OPIOID OVERDOSE DETECTION

GNG1103

Design Project User Manual

Submitted by:

Linea Vitam, Group 12C

Abdullah Abdulmajeed

Alyssa Wang

Antonia Zupu

Spencer Henry

Yomna Elsahli

April 19th, 2020

University of Ottawa

To: Prof. David Knox, Ms. Rani Damuluri, Mr. Selameab Demilew

This report serves to be a clear guide to use, maintain and reproduce the LifeLine project.

Abstract

The goal of this project was to create a discrete, non-invasive device that users can wear to detect opioid overdoses. The device must send the user's GPS location to a pre-set emergency contact when an overdose is detected. This report will contextualize readers to the details of the opioid epidemic and the depth of the impacts it has had on the different subsections of society. It will also introduce LifeLine, a device created by the group *Linea Vitam* that promises to be a convenient and comfortable wrist mounted gadget that can sufficiently solve the problem at hand. It can measure oxygen saturation using a MAX30100 chip and sync these measurements to an app installed on the users own smartphone via bluetooth. In the event of overdose the app will then alert an emergency contact. This report is also an exhibit of observations and documentations collectivised from prototypic experiments of the LifeLine device throughout the stages of development, as well as a comprehensive blueprint of the final design including directive instructions to replicate the models of the design delving into each of the subsystems and their features. It also outlines a complete manual of use and maintenance on everything that a user might need to know. Lastly, it will offer insights to future developers that might want to replicate or improve on the model. LifeLine is a realistic and viable option as it was designed with the user in mind during every step of the process. It was made to be discrete as the notion of being labeled as a 'drug user' might by itself steer people away due to stigma. Comfort was one of the top priorities. The adjustable straps on the flat boxy wrist mounted design allow for the device to be comfortably worn for long periods of time without compromising the device's ability to take accurate measurements. The redundancy of fail-safe measures and the phone app that was designed with user experience in mind, all of this and more is what puts LifeLine ahead.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
PCB	Printed Circuit Board
PLA	Polylactic Acid
USB	Universal Serial Bus
DB	Database
AIA	Adobe Illustrator action data file
APK	Android Package File
CPR	Cardiopulmonary Resuscitation

Table 1. List of Acronyms

1. **Introduction**

This document explains the design and function of the opioid overdose detecting device, LifeLine. This first section explains the reason why this device was designed. The second section breaks down the device into several subsystems, and explains how each subsystem is built. The third section explains how our device is used, while the fourth section discusses the risks associated with this device.

The opioid overdose crisis is an ongoing issue that calls for emergency measures. Communities are trying to find ways to combat it, and have come up with supervised consumption and overdose prevention sites where people can be monitored while they safely use. However, the stigma against opioid users is so strong that many users are not comfortable being seen at these consumption sites. These are the users our device is made for. People who do not look like a 'stereotypical addict', such as doctors, lawyers, and especially blue collar workers. Because these people cannot go to these sites, the rate of opioid overdose for these workers is the highest. The purpose of this device is to lower this rate.

With the device, LifeLine, the user will be able to wear a watch-like device discreetly on their wrist. With our design, the user can comfortably wear the device without drawing attention to themselves.

The device, LifeLine, is designed to detect opioid overdoses. The wristband collects the user's blood-oxygen levels and displays them on the "LifeLine" android application. In the event that the user's blood oxygen level is detected to be less than 90%, the device takes this as a warning of an overdose. First, a failsafe is activated and a phone notification appears, asking the user if they are okay. In the event of a false reading, the user will be able to press the "Yes" button and the alarm protocol will cease. However, if no response is detected in 30 seconds, the application will consider this as confirmation of an overdose and, in response, a loud alarm will be activated and the user's GPS location will be sent to a preselected emergency contact.

LifeLine gives users a stress-free way of keeping them safe. The failsafe assures that users will never have to worry about false alarms occuring, and the alarm will increase the chance of the user being helped in the event of an overdose, thereby increasing their prospect of survival. With LifeLine, users have a second chance at life.

2.How the Prototype is Made

The final device has been split into certain components as they were made separately before being put together. Firstly, the mechanical portion describes the exterior device frame. Secondly, the electrical portion which encompases the circuit with all of the components was described. Lastly, the software of the device is explained. This includes the smart phone application and coding for the device..

In all of the sections, there is a description of its importance to the final device then the BOM, so what materials had to be bought to create the device. Next the list of tools or equipment needed was listed. Lastly the description on how to create the final prototype for this section was explained with details so that future developers can re-create the device and add their own adjustments or improvements as they please.

Information such as material consideration, calculations and all important decisions were included with each section on how the prototype was made. With all of this material, anyone with this project should be able to fully re-make this device.

2.1 Mechanical

The mechanical portion of the device encompases the external device frame. This part of the apparatus not only holds all of the electrical components together but makes it durable and functional for daily use. It is important that the device frame includes all of the key attributes that it needs as it is what the user and client first see along with functionality.

To make the device aesthetically pleasing, it has a sleek design with smooth rounded edges. On the inside, the hole was made rounded to optimize the space as much as possible, but then includes a rectangular shaped hole so that there are corners cut out to fit the PCB board snuggly to avoid it shaking around inside the device. The frame includes holes for the charger and switch on the walls of the bottom piece. Then on the floor of the same piece there is a hole for the sensor to stick out and be in contact with the wearer's skin. Grooves around the hole for the sensor had to be added in order for there to be full contact between the skin and sensor. If the grooves were not added the pins and soldered pieces would block the sensor from going through the hole by holding the component up by the pins against the wall of the device frame. This would break the sensor to skin contact and cause inaccuracy in the readings. Next, the watch strap pieces, which are a simple tube design, are incorporated to have something to attach the strap onto.

This device would be 3D printed as it is a costly effective way of constructing the exterior frame and would also be most precise with the tools that we had available. The most

common material used in 3D printers is PLA, which is a durable and light plastic. It was efficient as it was a strong enough material to keep the device from breaking easily and light enough to be comfortable and not increase the extra weight to the already added load from the electrical components. The PLA material was not chosen as it was the only option for 3D printing without extra charge, so it was a part of the mechanical aspect that was settled on. However, the material was never a problem once we had decided to glue the device shut.

Deciding to glue to the device shut was also an option that we settled on due to time restrictions. There was difficulty finding a way to open and close the device with hinges or sliding the lid onto the bottom piece as all the opening and closing mechanism designs were either too bulky, taking away from the discreet look, or were too precise or small to create using a 3D printer.

A recommendation is for the device to be printed in two parts, the bottom piece with all the holes and strap holders and the simple sleek lid, which is a basic model of the bottom. This is to maximize the time spent on the device by reducing the time printing. Lastly, the adjustable watch strap would be attached to the printed device and once all the components were inserted into the device and it was proven to work adequately, the device lid would then be glued shut to the bottom piece.

However, due to the COVID-19 pandemic, we were unable to obtain a watch strap, add the components to the device and shut it as our work timeline was abruptly cut short. Those would have been the next steps for the mechanical aspect of the final device. We also couldn't test the comfort of the final weighed device as well as how compact the PCB board would fit inside the device frame to make sure that it didn't shake around, due to the electrical timeline being cut short as well. The results of these tests could have possibly caused changes to the final device frame if they were not functional.

2.1.1 BOM (Bill of Materials)

Most of the mechanical aspects of our device cost little to no money. This is due to 3D printing being one of the many free services in the Makerspace at the University of Ottawa. The only cost concern that occurred for this section was the watch strap.

Listed in the table below are the best options of watch straps that we had to look into.

Options	Model	Price(CAD$)
#1	Quick Release Leather Watch Band	$16.99
#2	Black Silicone Rubber Watch Strap With Tool For G-Shock Watch	$5.00
#3	Garmin Vivosmart HR+ watch strap (black)	$6.00

Table 2. Watch strap options and their costs.

With more time, we would have chosen option #2 as it is the most cost effective and a good quality strap. The link to the second strap option can be found in the appendix at the end of the report.

2.1.2 Equipment list

When building this device frame the following tools are needed to help construct and design it:

- A 3D printer and with PLA material
- A computer with access to the application tinkerCAD to create the design
- A computer with the application CURA downloaded to be able to upload the design to the printer
- Glue to shut the device once all the components are inside

To have the device built, the materials bought, which are listed in the BOM, were also needed along with the tools used to build it. This includes:

- The black silicone rubber watch strap

2.1.3 Instructions

Creating the exterior device frame is fairly simple. There will be three sets of instructions should you choose to download the pre-designed device frame and print that or design yours from scratch using the instructed techniques. If the chosen route is to download the pre-made design, it can be found in the MakerRepo site, the link is in Appendix I. The last set of instructions is what to do with the frame once it is printed.

