Text, logo

Description automatically generated

Internet of Things Project

Health-Tracking Smartwatch

Mikaela Diaz

Bachelor of Software & Electronic Engineering

Atlantic Technical University

2022/2023

**Project Poster**

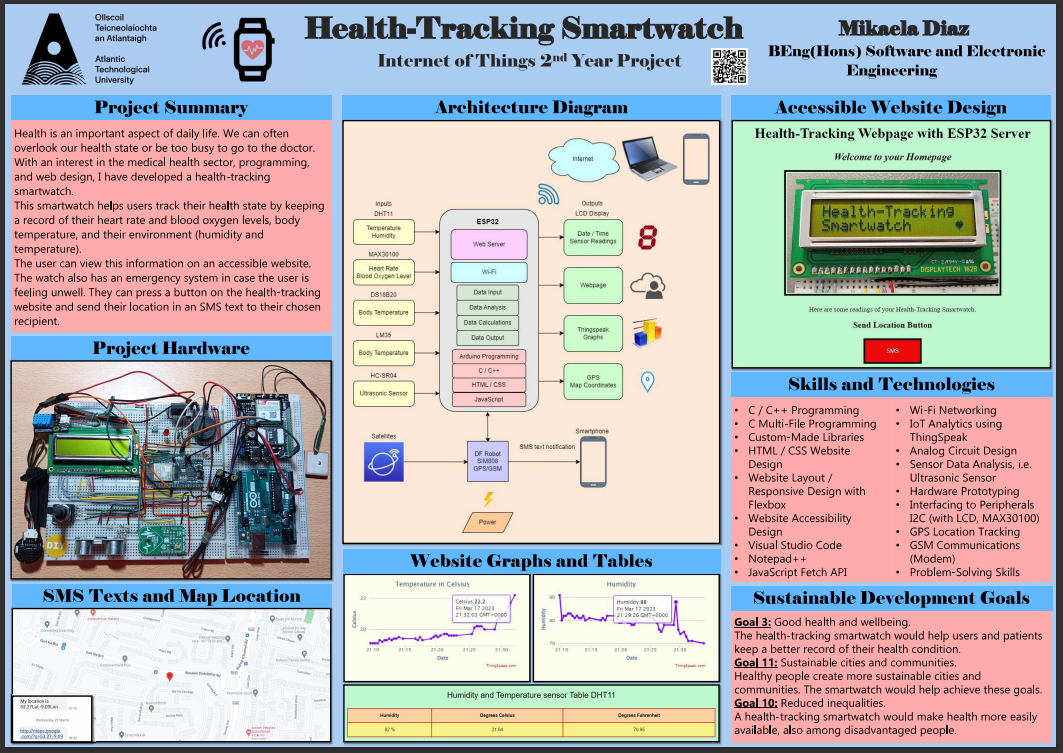


Figure 1 Project Poster

**Project Graphic (Hardware Image)**

A circuit board with wires

Description automatically generated with low confidence

Figure 2 Project Hardware Image

**Declaration**

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at the Atlantic Technical University, Galway campus.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Acknowledgements**

I wish to acknowledge the following people who kindly helped me during my project.

Natasha Rohan

Michelle Lynch

Brian O’Shea

Brian Costello

Parthib Barua

**Summary**

This project aims to improve the lives of common individuals by keeping a record of their health sensor readings so that they can be more aware of their health and improve their lifestyles.   
This is achieved by using several health sensors on the wearable health-tracking smartwatch to get the user’s health and environmental readings. These readings include the heart rate and blood oxygen level, body temperature, and the temperature and humidity of the air.

The health-tracking smartwatch also has the extra feature of detecting nearby obstacles to help visually impaired people orient themselves better. The watch also has an emergency system in case the user is feeling unwell. They can press a button on the health-tracking website and send their location in a short message service (SMS) text to their chosen recipient. The data collected by the sensors can be viewed on an accessible website in the form of tables and graphs. The sensor readings can also be displayed on a liquid crystal display (LCD).

The scope of this project involved the integration of hardware and software components to enable the collection, processing, and analysis of data related to health. This includes the use of sensors, data processing, user interfaces, and data analytics tools to provide users with insights about their health. I took into consideration the United Nations Sustainable Development Goals [1]. I found that my project targets three of those goals, which are: good health and wellbeing (goal 3), sustainable cities and communities (goal 11), and reduced inequalities (goal 10).

I approached the project by beginning to get sensors working individually, then I made the website, and I started integrating the sensor codes. Lastly, I added the GPS and SMS code.

The main methods and technologies used are C programming, global positioning system (GPS), SMS, C++ on the Arduino IDE, and website design using hypertext markup language (HTML), cascading style sheets (CSS), and JavaScript fetch application programming interface (API).

I accomplished most of my goals, though I didn’t get the integration of the ultrasonic sensor integrated with the code. The main conclusions for this project are that integration was the hardest part of the project. And there were more problems than I had anticipated.

**Table of Contents**

[1 Introduction 11](#_Toc134655336)

[1.1 Goal and Scope of the Project 11](#_Toc134655337)

[1.2 Motivation for the Project 11](#_Toc134655338)

[1.3 Layout of the Report 12](#_Toc134655339)

[2 Project Plan 13](#_Toc134655340)

[2.1 Semester 1 Plan 13](#_Toc134655341)

[2.2 Semester 2 Plan 13](#_Toc134655342)

[3 The Internet of Things Overview 15](#_Toc134655343)

[4 Project Architecture 16](#_Toc134655344)

[5 Development Platform and Tools 18](#_Toc134655345)

[5.1 Development Platform 18](#_Toc134655346)

[5.2 Integrated Development Environment (IDE) 18](#_Toc134655347)

[5.3 Development Platforms for Website Design 18](#_Toc134655348)

[5.4 Visual Studio Code 20](#_Toc134655349)

[5.5 Web Browser Developer Tools 20](#_Toc134655350)

[5.6 IoT Platform - ThingSpeak 21](#_Toc134655351)

[6 Sensors 22](#_Toc134655352)

[6.1 Temperature and Humidity Sensor 22](#_Toc134655353)

[6.1.1 Temperature and Humidity Sensor Code 23](#_Toc134655354)

[6.2 Temperature Sensor (LM35) 24](#_Toc134655355)

[6.2.1 Temperature Sensor Code (LM35) 25](#_Toc134655356)

[6.3 Temperature Sensor (DS18B20) 26](#_Toc134655357)

[6.3.1 Temperature Sensor Code (DS18B20) 27](#_Toc134655358)

[6.4 Heart Rate and Blood Oxygen Sensor 28](#_Toc134655359)

[6.4.1 Heart Rate Sensor Code (MAX30100) 30](#_Toc134655360)

[6.5 DF Robot Heart Rate Sensor 30](#_Toc134655361)

[6.5.1 Heart Rate Sensor Code (DF Robot) 31](#_Toc134655362)

[6.6 Ultrasonic Sensor 32](#_Toc134655363)

[6.6.1 Ultrasonic Sensor Code 34](#_Toc134655364)

[7 Location and SMS Texts 35](#_Toc134655365)

[7.1 Arduino Uno and DF Robot SIM808 35](#_Toc134655366)

[7.1.1 Arduino Uno and DF Robot SIM808 Code 36](#_Toc134655367)

[7.2 ESP32 and DF Robot SIM808 37](#_Toc134655368)

[7.2.1 ESP32 and DF Robot SIM808 Code 39](#_Toc134655369)

[7.3 ESP32 and DF Robot SIM808 Website 40](#_Toc134655370)

[8 Interrupts 41](#_Toc134655371)

[9 NTP Client 42](#_Toc134655372)

[10 Web Server 43](#_Toc134655373)

[10.1 Wi-Fi 44](#_Toc134655374)

[10.2 Web Server 45](#_Toc134655375)

[10.3 HTML & CSS Code 45](#_Toc134655376)

[10.3.1 Responsive Design 45](#_Toc134655377)

[10.4 Web Design for Accessibility 46](#_Toc134655378)

[10.4.1 Lighthouse Chrome Extension 46](#_Toc134655379)

[10.4.2 Perceivable 47](#_Toc134655380)

[10.4.3 Understandable 47](#_Toc134655381)

[11 Multi-File Programming and Libraries 48](#_Toc134655382)

[12 System Integration 49](#_Toc134655383)

