screen text, like the time and date are placed towards the side of the screen, like in the wireframe for the home page. Additionally, the code draws the circle that is used to display the users' stress levels, using the pre-existing function "tfs.drawcircle" and setting the radius of the circle to 30. The arc that actually indicates the stress level is drawn using the same function, but with the addition of being multiplied by the math functions sin() and cos() to get a proportionate arc.

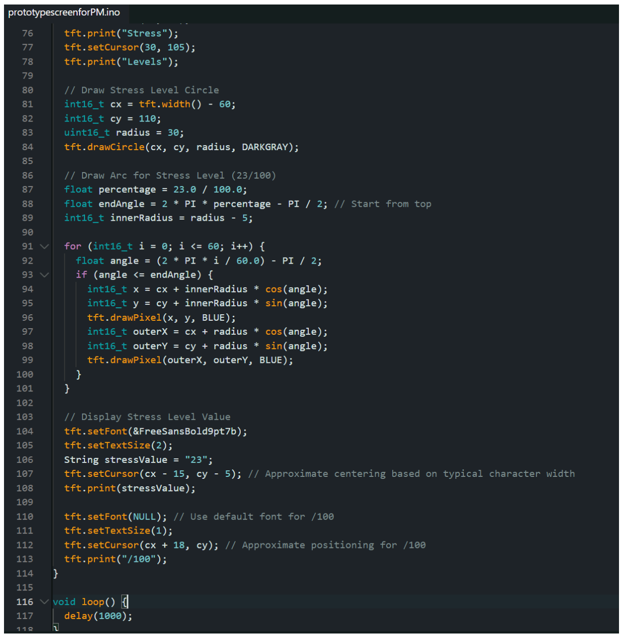


```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_GC9A01A.h>
#include <Fonts/FreeSansBold9pt7b.h>

// Define pins for display interface

#if defined(ARDUINO_SEEED_XIAO_RP2040)

// Pinout when using Seed Round Display for XIAO in combination with
// Seeed XIAO RP2040. Other (non-RP2040) XIAO boards, any Adafruit Qt Py
// boards, and other GC9A01A display breakouts will require different pins.
#define TFT_CS D1 // Chip select
#define TFT_DC D3 // Data/command
#define TFT_BL D6 // Backlight control

#else // ALL OTHER BOARDS - EDIT AS NEEDED

// Other RP2040-based boards might not have "D" pin defines as shown above
// and will use GPIO bit numbers. On non-RP2040 boards, you can usually use
// pin numbers silkscreened on the board.
#define TFT_DC  7
#define TFT_CS 10
// If display breakout has a backlight control pin, that can be defined here
// as TFT_BL. On some breakouts it's not needed, backlight is always on.

#endif

// Display constructor for primary hardware SPI connection -- the specific pins used for writing to the display are unique to each board and are not
// negotiable. "Soft" SPI (using any pins) is an option but performance is reduced; it's rarely used, see header file for syntax if needed.
Adafruit_GC9A01A tft(TFT_CS, TFT_DC);

// Define colors
#define BLACK 0x0000
#define WHITE 0xFFFF
#define BLUE 0x001F
#define LIGHTBLUE 0xADD8
#define DARKGRAY 0x5A5A
```



prototypescreenforPM.ino

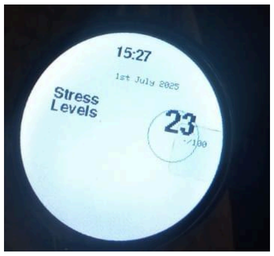
```
// Define colors
#define BLACK 0x0000
#define WHITE 0xFFFF
#define BLUE 0x001F
#define LIGHTBLUE 0xADD8
#define DARKGRAY 0x5A5A

void setup() {
  Serial.begin(9600);
  Serial.println("GC9A01A Custom Display");

  tft.begin();
  tft.fillScreen(LIGHTBLUE); // Set background to light blue

#if defined(TFT_BL)
  pinMode(TFT_BL, OUTPUT);
  digitalWrite(TFT_BL, HIGH); // Backlight on
#endif // end TFT_BL

  // Set text font
  tft.setFont(&FreeSansBold9pt7b);
  tft.setTextColor(BLACK);
  tft.setTextSize(1);

  // Display Time
  tft.setCursor(tft.width() / 2 - (6 * 9 / 2), 30); // Adjust position based on font size
  tft.print("15:27");

  // Display Date
  tft.setFont(NULL); // Use default font for date
  tft.setTextSize(1);
  tft.setCursor(tft.width() / 2 - (7 * 6 / 2), 50); // Adjust position based on font size
  tft.print("1st July 2025");

  // Display Settings Icon
  tft.setTextSize(2);
  tft.setCursor(tft.width() - 30, 25);
  tft.print("\u2699"); // Unicode for gear icon

  // Stress Levels Text
  tft.setFont(&FreeSansBold9pt7b);
  tft.setTextSize(1);
  tft.setCursor(30, 90);
```