2.1.3.1 Downloading the Design

1. Download the CURA application onto your computer
2. Download the pre-made design as an STL file onto your computer
3. Upload the STL file into the CURA application
4. Avoid changing the size in the CURA application as it is already designed to fit all of the component pieces perfectly
5. Slice the component to be sure it can be printed and have an idea of how long it will take to print the design
6. Then click save to file and export the design from CURA onto a ultimaker memory card
7. Lastly place the card into the printer and press print

2.1.3.2 Creating the Design

1. Open up the tinkerCAD application where you will be working
2. Use the rounded cube shape as the basic shape of the device and make it 75x60x30mm
3. The cut the shape in half horizontally and copy the cut shape to have two of them

4. Change the size of one of the shapes to be 70x55x15mm then hover it 2mm above the ground and place it in the center of the other shape, then turn the smaller shape into a hole and group the shapes together
5. Next get a cube of 63x48, height just must be over 11. Have it hover over the ground 5mm. Place it in the center of the shape, turn it into a hole and group the shapes together.
6. Copy and paste the shape to have two of them

The device should look something like this:

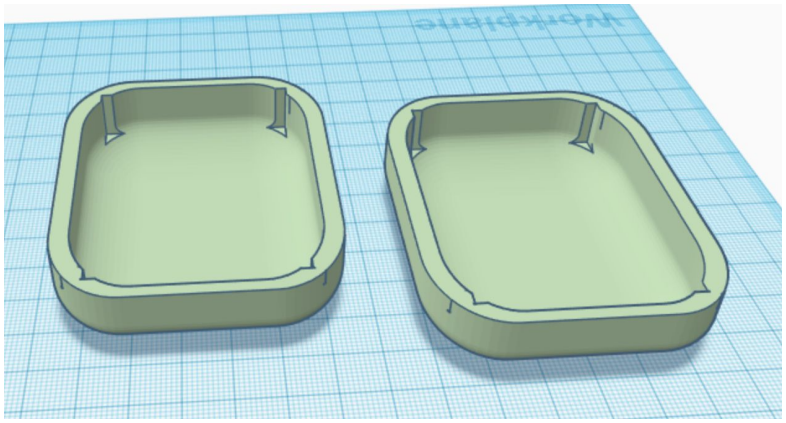


Figure 1. Basic shape of device frame.

The nexts steps are where more personal choices can be made.

7. Measure the switch, charger and sensor for the holes in the device frame. The dimensions are as followed:
    - Sensor: 12x5mm (length x width, height just must be larger than 3mm)
    - Charger: 7.5x2mm (width x height, length just must be larger than 5mm)
    - Switch: 10x5 (width x height, length just must be larger than 5mm)
8. Create holes on the wall of one of the two designs for the switch and charging port. We have chosen to place the holes in the center of the shorter sides of the basic device shape.
9. On the same design, create a hole for the sensor on the bottom of the device. We have placed it in the middle closer to the top of the device.

The device should now look like this:

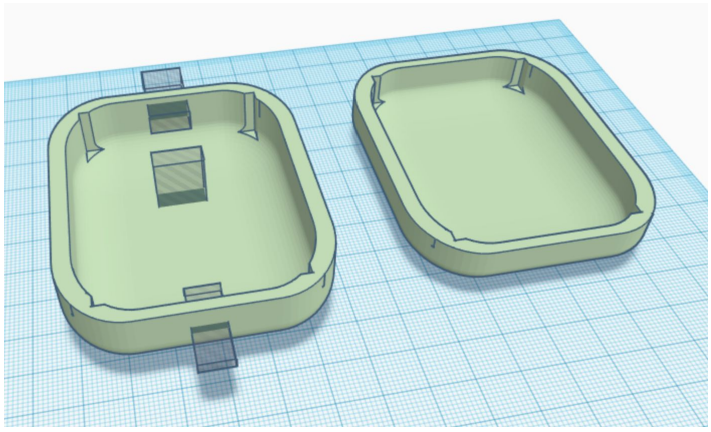


Figure 2. Holes shown as “hole blocks”.

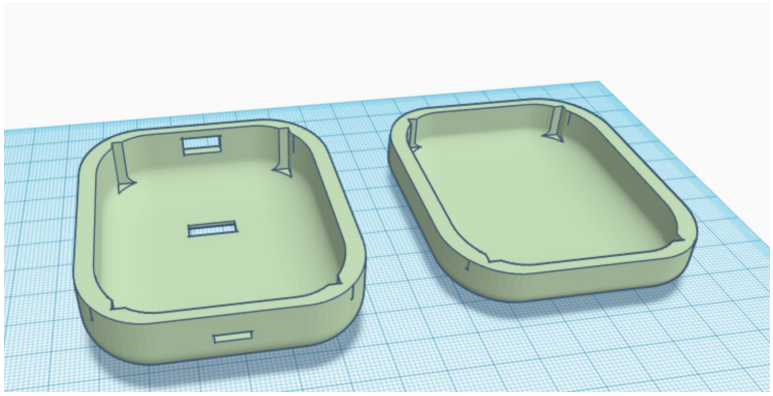


Figure 3. Device frame with holes.

10. Measure the dimensions of the pins and soldering on the sensor component around the sensor then create grooves around the sensor for them to sit into.
    The dimensions we used was a 20x20mm block, where the heigh just had to be over 5mm, that hovered just over the floor into the device frame to allow for that section of the frame to be as thin as possible. The block was placed over the sensor hole to surround the hole. Then turn the block into a hole and group it with the device
11. For the strap holder, get the tube shape and increase it so that the size is 28mm in diameter, then you may make the tube thinner to your liking by adding another smaller tube as a hole inside of it.
12. Cut the tube in half and stretch it slightly to make it fit the average watch strap
13. Place each half of the tub to the center lower part of the bottom piece device frame (this is the one with the holes and grooves)
14. Lastly group all the pieces together

The final product should now look like this:

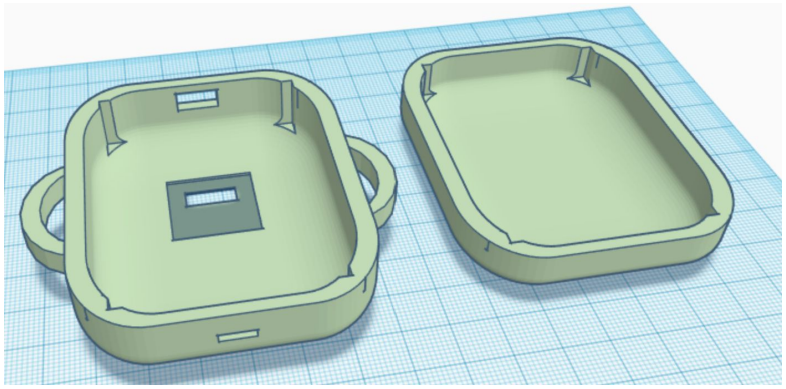


Figure 4. Final device frame.

15. The next steps are to export it from tinkerCAD as an STL file then using the steps in 2.1.3.1, you can print it using CURA and a 3D printer.

2.1.3.3 Final Steps

1. Remove the excess PLA material from the adhesion and supports. Adhesion and supports help 3D printers create designs without them moving or falling over during printing.
2. Add the straps onto the device frame
3. Carefully insert the device components inside the device frame
4. Glue the device shut

2.1 Electrical

The electrical component of this project was fairly uncomplicated. Connections were easily set up for data transfer between the Arduino microcontroller and the rest of the sub-systems (Oximeter and Bluetooth). However, there were certain complications pertaining to a discrepancy between voltage requirements of the different subsystems. Furthermore, a compromise was made to ensure that the device could be worn discreetly and comfortably by minimizing the size of the battery at the expense of lowering voltage supply to the circuit. Electrically speaking, Prototype III is essentially made up of two circuits.First, the inner circuit comprising the HC-05, MAX30100 and the Arduino nano. The team needed to take the components off the breadboard and fit it inside the device frame but because CEED facilities were closed, and the team did not have access to the facilities needed for PCB circuit implementation, an alternative method of circuit connections was attempted. The final circuit is not fixed to anything and is connected through wires only. Resistors are also connected across these wires to supply power to SDA and SCL pins on the MAX30100 chip. The oximeter is fully functional and so is the HC-05 bluetooth communication module if the Arduino is supplied with sufficient power through a USB drive or a 5V battery. Secondly, the outer circuit that is made up of the battery, the Powerboost 1000C, and a power switch. The Powerboost was an instrumental part of this design not only because it offered a convenient way to charge the battery but it also contained the micro booster. A microbooster or otherwise called boost converter is a simple circuit made from an inductor, a mosfet switch and a capacitor. This circuit uses induction technology to run current through alternating paths using the mosfet switch which quickly opens

and closes. When it's open the current is forced to flow one way to a capacitor that charges and quickly discharges much in the same way as a battery would. Except it discharges at a rate much faster than a battery would such that the voltage applied is boosted.