[12.1 Hardware Integration 50](#_Toc134655384)

[12.2 Plan 51](#_Toc134655385)

[13 Problem Solving 53](#_Toc134655386)

[13.1 LM35 53](#_Toc134655387)

[13.2 LCD Display 53](#_Toc134655388)

[13.3 Heart Rate and Blood Oxygen Sensor 54](#_Toc134655389)

[13.4 DF Robot SIM808 55](#_Toc134655390)

[13.5 Website Integration 56](#_Toc134655391)

[13.6 Buzzer 56](#_Toc134655392)

[14 Impact of Project on Sustainability 57](#_Toc134655393)

[14.1 United Nations Sustainability Development Goals 57](#_Toc134655394)

[14.2 Accessibility 58](#_Toc134655395)

[14.3 Power Budget 58](#_Toc134655396)

[14.4 Health and Safety 58](#_Toc134655397)

[14.5 Programming Style 58](#_Toc134655398)

[14.6 Project Test 58](#_Toc134655399)

[14.7 Component Reuse and Recycling 58](#_Toc134655400)

[14.8 Plagiarism / Referencing 58](#_Toc134655401)

[15 Conclusion 59](#_Toc134655402)

[16 References 60](#_Toc134655403)

[Appendix 1: Code 63](#_Toc134655404)

[Appendix 2: Bill of Materials 64](#_Toc134655405)

[Appendix 3: Schematic 65](#_Toc134655406)

[Appendix 4: Simulation 66](#_Toc134655407)

# Introduction

## Goal and Scope of the Project

The goal of this project is to help people improve their health and lifestyle by using a health-tracking smartwatch to make them more aware of their health state, also aiding them in their daily life. This is achieved by keeping a record of their health readings on an accessible website. The health-tracking smartwatch helps them in their daily life by offering the possibility of sending an SMS text message containing their location to their chosen recipient if they are feeling unwell. In the case of a visually impaired user, the health-tracking smartwatch also helps them by enabling the possibility of detecting nearby obstacles so that they can avoid crashing with them.

The scope of this project involved the integration of hardware components such as sensors, a GPS, and an LCD. Also, the integration of software components such as C, C++, HTML, CSS, and JavaScript fetch API code to enable the collection, processing, and analysis of data related to health. The analysis and display of the collected data were carried out in the ThingSpeak analytics platform service.

## Motivation for the Project

The main motivation for this project was an interest in the medical health sector, programming, and web design. The reason for the medical health sector’s motivation is because not everyone can have access to a good healthcare system, especially people in underdeveloped countries. The health tracking smartwatch would make keeping a record and display of people’s health condition easier and more available. This could help prevent diseases by detecting early symptoms. It would also help users in their daily life by detecting obstacles and being able to send their location in case of feeling unwell.

Another motivation is that this project would help me to develop valuable skills oriented to a medical and programming sector. These skills might help me find employment in a company such as Boston Scientific.

## Layout of the Report

The report first does an introduction about the goal and motivations for the project giving an overview of the whole project. Then, there is a planning for semester 1 and semester 2. Afterwards, there is a section describing the internet of things uses in this project.

The next section has information about the architecture diagram of this project.

Then, there is information about the development platform and tools used for the project. This describes what tools were used for the development and design of the website. Also, what tools were used for the development of the main project code.

Then the report will first describe the use of the sensors individually, with code examples and diagrams.

Afterwards, there is a section specifically for the GPS module used for this project. It has information about how to make the module work with both ESP32 and Arduino Uno microcontrollers.

Then, there is a small section about hardware interrupts and another section about network time protocol (NTP).

Next, there is a section about the web server used to host the web page for this project. It describes details about the web page, HTML and CSS code, Flexbox, and accessibility. It also includes details about how the ThingSpeak IoT analytics platform was used in this project.

Then, there is a section about multi-file programming and a custom-made Arduino library.

Afterwards, there is a section about how the project was integrated, describing the steps taken and showing code examples. Next, there is a section describing the problem skills used for different roadblocks encountered during this project.

Lastly, there is a section about the impact of this project on sustainability, another section about the conclusion of this project, a section containing all references in this report and afterwards there are four appendices containing the code files, bill of materials schematic and simulation.

# Project Plan

I made a basic initial plan for my project proposal for semester 1. Then, for semester 2 I used OneNote to add the tasks that I had planned to do each week of the project.

## Semester 1 Plan

10/2022: Do research. Acquire some of the necessary components, start programming the ESP32 for the temperature and humidity sensor.

11/2022: Acquire all sensors. Program some of the sensors and check that they work properly.

12/2022: Program most sensors and check that they work properly. Start building a watch model and integrate some of the sensors, check that they respond properly.

## Semester 2 Plan

Week 1:

* Partially fix the website code.
* Make graphs in ThingSpeak using the DHT11 sensor.

Week 2:

* Continue to fix the website code.

Week 3:

* Finish fixing website code.
* Design how the website will be seen (using responsive web design).
* Fix GPIO 21 problem.

Week 4:

* Integrate website + write multiple fields + DHT11 ThingSpeak graph.
* MAX30100 + LCD.
* Get the ultrasonic sensor working.
* Multifile DHT11 + LCD code.

Week 5:

* Focus on coding.
* Work with the GPS.
* Fix website graphs code.

Week 6:

* Fix GPS code.
* Display time and date on the LCD.
* Test basic button interrupt.

Week 7:

* ESP32 integrate (GPS + SMS)
* Add temperature sensors (LM35 and DS18B20)
* Update the architecture diagram.
* Start the poster (architecture diagram, graphs, and website (output) summary, image of the project hardware and sustainable development goals (SDGs).
* Display the GPS location in ThingSpeak.

Week 8:

* Code upload 2.
* MAX30100 + ThingSpeak.
* Fix the ultrasonic sensor.

Week 9:

* Draft poster upload.
* Draft poster peer assessment.
* Poster (update image and summary).
* Website button.

Week 10:

* Report half done.

Week 11:

* Report finished.
* Finish Flexbox responsive website.
* Multifile website for simplicity.
* Website + LCD
* Integrate more code.
* Library for the ultrasonic sensor code.

Week 12:

* Draft report.
* Add accessibility features to the website.
* Integrate some code (main website code + button website).
* Video GPS working for the final video.

Week 13:

* Upload code to GitHub (final code, project proposal, video, poster, report).
* Study before the demo.
* Make the final project video.

Week 14:

* Upload the report, final video, and code.
* Presentation and demo.

# The Internet of Things Overview

This project uses the Internet of Things (IoT) by interconnecting physical individual sensors through wiring to an ESP32, then through code, and lastly through network connectivity. These sensors can collect and share data with other devices such as a laptop or smartphone. This allows the microcontroller (ESP32) [2] to connect and interact with the broader digital world. In this case, it is used to send those readings to ThingSpeak [3] and to a website hosted on the ESP32 server. ThingSpeak is an IoT Analytics platform service that allows for the aggregation and visualisation of live data streams in the cloud.

Fitness trackers such as the health-tracking smartwatch are a common application of IoT. The data collected by the sensors can be used for a variety of purposes, such as improving the user’s environment or alerting them if their health readings are dangerous.

This data can be viewed in different forms, such as in graphs, tables, or in the case of the GPS location, it can be viewed on a map.

# Project Architecture

The main development tool I used for most of my project was Arduino IDE. I also used Visual Studio Code to develop the HTML and CSS code for the website. The C programming language was used for programming the sensors. C++ strings were used to build the final message from the location of the DF Robot SIM808. This message is then converted to a C string before being sent by SMS text.

The development platform I am using is an ESP32 microcontroller. I chose the ESP32 instead of the Arduino Uno because it has a more powerful processor and built-in Wi-Fi. This was crucial for developing the health-tracking website. The ESP32 contains a web server that communicates to the internet via Wi-Fi to create the health-tracking website and send the sensor readings. JavaScript fetch API was used to detect the SMS button press on the website.

The architecture diagram shows the input sensors on the left connected to the ESP32, and on the right, it shows the outputs from the ESP32. The DF Robot SIM808 is connected to the ESP32 and powered by a 9V power supply. The DF Robot works is by communicating with satellites to obtain its location and sending it in an SMS message to a smartphone.