prototypescreenforPM.ino

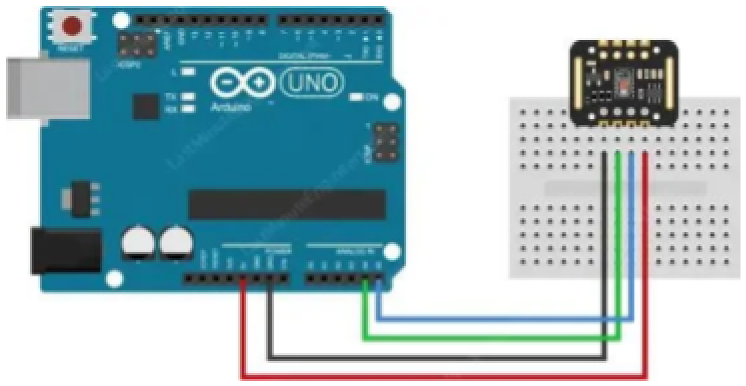
```
  tft.print("Stress");
  tft.setCursor(30, 105);
  tft.print("Levels");

  // Draw Stress Level Circle
  int16_t cx = tft.width() - 60;
  int16_t cy = 110;
  uint16_t radius = 30;
  tft.drawCircle(cx, cy, radius, DARKGRAY);

  // Draw Arc for Stress Level (23/100)
  float percentage = 23.0 / 100.0;
  float endAngle = 2 * PI * percentage - PI / 2; // Start from top
  int16_t innerRadius = radius - 5;

  for (int16_t i = 0; i <= 60; i++) {
    float angle = (2 * PI * i / 60.0) - PI / 2;
    if (angle <= endAngle) {
      int16_t x = cx + innerRadius * cos(angle);
      int16_t y = cy + innerRadius * sin(angle);
      tft.drawPixel(x, y, BLUE);
      int16_t outerX = cx + radius * cos(angle);
      int16_t outerY = cy + radius * sin(angle);
      tft.drawPixel(outerX, outerY, BLUE);
    }
  }

  // Display Stress Level Value
  tft.setFont(&FreeSansBold9pt7b);
  tft.setTextSize(2);
  String stressValue = "23";
  tft.setCursor(cx - 15, cy - 5); // Approximate centering based on typical character width
  tft.print(stressValue);

  tft.setFont(NULL); // Use default font for /100
  tft.setTextSize(1);
  tft.setCursor(cx + 18, cy); // Approximate positioning for /100
  tft.print("/100");
}

void loop() {
  delay(1000);
}
```

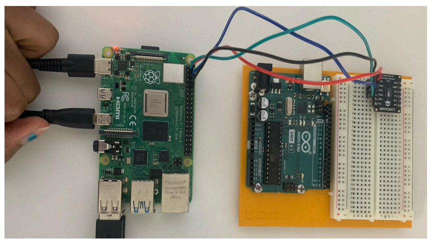
The outcome of the code was a light blue screen display with black writing stating the time, date and stress levels, plus a circle with an arc inside of it to indicate the stress level. The outcome was not an exact replica of the wireframes we'd designed, but we were able to replicate a similar version that kept the essential features of the screen intact.



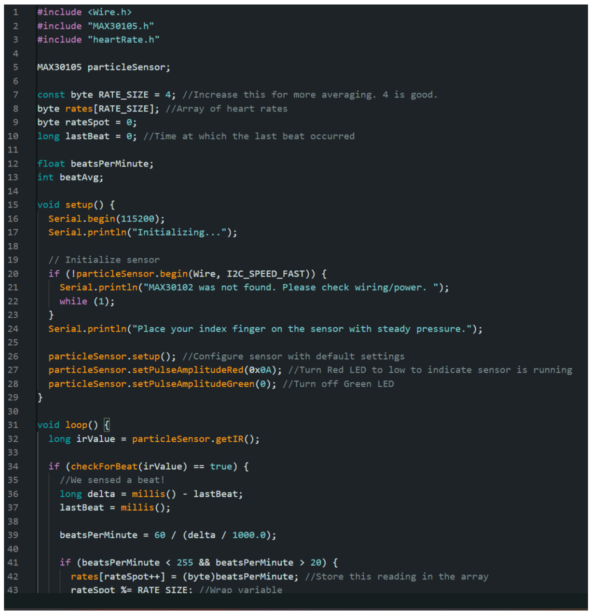
The next thing we did for the first prototype was try and connect the MAX30102 heart rate sensor to Arduino. Though we wanted to use the Raspberry Pi Pico to create our final version of the wristband, we felt that, for testing purposes, we could use the Arduino Uno just to try and read the heart rate data.