The Powerboost we received seems to be defective, as it would heat up when connected to anything so we decided to not risk frying our circuit by connecting it. As a result, we were unable to include the battery into the circuit as it only produced 3.7v which is not sufficient to run the circuit without the Powerboost converter. Therefore, the outer circuit is not functional. We also have a power switch that would have been connected in series to the power wire coming out the Powerboost and into the Arduino. The power switch would break the circuit and shut off the current going into the Arduino. It would be there so that the user can fully turn off the device throughout the day when the device is not in use. Development of the device stalled before completion in the last weeks since under the pretext of the COVID-19 crisis CEED was closed. CEED facilities provided services and equipment that was imperative at that point. Under normal circumstances had we been able to access the resources we needed. We could've run the process to completion in terms of replacing the defective Powerboost chip and putting all the components together onto a PCB.

2.2.1 BOM (Bill of Materials)

Attached Below, is a complete list of electrical components used to construct the final LifeLine prototype along with their respective costs.

Material	Price (CAD$)
MAX30100	$12.00
Arduino Nano	$7.00
HC-05 Module	$11.00
Powerboost + 1000mAh Battery	$43.09
Wires	$1.46
4.7kΩ Resistors	$0.00
Total Estimated Price:	$74.55

Table 3. Tabulation of the complete bill of electrical materials.

2.2.2 Equipment list

- Solder and soldering iron (soldering kit)
    - A soldering kit is necessary because the components come in with the header pins unsoldered.

- Breadboard
    - A breadboard may be optional depending on the prefered method of connection.

2.2.3 Instructions

When finishing up the project, the circuit should ideally be set up on a PCB since that might be the best basis to create a compact product with reliable connection. However, regardless of the method of connections chosen, be it through wires like prototype III or a breadboard. The connections are still the same. Attached below is a complete guide to make the necessary connections and a figure with the connections made on a breadboard.

2.2.3.1 Inner Circuit.

1. Solder the header pins onto the Aduino nano, the MAX30100 and HC-05 chips
2. Connect GND from the arduino nano to GND on the MAX30100 oximeter chip and GND on the HC-05 bluetooth chip.
3. Connect SDA, SCL pins from the MAX 30100 to the Arduino nano's A4, A5 pins respectively
4. Use 2X 4.7k Ohm resistors to connect 5V power from the Arduino Nano 5V pin to each of the SCL and SDA MAX30100 pins in series.
5. Connect RX, TX pins on the HC-05 chip to TX1 and RX0 pins respectively on the arduino.
6. Run 5V power from the 5V pin on the arduino to the VCC pin on the HC-05 chip and VIN on the MAX301000 chip.

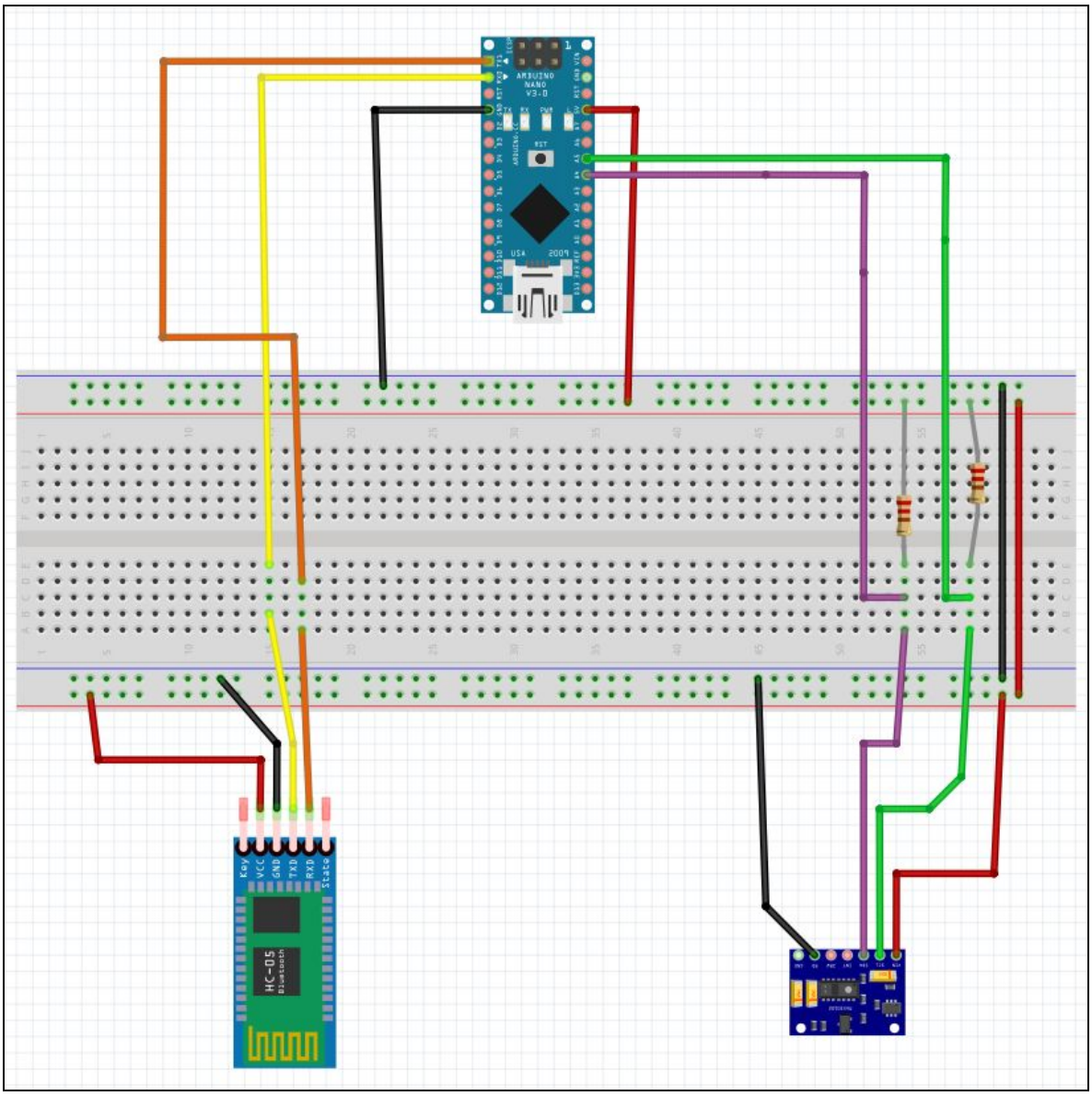


Figure 5. Electrical connections of the inner circuit on a breadboard.

2.2.3.2 Outer Circuit.

1. Solder the header pins onto the power boost 1000C chip.
2. Connect the GND pin that's adjacent to the 5v pin on the Powerboost to the GND pin adjacent to the Vin pin on the Arduino nano.
3. Connect the 5v pin on the Powerboost to the power switch.
4. Connect the other end of the power switch to the Vin pin on the Arduino nano.
5. Connect the black wire from the battery to the GND under the Bat pin on the Powerboost chip.
6. Connect the red wire from the battery to the Bat pin on the Powerboost chip.