Diagram

Description automatically generated

Figure 4.1 Architecture Block Diagram

# Development Platform and Tools

## Development Platform

The ESP32 is a low-power consumption microcontroller that is often used in Internet of Things (IoT) applications. It is a dual-core processor with Wi-Fi and Bluetooth connectivity, and it can be programmed using the Arduino IDE or other software development tools. An ESP32 is more versatile than an Arduino Uno because it has built-in Wi-Fi and Bluetooth, which makes it easier to connect to other devices. The dual-core processor also makes it more powerful and capable of doing multiple tasks simultaneously.

## Integrated Development Environment (IDE)

The Arduino IDE was the most convenient development tool because it enabled the instant test if a piece of code was working. For example, testing if the code for the DHT11 sensor is working. Also, the Arduino IDE enabled fast error checking, which allowed the integration of code in steps and easily find bugs by checking every new piece of code.

The Arduino IDE also allowed for multi-file programming, which I used throughout the project to make code more readable and efficient.

## Development Platforms for Website Design

I used OneNote [4], Draw.io [5], W3Schools [6], and Notepad++ [7] for the design of the website.

OneNote was used to design the layout of the website, and W3Schools code editor was used to quickly test new features for the website, for example, to design the tables for the website.

Notepad++ was used to convert a web page .txt file into a .html file to view how to website would look in a web browser.

Graphical user interface

Description automatically generated

Figure 5.1 OneNote Website Layout Design

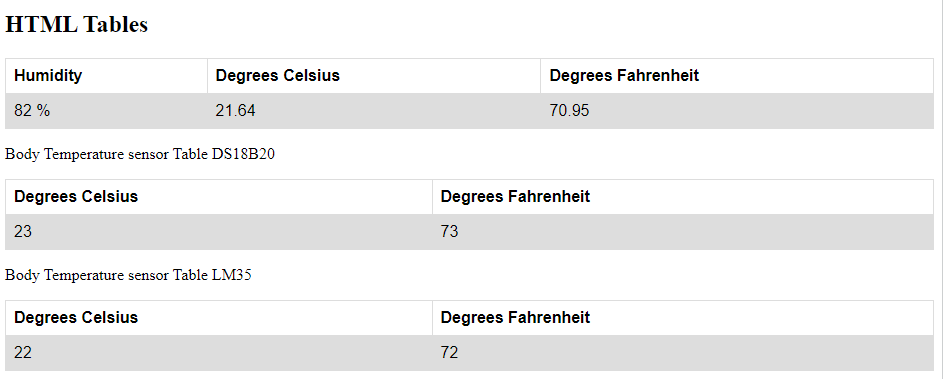


Figure 5.2 W3Schools Early Design of Tables

## Visual Studio Code

I used Visual Studio Code (VS Code) [8] when I had the base of my website designed. This is because VS Code allows for error checking, suggestions, colour selection, editing, and quick formatting of code with the shortcut Shift + Alt + F. It also allows for quick viewing of the website in a web browser with “reveal in file explorer”.



## Web Browser Developer Tools

I used Lighthouse [9] to test the accessibility of the website. The health-tracking website scored a 94% in accessibility.   
Lighthouse is a web server for Chrome that serves web pages from a local folder over the network by using hypertext transfer transmission protocol (HTTP). Lighthouse runs offline. The local folder I used for Lighthouse was my Visual Studio Folder, which contained the HTML and CSS design code for the health-tracking website with dummy values for the sensor readings.

Graphical user interface, website

Description automatically generated

Figure 5.3 Lighthouse Website Report

## IoT Platform - ThingSpeak

I used the free version of ThingSpeak [3] to display the readings from the DHT11 humidity and temperature sensor, MAX30100 heart rate and blood oxygen sensor, DS18B20 temperature sensor, and GPS coordinates.

I displayed these readings in the form of graphs, which then I added to the health-tracking website.

Graphical user interface, application

Description automatically generated

Figure 5.4 ThingSpeak Channels

A picture containing text, line, font, plot

Description automatically generated

Figure 5.5 ThingSpeak DHT11 Graphs

# Sensors

I used a DHT11 to record the humidity and temperature of the user’s environment, a DS18B20 and an LM35 to get the user’s body temperature, a MAX30100 heart rate and blood oxygen sensor, a DF Robot heart rate sensor, and an ultrasonic sensor to detect obstacles.

## Temperature and Humidity Sensor

The DHT11 sensor [10] is a low-cost digital temperature and humidity sensor. The DHT11 works by using a capacitive humidity sensor and a thermistor to measure the surrounding air, then it generates a digital signal on the output data pin. The DHT11 can measure temperatures with an accuracy of ± 2 degrees. It can also measure humidity with an accuracy of ± 5%. In this project, the DHT11 can help users detect if their environment is too dry, too hot, or cold for medical purposes.

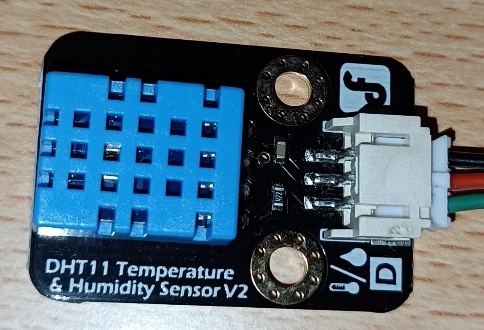


Figure 6.1 DHT11 Sensor Image

The DHT11 converts analogue signals from its temperature and humidity readings into digital signals. Analogue signals are continuous electrical signals, while digital signals are non-continuous electrical signals. This means that the conversion from analogue to digital signals allows for more accurate sensor readings.

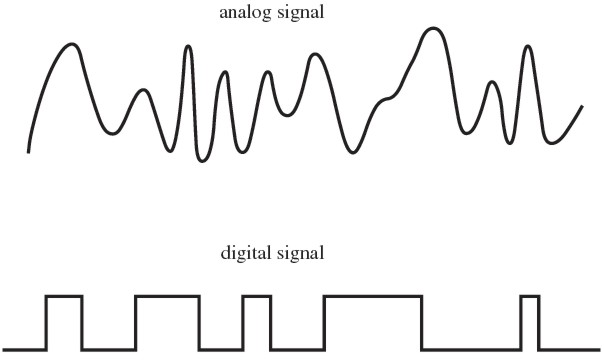


Figure 6.2 Analog Signal vs Digital Signal

For the DHT11 sensor, I used an input/output data pin, in this case, GPIO 4. The DHT11 can operate at 3.3V or at 5V. I chose to use 5V.

A diagram of a circuit board

Description automatically generated with low confidence

Figure 6.3 DHT11 Sensor Connection Diagram Image

### Temperature and Humidity Sensor Code

This code uses the library DHT [11] to call the functions that calculate the humidity and temperature in Celsius and Fahrenheit. In the third function, the “(true)” argument makes it possible to request the temperature value in Fahrenheit instead of Celsius.  
Afterward, the values obtained are printed on the serial monitor.

Text

Description automatically generated

Figure 6.4 DHT11 Functions

Before the values are printed to the serial monitor, an error checking if statement detects if the sensor readings are null. If they are, the code will not be printed, and an error message will be displayed instead.

A picture containing text, font, screenshot, line

Description automatically generated

Figure 6.5 DHT11 Error Checking Statement

## Temperature Sensor (LM35)

The LM35 sensor [12] is a temperature sensor that outputs an analogue voltage proportional to the temperature. Every 10 millivolts are equal to 1 degree Celsius. The output of the LM35 is linearly proportional to the temperature. It has an accuracy of ± 0.5 degrees Celsius at 25 °C.

In this project, the user can get their body temperature by touching the LM35. The LM35 sensor is an addition to the DS18B20 temperature sensor.



Figure 6.6 LM35 Sensor Image

The LM35 is connected to the GPIO 36 input pin of the ESP32.

A picture containing text, screenshot, diagram, line

Description automatically generated

Figure 6.7 LM35 Connection Diagram Image

### Temperature Sensor Code (LM35)

How the LM35 code works is the first line of code (in Figure 6.9) reads the analog-to-digital converter (ADC) value from the sensor. Then, in the second line, it calculates the value in millivolts using the previously defined constants. No libraries are used for this code. How the code calculates the millivolts value is by multiplying the analog-to-digital converter (ADC) value obtained from GPIO36 by the voltage reference (5V from supplied by the ESP32) which is divided by the ADC resolution. The ADC resolution is the ESP32 ADC resolution, in this case 12 bits. Therefore 2^12 = 4096.