With the version of the heart rate sensor we had, to use the sensor on the Arduino breadboard, we had to solder on conductive pins to the sensor. We used 3 jumper cables to connect the sensor to the UNO – Ground to Analogue Ground, SCL to A4, SDA to A5 and Vin to 5V.

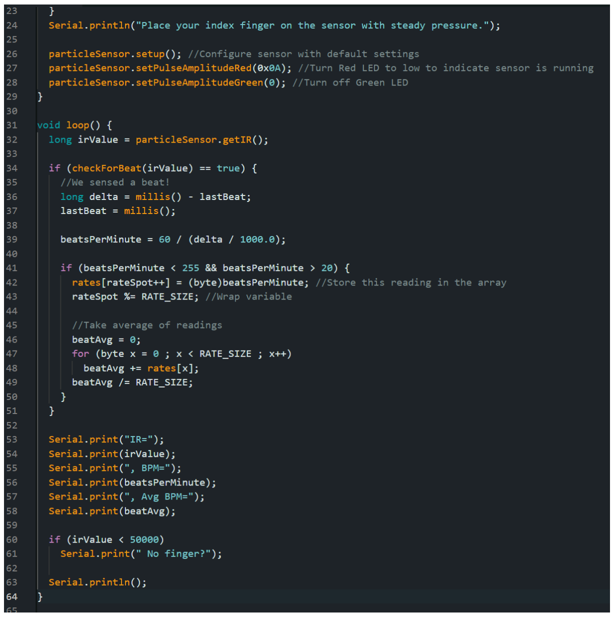




Then we wrote code (see below) to try and read the results from the sensor. We started by importing the relevant libraries: Wire.h, MAX30105.h and heartRate.h, then defined the sensor. The code then creates the variables beatsPerMinute to record the sensed bpm and beatAvg to record the average beat. Next, the code changes the serial baud to 115200 because this is the rate the MAX30102 sensor uses, so to get coherent results, the IDE baud rate needs to match the sensor's rate. The sensor is then initialised using an if statement to check if the sensor is connected to the board. If the sensor is detected, a statement is printed telling the user to put their finger on the sensor with steady pressure. In the setup function, the code also turns on the red LED built into the sensor to indicate to us if the sensor is on and working. After this, there is the loop function; within this, the code uses the if function to test if a heartbeat is sensed from the particle sensor in the heart rate sensor by using the pre-existing IR function and particle sensor functions. This data is converted into BPM by using the delta function and stored in an array called rates, allowing for the average BPM to be worked out. The results from the sensor are outputted (IR, BPM and average BPM). None of this code would be executed if the sensed IR value is below 5000, as that would mean no finger is being detected by the infrared scanner that the heart rate sensor is using to calculate the BPM.



```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"

MAX30105 particleSensor;

const byte RATE_SIZE = 4; //Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE]; //Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; //Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;

void setup() {
  Serial.begin(115200);
  Serial.println("Initializing...");

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 was not found. Please check wiring/power. ");
    while (1);
  }
  Serial.println("Place your index finger on the sensor with steady pressure.");

  particleSensor.setup(); //Configure sensor with default settings
  particleSensor.setPulseAmplitudeRed(0x0A); //Turn Red LED to low to indicate sensor is running
  particleSensor.setPulseAmplitudeGreen(0); //Turn off Green LED
}

void loop() {
  long irValue = particleSensor.getIR();

  if (checkForBeat(irValue) == true) {
    //We sensed a beat!
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute < 255 && beatsPerMinute > 20) {
      rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the array
      rateSpot %= RATE_SIZE; //Wrap variable
```



```
  }
  Serial.println("Place your index finger on the sensor with steady pressure.");

  particleSensor.setup(); //Configure sensor with default settings
  particleSensor.setPulseAmplitudeRed(0x0A); //Turn Red LED to low to indicate sensor is running
  particleSensor.setPulseAmplitudeGreen(0); //Turn off Green LED
}

void loop() {
  long irValue = particleSensor.getIR();

  if (checkForBeat(irValue) == true) {
    //We sensed a beat!
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute < 255 && beatsPerMinute > 20) {
      rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the array
      rateSpot %= RATE_SIZE; //Wrap variable

      //Take average of readings
      beatAvg = 0;
      for (byte x = 0 ; x < RATE_SIZE ; x++)
        beatAvg += rates[x];
      beatAvg /= RATE_SIZE;
    }
  }

  Serial.print("IR=");
  Serial.print(irValue);
  Serial.print(", BPM=");
  Serial.print(beatsPerMinute);
  Serial.print(", Avg BPM=");
  Serial.print(beatAvg);

  if (irValue < 50000)
    Serial.print(" No finger?");

  Serial.println();
}
```

Unfortunately, after executing the code, we realised the MAX30102 sensor we were using was not being detected. We double-checked that our wiring was correct and that there were no logic errors in the code that would lead to this, but the sensor was still not being found by the Uno or IDE. We attempted to use a Raspberry Pi Model 4B to test the same thing but were getting the same outcome. Online guides from the sensor manufacturer itself did not indicate we were doing anything wrong either (DFRobot, 2018). This led us to believe there may have been a fault with the sensor we were using.