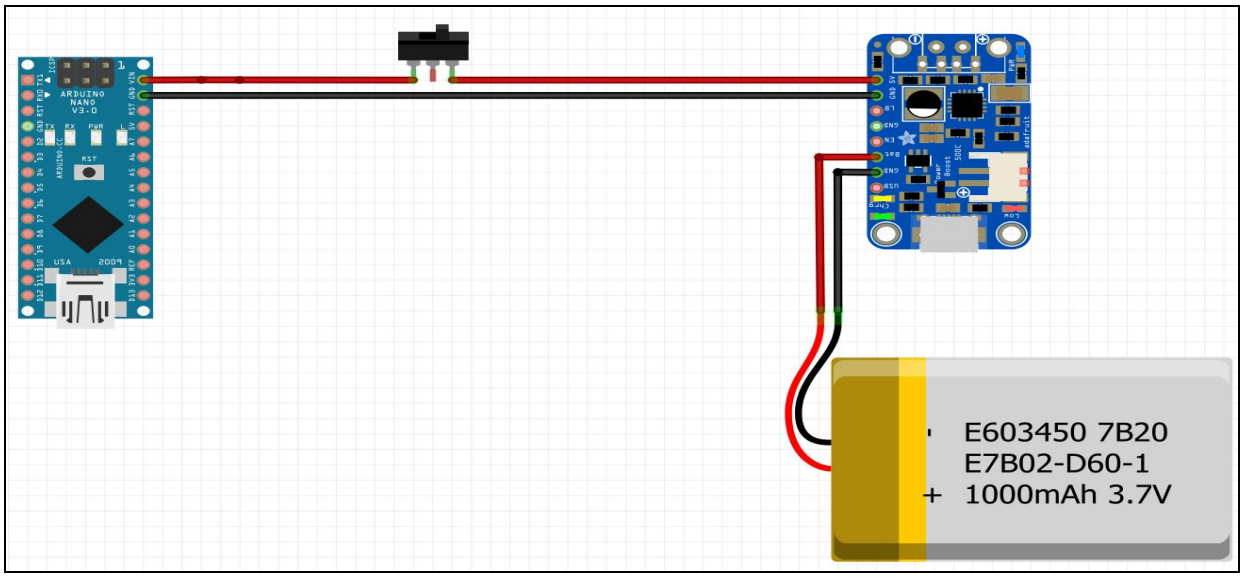


Figure 6. Electrical connections of the outer circuit.

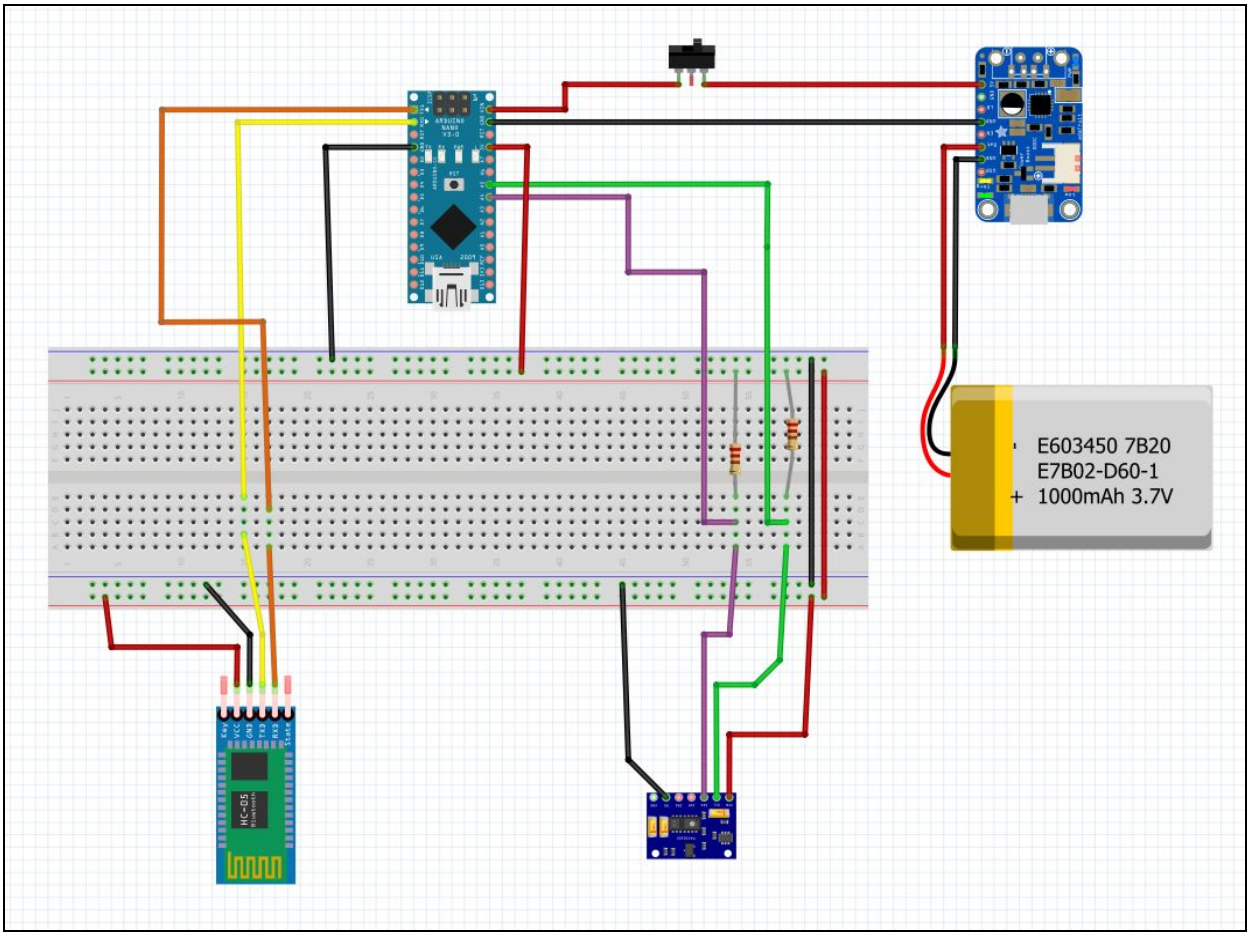


Figure 7. Electrical connections of the integrated LifeLine device on a breadboard.

2.3 Software

2.3A – MIT App Inventor

2.3A.1 BOM (Bill of Materials)

The entirety of the software we created did not require spending of any sort. The application creator we used, *MIT App Inventor,* is completely free for anyone to use, and MIT App Inventor's companion application for Android phones is completely free as well. Many of the application makers we wanted to use were free as well such as Android Studio. We had android phones already available, and an emulator that we could use to test our code if we didn't have our phones on us, so everything we needed for the creation of our application was readily available and cost us nothing.

2.3A.2 Equipment list

- Access to a computer or mobile device that can access MIT App Inventor
- An Android Device with the MIT App Inventor Companion app installed
- A working Arduino Pulse Oximeter and HC-05 for testing the application

2.3A.3 Instructions

MIT app inventor is separated into 2 sections, the designer and blocks. The designer shows how the screen will look like on the device and the blocks is where the code that controls all the app functions is made. The code is created with an interlocking block format where when the components are added in the design section, they each will display their own set of features that could be used in the code in the blocks section. In the appendix section, we will be providing a link to a google drive folder with an AIA file for users to download onto MIT App inventor and an APK file for users to download straight to their phones using the MIT App inventor Companion app.

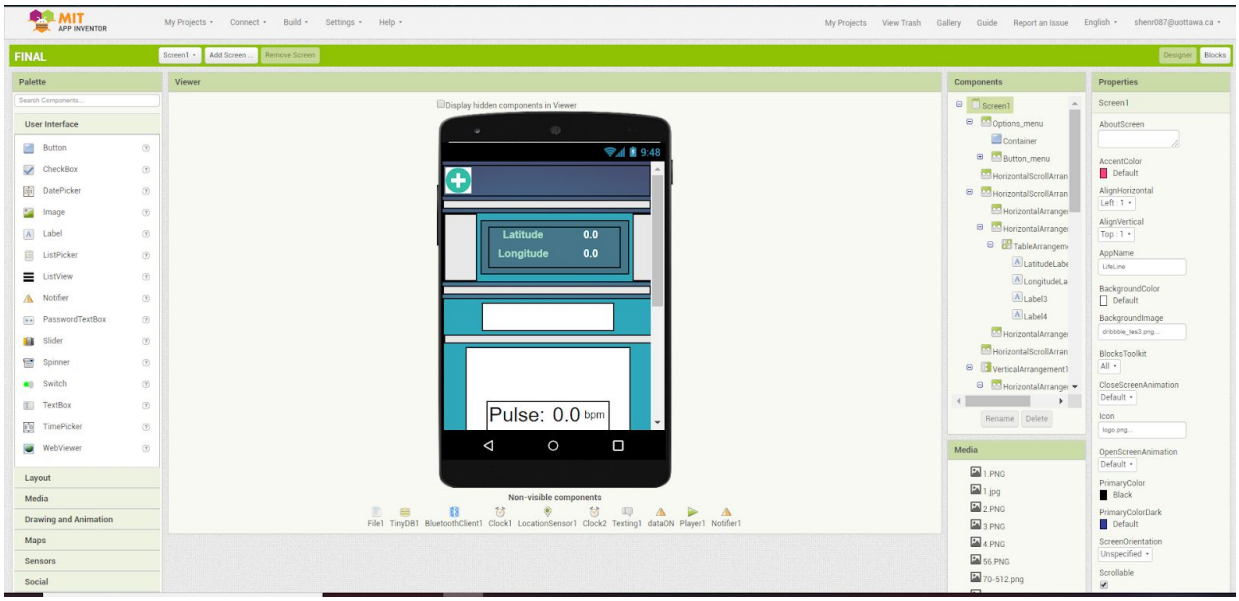


Figure 8. The Designer Section.