A picture containing text

Description automatically generated

Figure 6.8 LM35 Constants

Then, in the fourth line, it calculates the value in degrees Celsius (every 10 mV equals 1 °C). This formula is mV / 10. Afterward, in the next line, the value is converted to Fahrenheit by the Celsius to Fahrenheit formula.

Graphical user interface, text

Description automatically generated

Figure 6.9 LM35 Code

## Temperature Sensor (DS18B20)

The DS18B20 is a digital temperature sensor [13]. It can operate at voltages from 3.3V to 5V. It has an accuracy of ± 0.5 degrees Celsius from -10 °C to 85 °C.

This sensor is used as the main body temperature sensor in this project. The user can get their body temperature by touching the DS18B20.

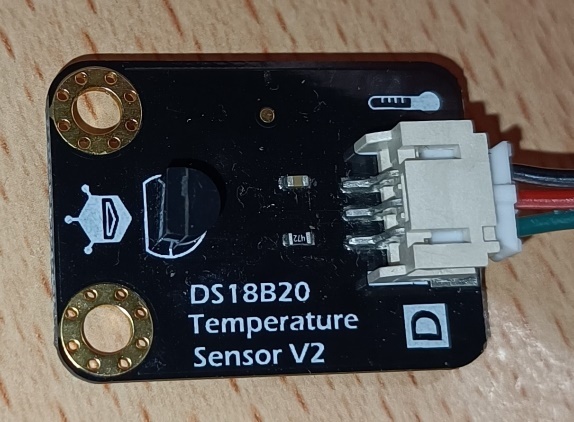


Figure 6.10 DS18B20 Sensor Image

The DS18B20 is connected to the GPIO 0 input/output pin of the ESP32. It has a pull-up resistor connected between the power supply voltage and data pins. The pull-up resistor makes sure that the data line always has a specific voltage when the sensor is not sending any information. This helps the DS18B20 to communicate with other devices without any confusion or mistakes. This feature is useful in my project to avoid interference with the DHT11 or LM35 sensors.

A diagram of a circuit

Description automatically generated with low confidence

Figure 6.11 DS18B20 Connection Diagram

### Temperature Sensor Code (DS18B20)

This code uses the one wire library [14]. How the code works is the two first lines of code are used to initialise the one wire bus communication and create an instance of the Dallas temperature library.

Text

Description automatically generated

Figure 6.12 DS18B20 Code Instances

The first line of code initiates a temperature conversion on the DS18B20 sensor. Then, the next two lines read the temperature values in degrees Celsius and Fahrenheit.

Text

Description automatically generated

Figure 6.13 DS18B20 Code

After this code, the temperature in degrees Celsius and Fahrenheit is printed to the serial monitor.

## Heart Rate and Blood Oxygen Sensor

The MAX30100 is a pulse oximetry and heart rate monitor sensor [15]. It operates from 1.8V to 3.3V. The module uses a two-wire I2C interface for communication with the ESP32 microcontroller.

The MAX30100 has a finger placement pad, seen as the small black square in the image below (Figure 6.14). How this sensor works is by using two light-emitting diodes (LEDs), a red LED and an infrared LED, and a photodetector. When the finger is placed on top of the pad, the light from both LEDs is reflected on the photodetector.

When the blood is pumped through the finger with each heartbeat, the amount of reflected light changes, creating a changing waveform at the output of the photodetector. As you continue to shine light and take photodetector readings, you quickly start to get a heartbeat (HR) pulse reading.

To get the blood oxygen level, it simply works by the more amount of light detected by the photosensor, the less oxygen in the blood. This is because when there is a low amount of oxygen in the blood, there is also a low absorption of light.

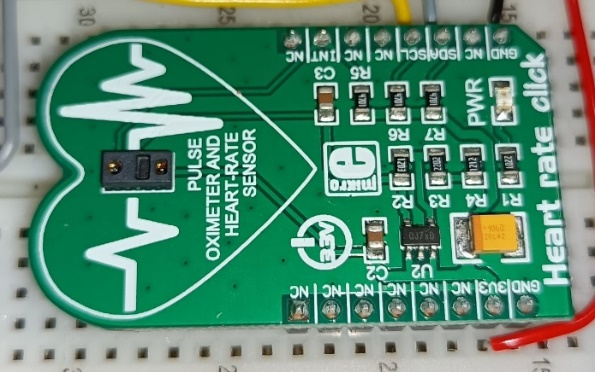


Figure 6.14 MAX30100 Sensor Image

The MAX30100 is connected to GPIO 22 and GPIO 21 input/output pins of the ESP32. The SCL pin of the MAX30100 is the serial clock pin, and the SDA is the serial data pin. The SCL pin controls the timing of data transmission. The SDA pin is used to send and receive data between the MAX30100 and the ESP32. The MAX30100 sensor uses I2C.

The I2C interface is used to send and receive data, such as the heart rate and blood oxygen level readings, as well as to configure the sensor settings, such as the LED brightness and sampling rate. In this project, I configured the sampling rate to be of 5 seconds instead of 1 second to make it easier during the integration with the ThingSpeak code.

A diagram of a circuit

Description automatically generated with low confidence

Figure 6.15 MAX30100 Connection Diagram

### Heart Rate Sensor Code (MAX30100)

The MAX30100 uses the library MAX30100 pulse oximeter [16]. How this code works is it first creates a pulse oximeter object called pox.

A picture containing text

Description automatically generated

Figure 6.16 MAX30100 Object

Then, that object is initialized in void setup. In the first line of Figure 6.17, the pox update method reads from the sensor. The next line checks if the time elapsed since the last report is greater than the reporting period of 1 second (defined previously). If it’s greater, it will print the readings obtained to the serial monitor.

Text

Description automatically generated

Figure 6.17 MAX30100 Code

## DF Robot Heart Rate Sensor

This sensor is the DF Robot heart rate sensor. I added this sensor as an extra for the project. It works in a similar way to the MAX30100 sensor in terms of LED and photosensor. Its input voltage is from 3.3V to 6V (5V recommended). Tutorial website for this sensor [17].



The DF Robot heart rate sensor is connected to the GPIO 33 input/output pin of the ESP32.

Figure 6.18 DF Robot Heart Rate Sensor Image

A picture containing text, diagram, screenshot, line

Description automatically generated

Figure 6.19 DF Robot Heart Rate Sensor Connection Diagram

### Heart Rate Sensor Code (DF Robot)

This code works by using the library DF Robot heart rate [18]. It creates an instance called heart rate using the digital mode option (because the DF Robot sensor is connected to a digital pin of the ESP32). The sensor has the option to used in analogue mode.

Then, the method ‘get value’ samples the sensor, and the ‘get rate’ method obtains the heart rate value obtained in the sampled data.

If the rate value variable is not 0, the value is printed to the serial monitor.

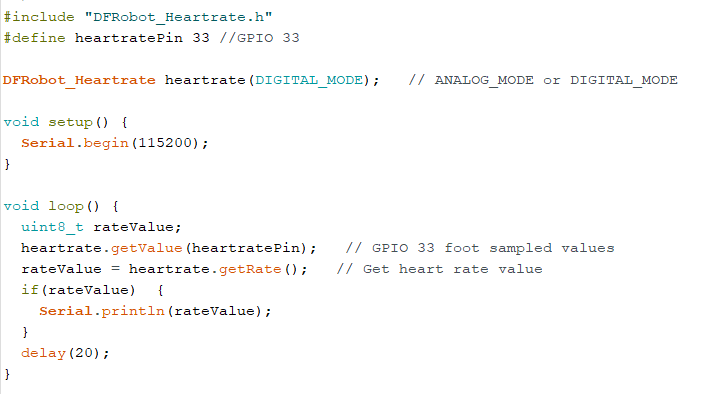


Figure 6.20 DF Robot Heart Rate Code

## Ultrasonic Sensor

The ultrasonic sensor HC-SR04 is used for distance measurement [19]. It works at 5V and has a maximum range of 4 meters. It has a measuring angle of 15 degrees, which is quite small. This means it can only detect objects that are almost right in front of it.

In this project, the ultrasonic can help visually impaired users detect obstacles in front of them to avoid crashing and make walking easier. This would greatly improve their quality of life.