Feedback Session

26th March: After our half-term check-in, we received feedback based on our original presentation and concept of the project. The feedback is below:

"Your wristband concept for measuring stress and promoting awareness in young people is timely and creative. Your approach combines technical application with important mental health considerations.

The integration of ECG sensors, body temperature monitoring, and skin conductance through e-textiles shows careful thought about biometric data collection. But I advise as you already noted that you choose one sensor – you can still create something artistic, engaging and technically sound with a more basic sensor that isn't ECG, etc. Think about it carefully and if you need anything specific, make sure you make a decision asap and let Alice know so she can order soon.

For hardware, the progression from Arduino Uno for proof of concept to Arduino Nano for prototyping shows practical thinking about development stages.

Things to consider:

- The planned ML integration will add valuable intelligence to your stress detection system but might be a bit rushed at this time, as we haven't really touched on this subject until the last 3 weeks of the unit. But if you feel strong about applying it and confident as a team that you can, go ahead.
- From your mood board inspirations, I suggest selecting one key interaction concept to focus on initially. This will help refine your design direction and user experience.

- Consider what screen design would best serve your users—perhaps a simple interface showing calm/alert states with breathing guidance?
- Consider adding artistic embroidery or other artistic components into your textile band, which will also require careful selection for a creative and technically strong output. Think of how you will present your final wristband or whatever you decide. I attach some artistic possible images for your band that you should also consider adding to your designs and prototypes.
- Bring in more references to your final works and visuals to support it. For example, what artists are driving your inspiration, what research, what articles, what works?

  These are some examples:

  https://marcodonnarumma.com/works/xth-sense/;

  https://www.scifilab.org/fingertrak

- Push often to your GitHub repo created on Wednesday, as this will showcase your iterative approach to your project. ".

After hearing this, we decided to implement some of these ideas within our project and adjust the concept. Our original aim was to create a working wearable wristband, similar to fitness watches, by the end of our project. This wristband would have measured the wearer's heart rate (EKG), sweat levels, and skin temperature and then used these readings to detect the wearer's stress levels. This data would also have been accessible within our app. This would then tell users when their stress levels are too high with haptics and a visual alert, and show them breathing exercises they can do to help reduce stress.

We changed our goal to a wristband with a temperature and humidity sensor that will still detect stress levels and alert the user when they are too high; however, this would be done through lights rather than an OLED screen. The lights will then execute a sequence to act as a guided breathing exercise. On the wristband itself, we will have embroidery to add to the overall aesthetic of the piece based on the artists' styles we liked. We will incorporate beads into the embroidery art to act as a fidget toy for the wearer as a way to reduce stress and add extra functionality to the band.

For our next steps, considering the feedback, we wanted to research artists and their art styles to get an idea of what embroidery we would have on the wearable. Additionally, we will create sketches for the embroidery art and execute these designs by embroidering them onto the nylon wristband we have for the project. We will also test out the temperature sensor and see if we can get it to link to an Arduino Uno. Also, we edited our mood board to have more of a focus on the LED aspect of the project.

User Review

Users have reported that removing the screen from the wearable wristband made it feel more practical and purposeful. Without a display, the bracelet becomes less obtrusive and more comfortable, allowing you to focus solely on what's most important: precise stress monitoring. Many users appreciate that it does not necessitate regular checking or engagement, allowing them to go about their business without interruptions or screen weariness. This basic approach has allowed the bracelet to flow more seamlessly into consumers' lifestyles. They are no longer under pressure to constantly "check" their data; instead, it syncs quietly in the background, delivering important insights as needed via a connected app. By removing the screen, the bracelet eliminates extraneous features and instead focuses on its essential function: helping people comprehend and manage stress in a subtle, seamless manner.

Revised Hardware Components
After changing our concept, we realised we would have to adjust the components we were utilising so that they would function effectively for our adjusted wristband. Our new hardware list consisted of:

- A DHT11 Temperature and Humidity sensor
- LEDs (most likely yellow, minimum of two)
- An Arduino Uno Board

We chose these components as they were the most cost-efficient whilst preserving their practicality. As a team, we also had slight experience with these components and due to time constraints, it felt realistic to use items we were more comfortable with rather than completely new items. On top of that, all the hardware we'd chosen had lots of online guides, materials and tutorials on how to use them, so we would find the making process simpler.