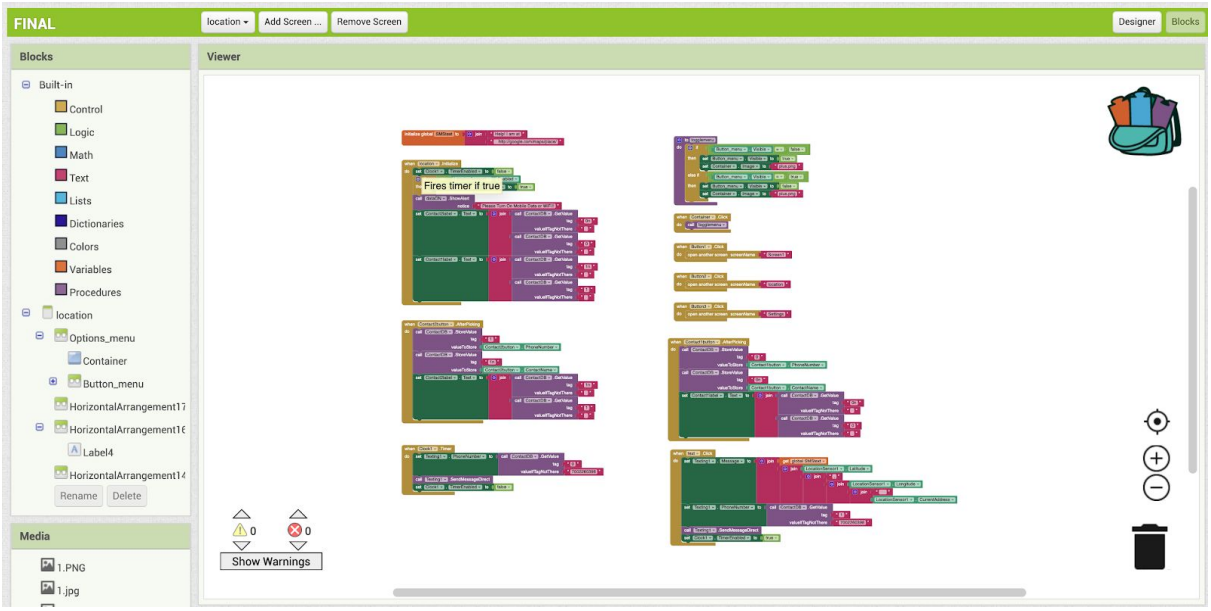


Figure 9. The Blocks Section.

2.3.3A.1 Main Screen.

The main screen, screen1, shows bluetooth connectivity options, displays pulse oximeter values, and controls the emergency failsafe alarm. This part will focus on these features since additional features like the toggle buttons will be described in the following screen explanation. For the design section, components need to be added by dragging the feature into the display device screen. The following components are needed:

- User interface category
    - 11 labels (longitude, latitude, pulse, spO2, connection status, etc)
    - List picker (when button is clicked, options for bluetooth connections will show)
    - Notifier (creates notification alert)
- Media category
    - Player (will play uploaded sound for failsafe alarm)
- Sensors category
    - 2 clocks (will set notification timer and allow app to receive bluetooth data)
    - Location Sensor (for showing longitude and latitude)
- Social category
    - Texting (for messaging emergency contact)
- Storage category
    - File (for reading stored data)
    - TinyDB (stores data in app)
- Connectivity category

- Bluetooth Client (allows app to connect to bluetooth)

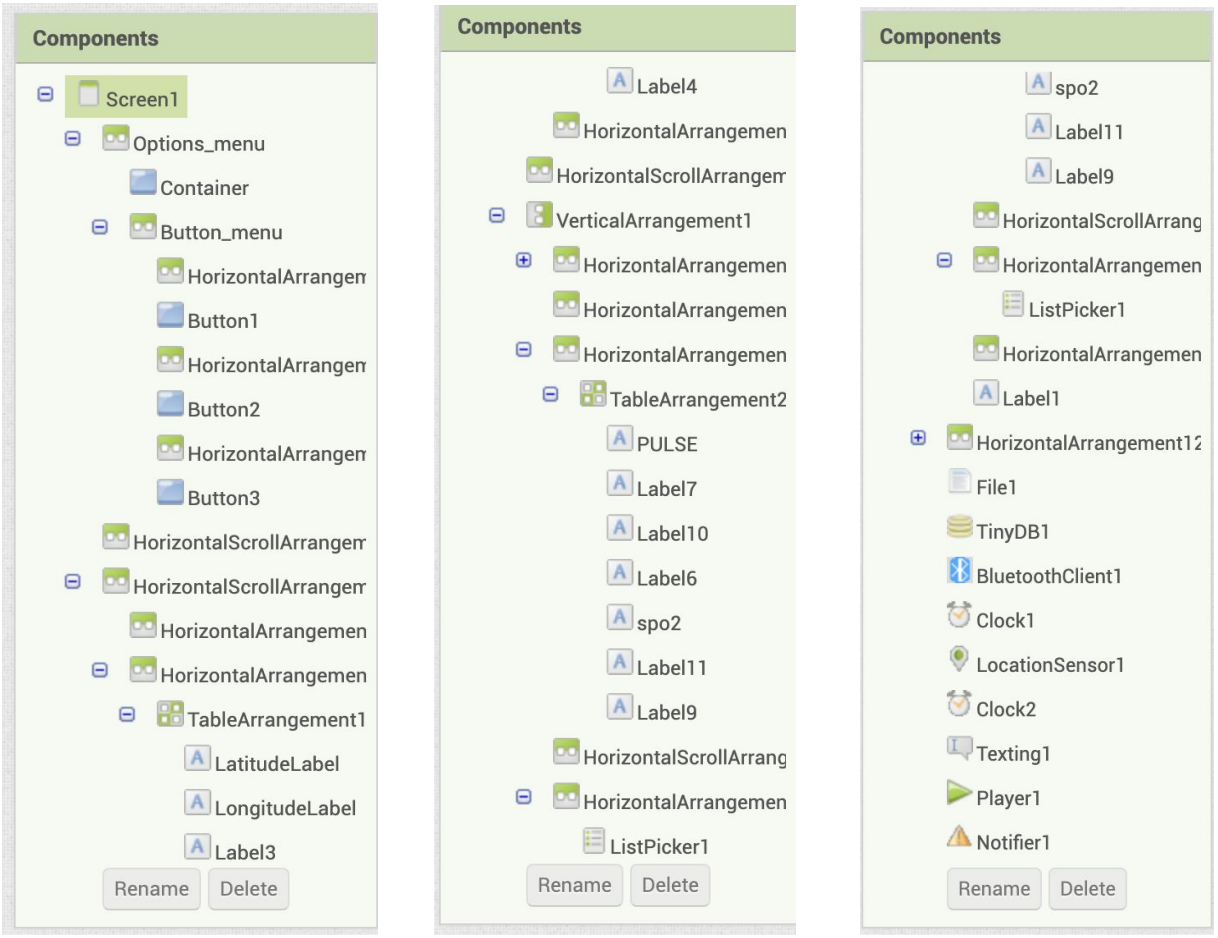


Figure 10. (a), (b), (c) Organization of components in the design section for the main screen.

These can be organized and customized in order to create the user interface desired using horizontal arrangements as well as different colour palettes, shapes, and text fonts.

Component changes required

- Both clocks must have their "TimerAlwaysFires" and "TimerEnabled" settings on and have the "TimerInterval" set to 100 milliseconds.
- The bluetooth client's "DelimiterByte" value must be adjusted to 10.
- The alarm sound file must be uploaded under the subheading "source" and the "loop" feature must be enabled. The link for the alarm sound used can be found in the appendix.