How the ultrasonic sensor works is by using a transmitter and a receiver. The sensor transmits a short ultrasonic pulse towards an object and measuring the time it takes for the pulse to be reflected on the receiver, it calculates the distance of the object.



Figure 6.21 Ultrasonic Sensor Image

The HC-SR04 is connected to the GPIO 18 and GPIO 5 input/output pins of the ESP32. This is because these pins support pulse width modulation and interrupts, which are essential for the sensor to work properly. The GPIO 18 pin generates a trigger signal, while the GPIO 5 pin listens for the echo signal.

The ESP32 is a 3.3V device, and the HC-SR04 sensor is a 5V device. Using the resistor divider to connect to the ESP32, the voltage is stepped down to be 3.3V compatible.

A diagram of a circuit

Description automatically generated with low confidence

Figure 6.22 Ultrasonic Sensor Connection Diagram

### Ultrasonic Sensor Code

The trigger pin of the sensor is set as an output because it generates the trigger signal, while the echo pin is set as an input because it receives the echo signal.

Text

Description automatically generated

Figure 6.23 HC-SR04 Outputs

How this code works is by measuring the pulse on a pin (TRIG\_PIN) waits for the pin to go from low to high, starts timing, then waits for the pin to go low, and stops timing. This is to measure the time taken for the signal to travel from the sensor to an obstacle and back to the sensor.

Graphical user interface, text, application, email

Description automatically generated

Figure 6.23 HC-SR04 Code

The code then returns the length of the pulse in microseconds ‘duration’ or gives up and returns 0 if no complete pulse was received. The formula to calculate the distance of an object is given by distance = (time \* speed of sound) / 2. The formula is divided by two because the signal travels from the sensor and back to the sensor. The distance is stored in centimetres in the variable ‘cm’.

Graphical user interface, text, application, email

Description automatically generated

Figure 6.24 HC-SR04 Code

# Location and SMS Texts

The DF Robot SIM808 GPS/GPRS/GSM is an Arduino shield [20]. It has an operating voltage of 5V. Its input power is 7V to 23V.

In this project, the DF Robot SIM808 works by sending an SMS text message containing the previously acquired GPS location. This is with the purpose that, if the user is feeling unwell, they can send their GPS location to their chosen recipient as an emergency mechanism.

I used the Arduino Uno for testing how to integrate the acquire GPS code and send SMS code. After completing the testing, I transitioned to using the ESP32 instead and integrating the GPS SMS code with the button website code.

## Arduino Uno and DF Robot SIM808

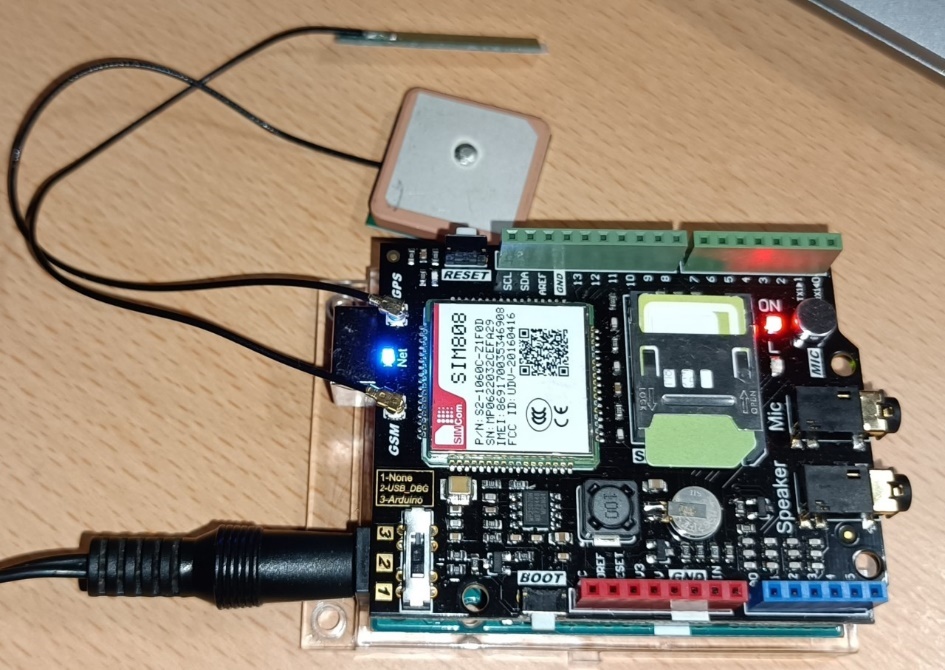


Figure 7.1 DF Robot SIM808 Circuit with an Arduino Uno

### Arduino Uno and DF Robot SIM808 Code

This code uses the library DF Robot SIM808 [18] and DF Robot\_SIM808 sim808(&Serial).

How it works is it first initialises the DF Robot. If the initialization fails, the code waits for 1 second and prints an error message to the serial monitor. Then, the code waits for the GPS to attach by checking if the SIM808 has been successfully attached to the GPS.

Text

Description automatically generated

Figure 7.3 DF Robot Initialization Code

Then, the code gets the data and creates two variables to store the latitude and longitude. These are in the form of C++ strings. The C++ string is converted to a C string using to char array. This allows the SMS message to be sent to the chosen defined phone number’s recipient.

Text

Description automatically generated

Figure 7.4 DF Robot Get Data Code

Then, the GPS power is turned off and the SMS text message is sent to the receiver.

Graphical user interface, text, application

Description automatically generated

Figure 7.5 DF Robot Send SMS Code

## ESP32 and DF Robot SIM808

In the image seen below, the DF Robot SIM808 is powered using the Arduino Uno [21] to obtain 9V. The Arduino Uno is connected to a 9V power adapter.

A picture containing text, electronics, circuit

Description automatically generated

Figure 7.6 DF Robot SIM808 Circuit Powered with an Arduino Uno

To connect the Arduino Uno to the DF Robot, connect Vin from the Arduino Uno (Vin of the Arduino Uno can also output voltage) to Vin of the DF Robot. Then, ground to ground. Also, connect ground to ground of the Arduino Uno and ESP32.

A picture containing text, diagram, plan, line

Description automatically generated

Figure 7.7 DF Robot SIM808 and ESP32 Connection Diagram

The ESP32 contains the web server that hosts the SMS button website.

The request to send the text message is done by pressing the SMS button on the health-tracking website. The button works by using JavaScript fetch API.

Figure 7.2 is a flowchart for this process.   
First, the ESP32 starts and connects to the internet. Then, the DF Robot SIM808 is initialised, and the website is displayed. Next, if the SMS button is pressed, the SIM808 will get the GPS location and send it on the SMS text message. After the process in complete, the code goes back to waiting for an SMS button press.

A picture containing text, diagram, screenshot, line

Description automatically generated

Figure 7.2 High Level Flowchart of Website SMS button code

### ESP32 and DF Robot SIM808 Code

This code uses the library DF Robot SIM808 [18] and DF Robot\_SIM808 sim808(&Serial2).

How it works is it first initialises the DF Robot, then it gets the data and creates two variables to store latitude and longitude. These are in the form of C++ strings. The C++ string is converted to a C string using to char array. This allows the SMS message to be sent to the chosen defined phone number’s recipient.

The main difference in the code of the ESP32 vs the Arduino Uno is that the ESP32 uses Serial2 instead of Serial. It uses Serial2 because in UART2, GPIO 16 is RX and GPIO 17 is TX.

In the ESP32, Serial is the default serial interface. Serial1 and Serial2 are additional Serial interfaces that can be used for other purposes, such as communication with external devices. Serial2 uses pins GPIO16 (Connected to RX of the DF Robot) and GPIO17 (Connected to TX of the DF Robot).



Figure 7.8 Serial and Serial2

## ESP32 and DF Robot SIM808 Website

This is the button website, which is a separate website from the main website where the sensor readings in the form of tables and graphs are displayed. This button website uses JavaScript Fetch API to make the request to the DF Robot SIM808.

The code in Figure 7.9 creates the HTML button that, when clicked, sends a get request to the web server with a query indicating the button was clicked. The web server can then use this information to trigger the get GPS and send SMS code.



Figure 7.9 HTML Code

The output when the button is pressed is a link to the Google Maps location.