Main Screen Block Code Instructions

1. Select the "BeforePicking" listPicker1 block and set the list picker elements to the bluetooth client address and names. This will retrieve all the addresses and names of nearby possible bluetooth connections

2. Select the “AfterPicking” listPicker1 block and create an “if or else” statement which states that if the app is connected to a bluetooth device then the label that says “connection status” will change to “connected”
3. Use the location sensor “LocationChanged” block to change the longitude and latitude labels displayed on the screen by setting each label to the global longitude and latitude variables which will update once the user’s location is detected
4. Initialize 4 global variables called input, list, warning, and counter. Set the first 2 to a blank text string, the third to false, and the last to 0
5. Select the clock1 “Timer” and make it consist of the blocks stated in steps 6-15
6. Add an “if or then” statement. The “if” condition should check if the bluetooth client is connected and if the bytes available are greater than 0 (data is being received)
7. The “then” portion should include the blocks from steps 8-10
8. The bluetooth bytes received should be assigned to the global variable input
9. The input variable is then split at the “,” and this modified data is assigned to the list variable
10. The list variable is defined as separate information where 1 is the pulse data and 2 is the spO2 data. This is done by the “select list item list” block and equating it to either text labels so that it can be updated instantaneously
11. Another “if or then” statement is made after the last one. The “if” condition should check if the bluetooth is still connected and if the retrieved spO2 values are less than 90%
12. In the “then” section, set the warning variable to be true and the counter variable to increase by 1
13. Add another “if or then” statement. The “if” condition should determine if the warning variable is equal to true and the counter variable is equal to 1
14. The “then” section will create an alert by using the notifier1 “ShowMessageDialog” block. Make the message ask, ”Are you having an overdose?”, the title be “Warning!!” and the button text say “I am not having an overdose”.
15. Under all of the “if or then” statements, add a file1 “AppendToFile” block to allow the storing of the data. Set the text to the input variable and call the file name “heartdata.txt”
16. Initialize 2 more global variables called timer and response and make them respectively equal to 300(30 seconds) and 0.
17. Select the clock2 “Timer” and make it include the blocks stated in steps 18-21
18. Add an “if or then” statement and put the warning variable to true and the timer variable to be greater than zero in the “if” condition.
19. Decrease the timer by 1 in the “then” statement.
20. Include another “if or then” statement following the last one where the “if” condition has the timer and response variables equal to 0.
21. The “then” statement should call the player1 to start and make the player1 loop equal to true.

22. Add the notifier1 “AfterChoosing” block and make it consist of blocks in steps 23-25
23. Make an “if or then” statement where the “if” condition checks if the choice variable is equal to “I am not having an overdose”. This choice variable correlates to the option that the user picks when the notifer1 alert is displayed.
24. Make another “if or then” statement embedded within the last one and have the “if” statement check if the timer variable is less than 300.
25. If so, the “then section” will change the response and timer variables so that they are respectively 1 and 0. Also, it will call player1 to pause.

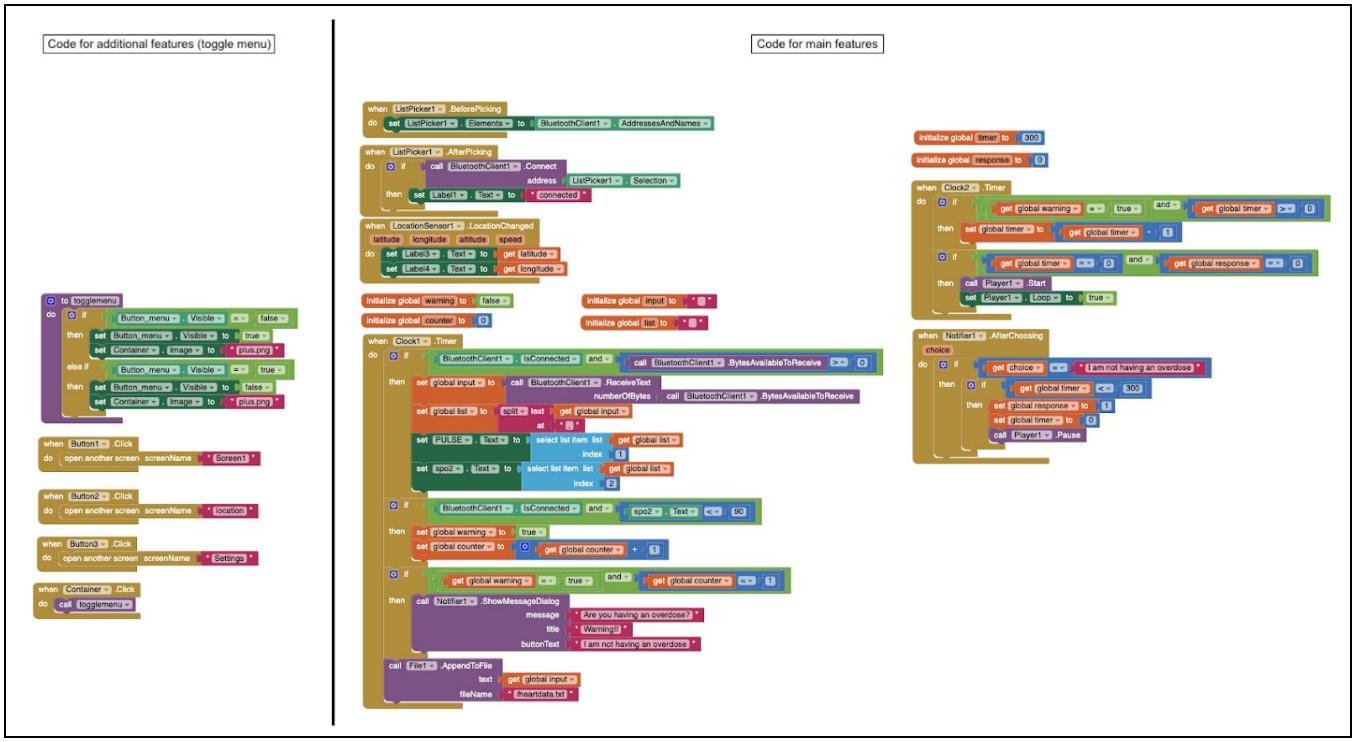


Figure 11. Main Screen Coding Blocks.

2.3.3A.2 The Contacts Menu

Figure 12. The Contacts menu we will be recreating

The Contacts Menu will allow the user to input up to 2 save contacts that will receive emergency texts from the user. The following components will be needed:

- User interface category
    - 2 text boxes
- Layout category
    - 1 Table Arrangement
    - 1 Horizontal Arrangement
- Sensors category
    - 1 clock
- Social category
    - 1 Texting
- Storage category
    - TinyDB

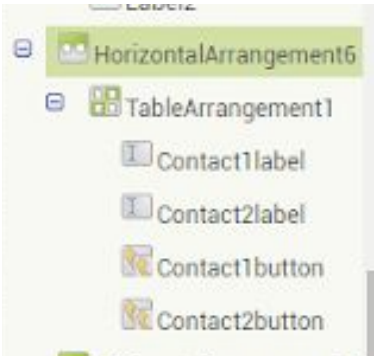


Figure 13. What the Components will look like on the sidebar

Instructions on how to make a Contacts Menu

1. To begin working on this feature, we will first set up the properties of the screen. Click on the Component of the main screen you are working on (Mine will be called location) and make sure you have the following settings:
    - AlignVertical - 1
    - AlignHorizontal - 3
    - TitleVisible - Unchecked
    - Play around with the BackgroundImage, BackgroundColor, ScreenOrientation, and CloseScreen/OpenScreenAnimation Properties to get the menu you would like

2. Now that we have the main screen setup, we will not add a horizontal arrangement to the screen. This arrangement will be the space where our contacts menu will be placed. Make these changes to the arrangements properties:
    - AlignHorizontal - 3
    - AlignVertical - 2
    - Height - 200 Pixels
    - Width - 320 Pixels
    - Visible - Checked
3. Next, we will add a Table Arrangement inside the horizontal arrangement. This table will contain the buttons and labels for the Contacts Menu. Make these changes to the arrangements properties:
    - Columns - 2
    - Width - 2
    - Height - 150 Pixels
    - Width - 280 Pixels
    - Visible - Checked
4. Next, add text boxes and buttons inside the table arrangement so it looks like this:

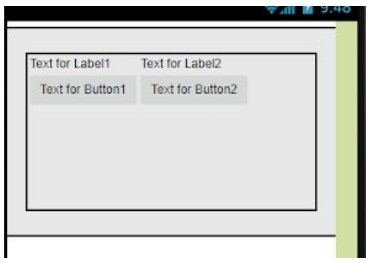


Figure 14. An example of what the screen should look like at this stage. Ensure that the buttons and text boxes are inside the table and not just in the horizontal arrangement.