Text

Description automatically generated

Figure 7.11 Button Press Output

# Interrupts

I planned to use a timer interrupt to check the MAX30100 after a certain period so that I could integrate it with other sensors, but I didn’t reach this point in the project, so I only practiced doing a simple hardware interrupt with a pushbutton using software debouncing.

How the code works is it uses two variables, one for the number of switch presses, and another to detect if the switch has been pressed or not, therefore using a bool variable.

In setup, set the interrupt as a falling edge to detect the switch pin change only when the button is pressed (a falling edge). Tutorial website [22].

Text

Description automatically generated

Figure 8.1 Setup Code

Then, if the button has been pressed, check that the last time the button was pressed was longer than 250 milliseconds, therefore avoiding switch bouncing.

Set the value as true and recorded the last time the button was pressed as the new time.

Text, letter

Description automatically generated

Figure 8.2 Interrupt Function

Then, if the button was pressed, print the message indicating it and how many times it was pressed. Then, reset the value to false.

A picture containing website

Description automatically generated

Figure 8.3 Print Feedback

# NTP Client

The health-tracking smartwatch displays the date and time on its LCD. Tutorial website [23].

How the code for the time and date works is by connecting via Wi-Fi to the nearest network time protocol (NTP) server and getting the time and date.

Then, using the liquid crystal library [24] and pin constants, an LCD object is created which sets the cursor and prints the values.

In Figure 9.1, the network pool, GMT offset and daylight-saving time constants are configured.

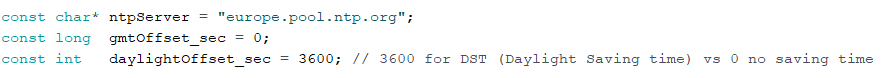


Figure 9.1 NTP Pool

Then, in figure 9.2, the code checks if it can get the data, and then prints it to the LCD.

Text

Description automatically generated

Figure 9.2 NTP Code

# Web Server

Using the ESP32’s web server, I made a health-tracking website to display the user readings in the form of tables, graphs, and on an LCD. I considered visually and mobility impaired people by making good contrast between the colours and a big button for the SMS message.

Graphical user interface, text

Description automatically generated with medium confidence

Figure 10.1 Website

Timeline

Description automatically generated

Figure 10.2 Website Tables for DHT11, DS1820, LM35

A picture containing graphical user interface

Description automatically generated

Figure 10.3 Website ThingSpeak Graphs for DHT11

Graphical user interface

Description automatically generated with medium confidence

Figure 10.4 Website ThingSpeak Graphs for DS18B20

## Wi-Fi

Wi-Fi is used to connect the ESP32 to the internet, therefore being able to access the web browser and ThingSpeak. To connect to the Wi-Fi the service set identifier (SSID) and password are set as constants. The ‘Wi-Fi begin’ function is responsible for making the connection. While the Wi-Fi is not connected, a dot is printed to the serial monitor every second.

Text

Description automatically generated

Then, when the IP address of the website is generated, copy and paste it to a web browser to see the website.

Figure 10.1 Code for Wi-Fi connection

Text, letter

Description automatically generated

Figure 10.2 Wi-Fi Connection Output

## Web Server

To serve web pages or handle HTTP requests, the ESP32 acts as a web server.

## HTML & CSS Code

### Responsive Design

I used Flexbox to adapt the website to different screen sizes and resizing. In the image below, the LCD image and tables can be seen resizing due to the Flexbox in the code.

Diagram

Description automatically generated

Figure 10.3 Flexbox Website Demonstration

## Web Design for Accessibility

### Lighthouse Chrome Extension

The lighthouse Chrome extension [9] generates a web server for Chrome. It serves web pages from a local folder (VS Code folder in this case) over the network using HTTP.

Then, it runs an accessibility audit of the selected website.

Graphical user interface, website

Description automatically generated

Figure 10.4 Lighthouse Report

### Perceivable

The web content is made available to the sight, hearing and touch.

The web content is easily accessible because of the good colour contrast of the website elements. The green, orange, and yellow colour with the black casing is easily seen by colourblind people. Also, the elements on the website are big and easy to see. The button is also red and big, easily seen and clicked. The screen can be touched to select the buttons or navigate the website.

The language of the website is specified so that screen readers can read it.

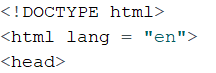


Figure 10.5 Screen Reader Code in English

A screen reader would be able to read the alt text of this image instead of reading the link for the image.



Figure 10.5 Alt Text for an Image

### Understandable

The information on the website is easy to understand. There are tables and graphs that display the same data in a clear understandable way. The operation of the website is also easily understood. There is only a button.

# Multi-File Programming and Libraries

For the ultrasonic sensor HC-SR04, I used a custom-made library containing functions using multi-file programming. The ultrasonic sensor is also integrated with the LCD, an LED, and a buzzer.

The ultrasonic sensor displays the distance in the LCD, and when an object is closer than 60 centimetres, it prints a warning message to the LCD, sounds the buzzer, and the LED turns on.

In the .h file, the function prototypes for the library are created.

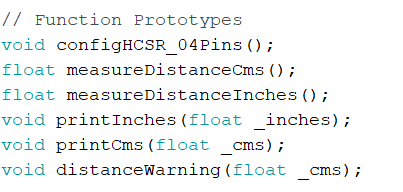


Figure 11.1 Function Prototypes in the .h File

Then, the measure distance function gets the distance in centimetres (the process was already explained in the sensors section).

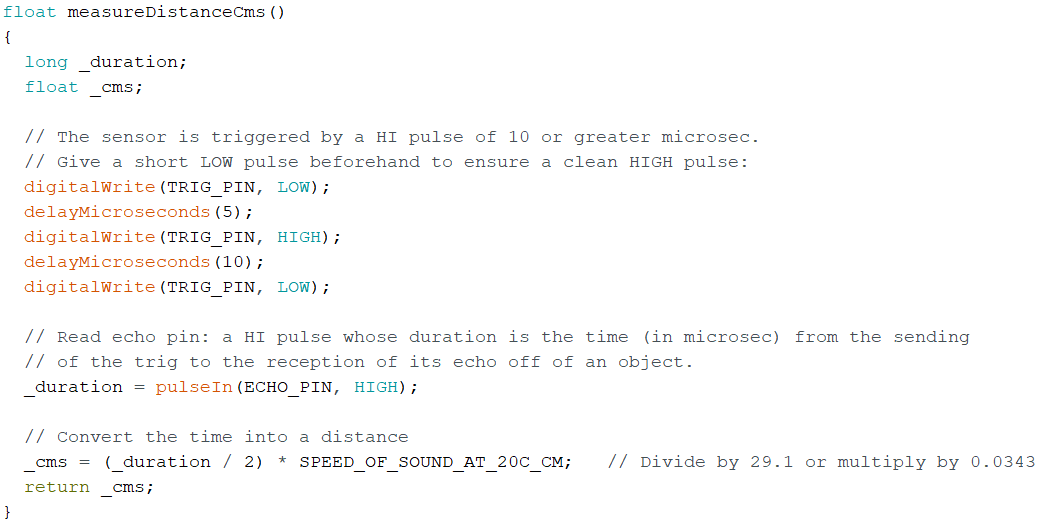


Figure 11.2 Measure Distance in Centimeters Function

Then, there is the distance warning function to check if the distance is closer than 60 centimeters. If it is, the buzzer and LED turn on and the LCD prints the warning message, if it is not, the buzzer and LED are off.

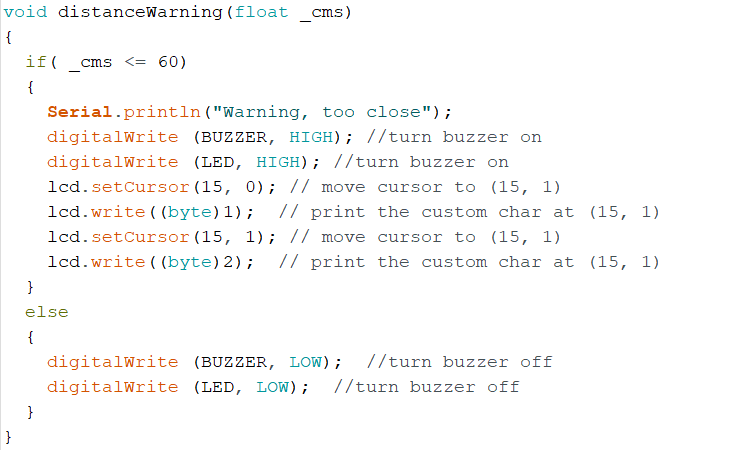


Figure 11.3 Distance Warning Function

# System Integration

The method I followed for integrating each part of the project was by investigating the specific sensor, library, or piece of code I wanted to integrate. Then I planned what I wanted the result of the code to do. I designed the code and created it. Then I evaluated the code to make sure it did what it was intended to do and checked if I could make any improvements to the integrated code.