5. We are now going to make these changes to the text boxes:
    - Height - 50 pixels
    - Width - 135 pixels
    - Rename the left box Contact1label, and the right box Contact2label
    - Hint - Phone Number
    - TextAlignment - 1
    - Make sure Enabled and Visible boxes are checked
    - Change font sizing , bolding etc to your liking

    And these changes to the buttons:

- Height - 50 Pixels
- Width - 135 Pixels
- Rename the left button Contact1button and the right button Contact2button
- Text - Contact 1 (for left button), Contact 2 (for right button)
- TextAlignment - 1
- Make sure Enabled and Visible are checked
- Change font sizing, bolding etc to your liking

6. Add a Texting1 component from the social palette, a Clock1 component from the sensor palette and make the Timerinterval - 2000, and a TinyDB component from the storage palette that you will rename ContactDB
7. Adjust the colours to your liking and readjust the alignments and sizes to your liking and you now have a Contacts Menu designed! Now, we have to create the blocks that will get the menu to actually work. Go to the Blocks screen on MIT App Inventor and grab a "when Screen1.initalize" block (this block will come from the main screen component, we kept ours named as screen1 but you may have changed it to something else). Add a "set Clock1.TimerEnabled" block from the Clock1 components and a "false" logic block to the "do" part of the "when Screen1.initialize" you have.
8. Grab an "if then" control block, attach a "not" logic block to the "if" part of that control block, and add what you have to the "do" section of the "when Screen1.Initialize" block.

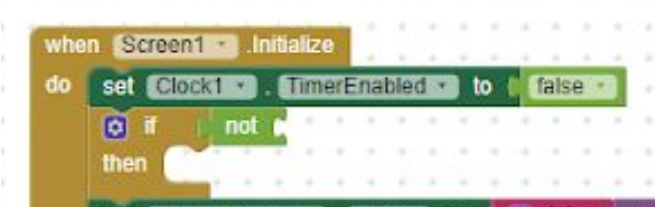


Figure 15. What you should have so far for Screen1

9. Under the "if then" block, add a "set Contactlabel.Text to" from both Contact1label and Contact2label. Add a "join" text block to both "set Contactlabel.Text" blocks.
10. Go to the ContactDB blocks and grab 4 "Call ContactDB.Getvalue / valueiftagnotthere" procedure blocks and add them to the ends of the "join" texts blocks.
11. Add text blocks with specific texts to the ends of the "tag" and "valueIfNotThere" of the procedure blocks like so:

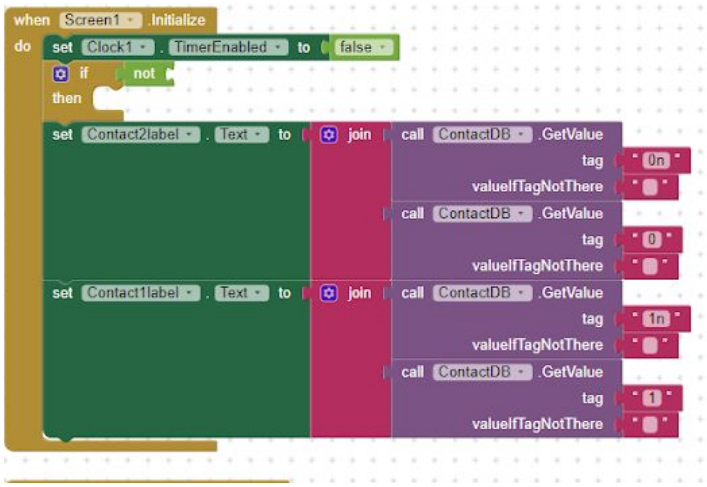


Figure 16. What the Screen1 block should look like at the end

12. Now that the screen1 block is done, we move on to a new block to create. Go to the Contact1button blocks and select a "when Contact1button.AfterPicking" block. Add Two "call ContactDB.StoreValue / valueToStore" Procedure blocks underneath each other from the ContactDB blocks to the Contact1button block. Underneath the two contactDB blocks, add a "Contact1label.Text" block.

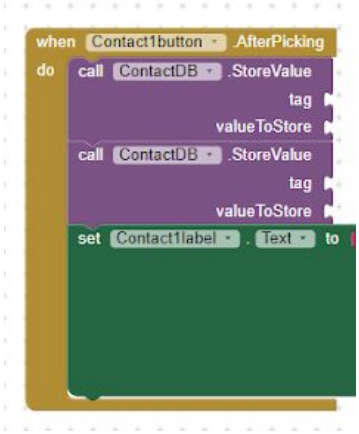


Figure 17. What the Contact1button block should look like so far (1)

13. Add a "join" text block to the "Contact1label.Text" block. Go to the ContactDB blocks and grab 2 "call ContactDB.GetValue / valueIfTagNotThere" blocks to the "join" text block.
14. Go to the Contact1button blocks and grab blocks so everything looks like this:

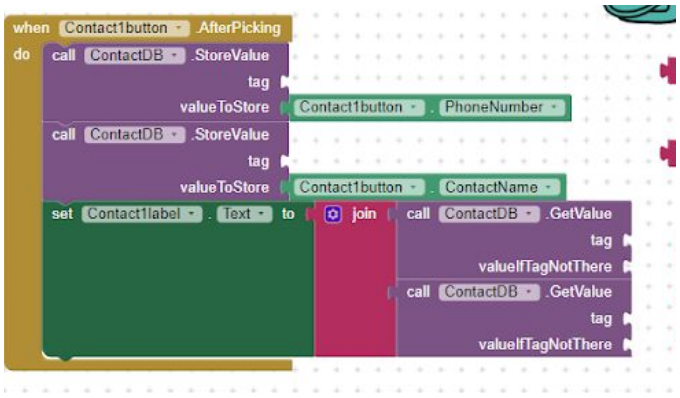


Figure 18. What the Contact1button Block should look like at this stage (2)

15. Add "_" text blocks and rename some of them so the final stage can look like this:

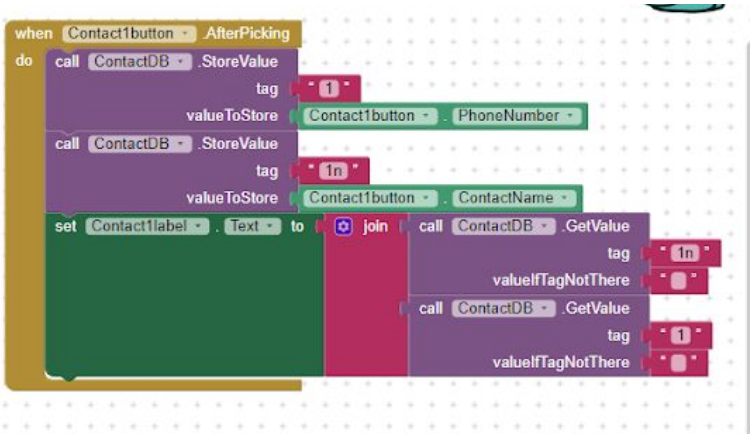


Figure 19. What the Contact1button block should look like (3)

16. Now that the Contact1button block is done, we move on to a new block to create. Go to the Contact2button blocks and select a "when Contact2button.AfterPicking" block. Add Two "call ContactDB.StoreValue / valueToStore" Procedure blocks underneath each other from the ContactDB blocks to the Contact2button block. Underneath the two contactDB blocks, add a "Contact2label.Text" block.

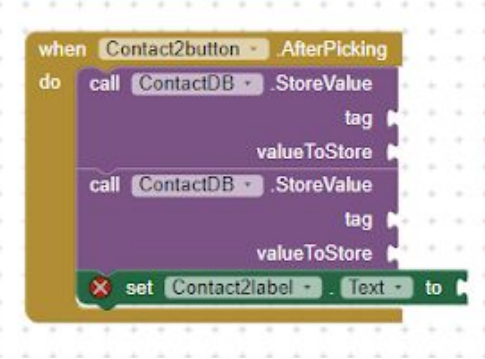


Figure 20. What the Contact2 block should look like at this stage (1)

17. Add a "join" text block to the "Contact2label.Text" block. Go to the ContactDB blocks and grab 2 "call ContactDB.GetValue / valueIfTagNotThere" blocks to the "join" text block.
18. Go to the Contact2button blocks and grab blocks so everything looks like this:

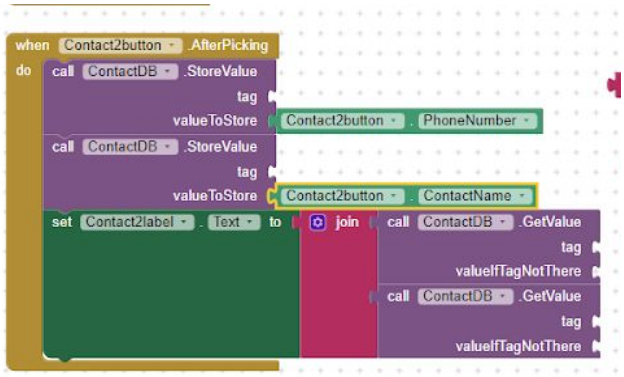


Figure 21. What the Contact2button block should look like at this stage (2)