Diagram

Description automatically generated

Figure 12.1 Iterative Design Process

## Hardware Integration

I used Altium Designer to create a schematic diagram of the connection and integration of the different hardware components. To choose which GPIO pin to use with each hardware component, I often used the ESP32 pinout diagram [25] to see which pins were more suitable for each component.

**A picture containing diagram, technical drawing, plan, schematic

Description automatically generated**

Figure 12.2 Schematic Diagram of the IoT Project

## Plan

I did some of the planning in OneNote to visualise how every part of the project would be integrated.

A picture containing diagram

Description automatically generated

Figure 12.2 OneNote Planning

I encountered the problem of not knowing how to integrate the ultrasonic sensor with the website because they are two very different pieces of code.

I also encountered the problem of not fully being able to integrate the main website containing the tables and graphs with the JavaScript Fetch API button website. This is because the button website uses a different connection method than the main website.

The main website uses Wi-Fi Server while the SMS button website uses web server.

Text

Description automatically generated with medium confidence

Figure 12.3 Main Website Connection

Text

Description automatically generated

Figure 12.4 Button Website Connection

I also encountered the problem of not being sure how to integrate the code for the MAX30100 sensor (working with ThingSpeak) with the main page code. This is because the MAX30100 must work for a very specific period of time. This is explained in the next section.

# Problem Solving

## LM35

When integrating an LM35 with an ESP32, I found that the temperature result was far too low (approximately 10 degrees Celsius too low). The code and calculations were right.

ADC is Non-linear, you would expect a linear behavior when using the ESP32 ADC pins. However, that doesn’t happen. This is because the ESP32 is not accurate at low voltages.

## LCD Display

When integrating the ESP32 with the LCD, I struggled to solder the potentiometer to control the contrast of the display. Then, the display didn’t work, and I realized that the LCD was broken. After replacing it, I realized that I had forgotten to put the R/W pin on the LCD to ground.

A picture containing text, diagram, line, plot

Description automatically generated

Figure 13.1 LCD Display Connection Diagram

## Heart Rate and Blood Oxygen Sensor

The integration of the MAX30100 sensor and ThingSpeak code gave me a roadblock for a few weeks. I had trouble trying to send MAX30100 sensor heart rate and blood oxygen level readings to ThingSpeak. I couldn't identify the problem because the code broke when I added the ThingSpeak code with a delay of 20 seconds. I removed the delay of 20 seconds and the code kept breaking, meaning that the code read the values only once and then kept repeating them repeatedly after sending them to ThingSpeak. I started the code again and integrated the ThingSpeak code in small pieces. Then, I found the exact line that broke the code.

The image 13.2 is the line of code that caused the problems. This line is sending data to the ThingSpeak channel using the ThingSpeak library [26].

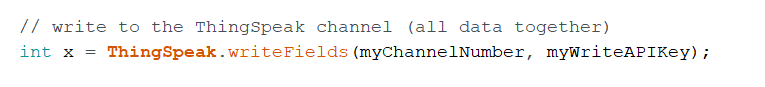


Figure 13.2 Write to Channel Code

I added some functions to measure the delay of this line of code. I found that this line of code takes 1.2 seconds to run. This is the reason that this line broke the code. Then, I deleted that function and added a delay to see at what point the code stopped breaking. I found that the code stopped breaking every 150 to 300 milliseconds.

Then I thought to add an interrupt to read the sensor every 100 milliseconds so that the ThingSpeak write fields function doesn’t interrupt with it. I wasn’t sure if this method would work because the new timer that I would be trying to use could be already in use in the code by one of the two libraries (the ThingSpeak or MAX30100 libraries). I measured the time it took to execute the function to read the sensor. I found that it takes less than a millisecond to read from the sensor. This is good because this means the interrupt time of 150 milliseconds wouldn't be shorter than the time to update the sensor.

Text

Description automatically generated

Figure 13.3 Code Testing

I kept testing the code, I had previously thought that the only problem in the operation of ThingSpeak is that the values are not reset, so for the sake of trying I copied the function to start the MAX30100 sensor at the end of the code, after the heart rate and blood oxygen values have been sent to ThingSpeak, and this function worked as a reset for those values, which was the problem that I originally had. So, the problem was solved without needing to use an interrupt to read the sensor every 150 milliseconds. I didn't find much information about the MAX30100 library, and there is no reset function.



Figure 13.4 Function to Reset Readings

But then I found another problem, I got a -401 error in my serial monitor every second time I sent data to ThingSpeak. I figured out that it was because I was sending the data every 15 seconds, which is a bit too fast for ThingSpeak. So, I adjusted the value to send the data every 30 seconds.

## DF Robot SIM808

The DF Robot SIM808 is supposedly only compatible with Arduino products, but I figured out how to make it work with the ESP32 microcontroller.

I used the website [20] for help in how to make the DF Robot work. Then I adapted the code to use Serial2 for GPIO pins 16 (RX) and 17 (TX). I used Serial2 because in UART2, GPIO16 is equivalent to RX, and GPIO17 is equivalent to TX. I used a website for this research [27].

## Website Integration

I also encountered problems integrating the main website containing the tables and graphs with the JavaScript Fetch API button website.

The main website uses Wi-Fi Server while the SMS button website uses web server.

I joined both websites using the Wi-Fi Server method and they worked, but when the SMS button is pressed, after the SMS message has been sent, the website won’t keep updating the sensor values on the website. I couldn’t find the reason for this problem.

## Buzzer

I found a hardware problem with the buzzer [28]. The buzzer I used is a Piezo buzzer. It works at less than 20 milliamps and works from 12V to 220V. This buzzer needs a flyback diode in parallel to prevent voltage spikes because the piezo buzzer works both ways and the mechanical stress can create a voltage.

A diagram of a circuit

Description automatically generated with low confidence

Figure 13.5 Circuit of a Piezo Buzzer [28]

The transistor is not necessary because the current is less than 20 milliamps. But I can’t connect the circuit without the transistor, so I need it anyway. The problem is that that type of transistor is not in stock, therefore I don’t have one.

# Impact of Project on Sustainability

## United Nations Sustainability Development Goals

I considered the impact of my project on the united nation’s sustainability development goals (UN SDGs) [1]. I found that my project hits up to four different goals.

**Good Health and Wellbeing (Goal 3):**

3.8 (Universal health coverage): The health tracking smartwatch would help users and patients keep a better record of their health conditions so they can be alerted if something goes wrong, they can realize it and get medical help faster. This would improve the health and wellbeing of the patient.

3.6 (Road traffic): The health tracking smartwatch could help alert the user if they are going to go unconscious or have a seizure by detecting a very fast heart rate and alerting the user by SMS text. This could prevent road accidents.

**Gender Equality (Goal 5):**

5.1 (End all forms of discrimination against all women and girls everywhere): The smartwatch would promote health for every person equally, making gender equality more accessible.

**Reduced Inequalities (Goal 10):**

A health-tracking smartwatch would make health more easily available, also among disadvantaged people.

**Sustainable Cities and Communities (Goal 11):**

Healthy people create more sustainable cities and communities. The smartwatch would help achieve these goals.

## Accessibility

I added captions to links and imaged on my website.



Figure 14.1 Caption Image

## Power Budget

The ESP32 microcontroller consumes less power than an Arduino Uno.

## Health and Safety

I took health and safety precautions in the lab by wearing protective glasses while soldering the potentiometer for the LCD.

## Programming Style

I followed a consistent programming style throughout my code by using the same type of layout for the curly braces around the block of code.

Text

Description automatically generated

Figure 14.1 Curly Braces

## Project Test

I tested that each piece of code worked properly before integrating it step by step with another piece of code, while testing each new block of code to detect errors and fix them as soon as possible.

## Component Reuse and Recycling

Every component used in this project will be returned to ATU. This is done with the purpose of reusing the same components instead of buying new ones, therefore reducing the cost and the hardware of the projects, also minimizing the amount of package waste produced.

## Plagiarism / Referencing

I added all the links and references to in the references section of this report.

# Conclusion

The outcomes of the project are two websites, one that displays the sensor readings (DHT11, DS18B20, LM35) in the form of tables and graphs (by using ThingSpeak). The readings are also displayed on the LCD. And the other website (SMS button website) uses JavaScript Fetch API to make a button work when it is pressed and sends the user location in an SMS text message in the form of a Google Maps location link.

Then, the integrated website of the sensor readings website and the button website, but this website can only send one SMS text message and then the code stops, so it doesn’t fully work.

Also, an ultrasonic sensor code to turn on an LED and sound buzzer if the ultrasonic sensor detects an object too close to the user (60 centimetres). It also displays the distance on the LCD, it also prints the warning message on the LCD.

Then, the code for the MAX30100 sensor working with ThingSpeak. Also, the LCD code that can display the sensor readings separately from the website.

Also, the code for the DF Robot heart rate sensor working with the LCD as a second heart rate sensor.

The LCD code to display the sensor readings from the DHT11, DS18B20, and LM35.

The NTP client code working with the LCD to display the date and time for the health-tracking smartwatch.

Other outcomes include the DF Robot SIM808 working with an Arduino Uno, the DF Robot SIM808 location being displayed in ThingSpeak using a graph, and lastly the hardware interrupt code.

# References

|  |  |
| --- | --- |
| [1] | UN Goals, "https://sdgs.un.org/goals". |
| [2] | Espressif, "ESP32\_Datasheet," [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32\_datasheet\_en.pdf. |
| [3] | MathWorks, "Thingspeak," [Online]. Available: https://thingspeak.com/. |
| [4] | Microsoft, "OneNote," [Online]. Available: https://www.microsoft.com/en-ie/microsoft-365/onenote/digital-note-taking-app?ms.officeurl=onenote&rtc=1. |
| [5] | Draw.io, "Draw.io," [Online]. Available: https://app.diagrams.net/. |
| [6] | W3Schools, [Online]. Available: https://www.w3schools.com/. |
| [7] | Microsoft, "Notepad++," [Online]. Available: https://notepad-plus-plus.org/. |
| [8] | Visual Studio, "Visual Studio Code," [Online]. Available: https://code.visualstudio.com/. |
| [9] | Lighthouse, "Web Server for Chrome," [Online]. Available: https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb. |
| [10] | DHT11, "DHT11 Datasheet," [Online]. Available: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf. |
| [11] | Adafruit, "GitHub DHT Sensor Library," [Online]. Available: https://github.com/adafruit/DHT-sensor-library. |
| [12] | LM35, "LM35 Datasheet," [Online]. Available: https://www.electroschematics.com/wp-content/uploads/2010/02/LM35-DATASHEET.pdf. |
| [13] | DS18B20, "DS18B20 Datasheet," [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf. |
| [14] | OneWire, "GitHub OneWire Library," [Online]. Available: https://github.com/PaulStoffregen/OneWire. |
| [15] | MAX30100, "MAX30100 Datasheet," [Online]. Available: https://download.mikroe.com/documents/datasheets/MAX30100%20DS%20.pdf. |
| [16] | MAX30100, "GitHub MAX30100 Library," [Online]. Available: https://github.com/oxullo/Arduino-MAX30100. |
| [17] | DFRobot, "DFRobot Heart Rate Sensor," [Online]. Available: https://wiki.dfrobot.com/Heart\_Rate\_Sensor\_SKU\_\_SEN0203. |
| [18] | DFRobot, "GitHub DFRobot SIM808 Library," [Online]. Available: https://github.com/DFRobot/DFRobot\_SIM808. |
| [19] | HC-SR04, "HC-SR04 Datasheet," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. |
| [20] | DFRobot, "DFRobot SIM808," [Online]. Available: https://wiki.dfrobot.com/SIM808\_GPS\_GPRS\_GSM\_Shield\_SKU\_\_TEL0097 . |
| [21] | Arduino Uno, "Arduino Uno Datasheet," [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf. |
| [22] | Last Minute Engineers, "Hardware Interrupt," [Online]. Available: https://lastminuteengineers.com/handling-esp32-gpio-interrupts-tutorial/#:~:text=ESP32%20provides%20up%20to%2032,(when%20touch%20is%20detected).. |
| [23] | Last Minute Engineers, "NTP," [Online]. Available: https://lastminuteengineers.com/esp32-ntp-server-date-time-tutorial/. |
| [24] | Arduino Libraries, "GitHub LiquidCrystal Library," [Online]. Available: https://github.com/arduino-libraries/LiquidCrystal. |
| [25] | Espressif, "Espressif ESP32 Pinout," [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/\_images/esp32-devkitC-v4-pinout.png. |
| [26] | MathWorks, "GitHub ThingSpeak Library," [Online]. Available: https://github.com/mathworks/thingspeak-arduino. |
| [27] | "ESP32 Hardware Serial2 Example," [Online]. Available: https://circuits4you.com/2018/12/31/esp32-hardware-serial2-example/. |
| [28] | DiyI0t, "Buzzer," [Online]. Available: https://diyi0t.com/active-passive-buzzer-arduino-esp8266-esp32/. |

# Appendix 1: Code

Here are the codes written during the project.



# Appendix 2: Bill of Materials

ATU stores: ‘y’ or ‘n’ under the ATU Stores column.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Quantity** | **Manuf.** | **Manuf. No.** | **ATU Stores** | **Sourced from** | **Order No** | **Cost Euros** |
| ESP32 | 1 | Espressif Systems | ESP32-DevKitC-32E | y | Mouser | 356-ESP32-DEVKITC32E | 9.4 |
| Arduino Uno | 1 | Arduino | A000066 | y | Radionics | 715-4081 | 18.86 |
| 1K | 4 |  |  | y |  |  | 0 |
| 3K3 | 1 |  |  | y |  |  | 0 |
| 1K8 | 1 |  |  | y |  |  | 0 |
| 220 Ω | 1 |  |  | y |  |  | 0 |
| 330 Ω | 3 |  |  | y |  |  | 0 |
| LED | 1 |  |  | y |  |  | 0 |
| NPN MOSFET | 1 |  |  | y |  |  | 0 |
| Push Button | 1 |  |  | y |  |  | 0 |
| Potentiometer | 1 |  |  | y |  |  | 0 |
| Breadboard | 4 |  |  | y |  |  | 0 |
| DHT11 | 1 | DF Robot |  | y | Radionics | 216-3753 | 7.16 |
| HC-SR04 Module | 1 |  |  | y | Radionics | 215-3181 | 2.93 |
| LCD | 1 | Displaytech | 162KBCBW | y | Radionics | 210-9031 | 11.94 |
| MAX30100 Heart Rate Sensor | 1 | Mikroe |  | y | Radionics | 136-0770 | 21.99 |
| DF Robot Heart Rate Sensor | 1 | DF Robot |  | y |  |  | 0 |
| DF Robot SIM808 GPS/GPRS/GSM | 1 | DF Robot |  | Y |  |  | 0 |
| DS18B20 | 1 | DF Robot |  | Y |  |  | 0 |
| LM35 | 1 |  |  | Y |  |  | 0 |
| MikroElektronika SIM808 | 1 | Mikroe |  | y | Radionics |  | 0 |
| GSM Antenna | 2 |  |  | y | Radionics |  | 0 |
| **Total Cost Approximation** |  |  |  |  |  |  | **72.28** |

Components marked with an asterisk indicate that the student sourced this component themself.

# Appendix 3: Schematic



Diagram

Description automatically generated

Figure Appendix 1 Architecture Diagram

# Appendix 4: Simulation

I used Tinkercad to simulate the buzzer working with an LED in this project, but this was used just to see how a buzzer and an LED would work. In my real circuit, I used an ESP32 instead of an Arduino Uno.

Graphical user interface, diagram

Description automatically generated

Figure Appendix 2 Buzzer and LED Circuit Simulation

Text

Description automatically generated

Figure Appendix 3 Buzzer and LED Circuit Simulation Code