19. Add "_" text blocks and rename some of them so the final stage can look like this:

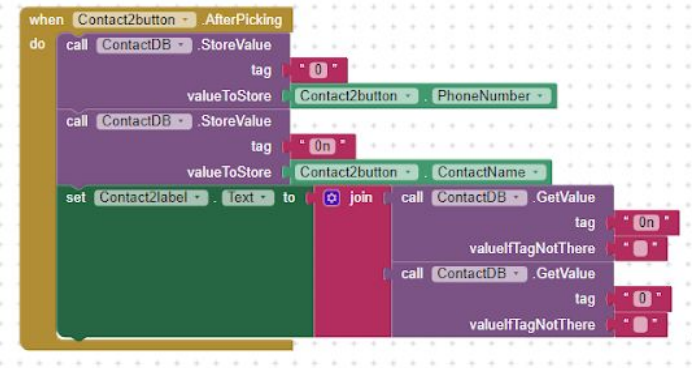


Figure 22. What the final stage of the Contact2button block should look like (3)

20. Now that this big block is done, we move on to the final block to create. Go to the clock1 component's blocks and grab a "when Clock1.Timer" Block.
21. Go to The Texting1 component's blocks and add a "set Texting1.PhoneNumber to" block and a "call Texting1.SendMessage" block inside the Clock1 block. Underneath the "callTexting1" block you just added, go to the clock1 Component's blocks again and grab a "set Clock1.TimerEnabled" block. You're block should look like this so far:

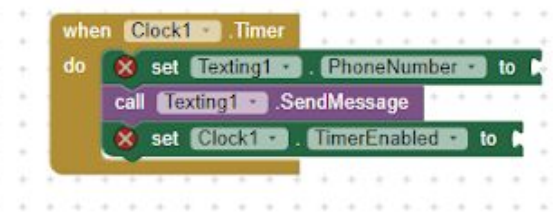


Figure 23. The current state of the Clock1 block (1)

22. Add a "false" logic block to the "set Clock1.TimerEnabled" block. Next, go to the ContactDB component's blocks and add a "call ContactDB.GetValue / valueIfNotThere" block.

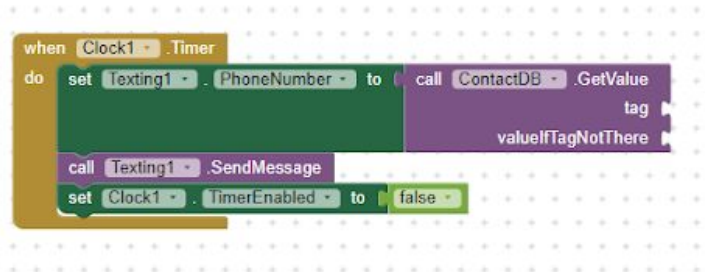


Figure 24. What the Clock1 block should look like so far (2)

23. Finally, add "_" text blocks and rename them so they look like this:

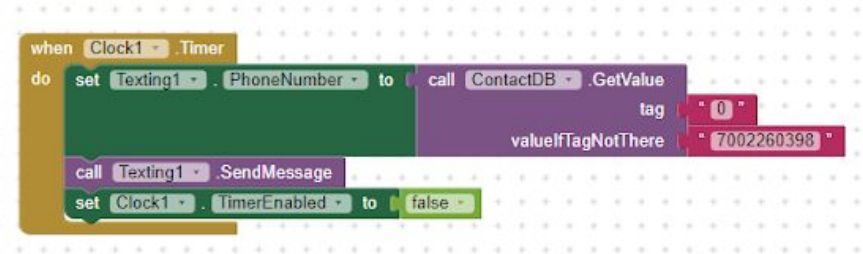


Figure 25. The final design of the block (3)

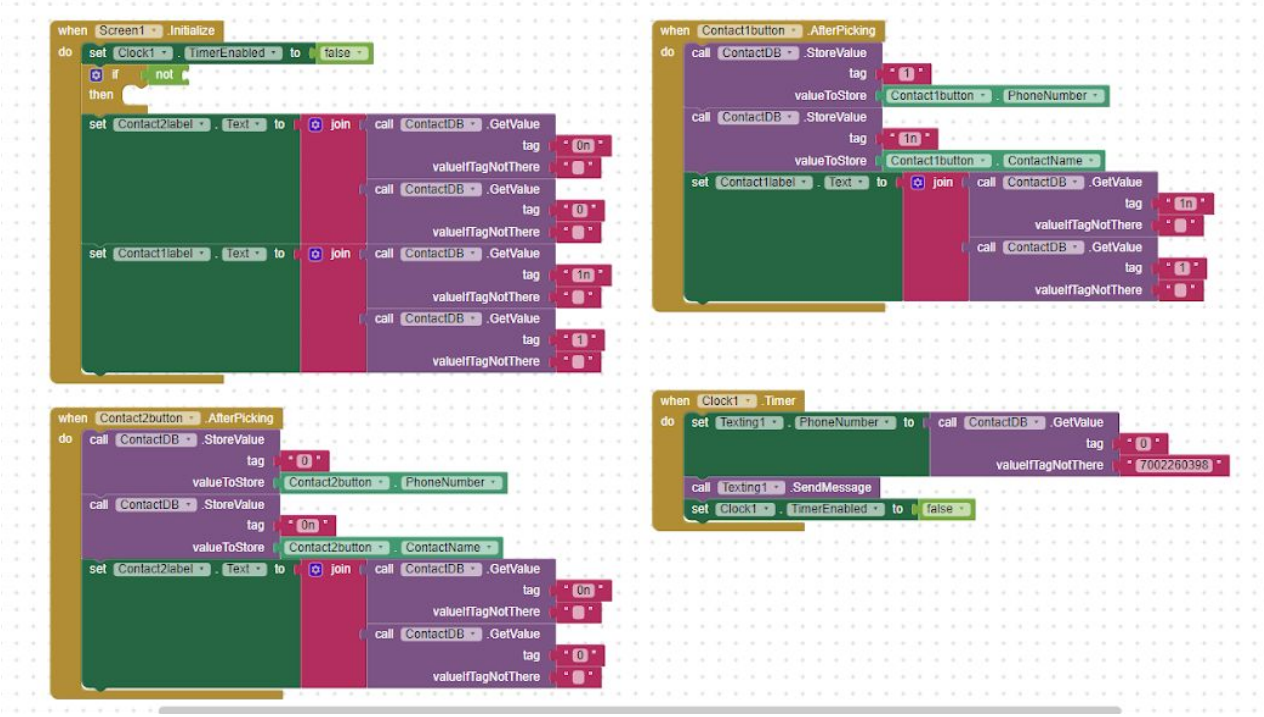


Figure 26. The final creation of all the Code Blocks for the Contacts Menu

2.3.3A.3 The Toggle Menu

Figure 27. Our example for what the toggle menu will look like on a finished application

The toggle menu will be the main feature that will allow users to navigate around the application and use the many features being implemented into the application. This is a very easy menu to create and a simple and clean menu to use. For the design of this section, the following components will be needed:

- User interface category
    - 4 Buttons
- Layout category
    - 5 Horizontal layouts

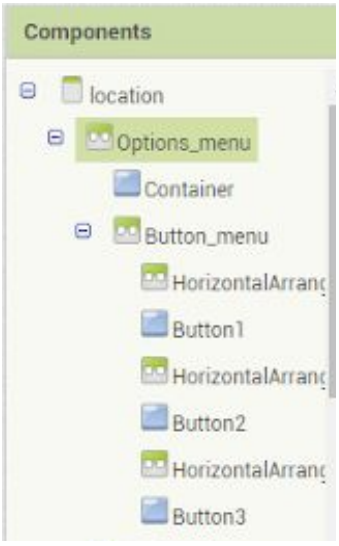


Figure 28. Organization of components being used to make the toggle menu

Instructions on how to create the Toggle Menu

24. To begin working on this feature, we will first set up the properties of the screen. Click on the Component of the main screen you are working on (Mine will be called location) and make sure you have the following settings:
    - AlignVertical - 1
    - AlignHorizontal - 3
    - TitleVisible - Unchecked
    - Play around with the BackgroundImage, BackgroundColor, ScreenOrientation, and CloseScreen/OpenScreenAnimation Properties to get the menu you would like
25. Now that the main screen is set up, we will now add the first horizontal Arrangement. This arrangement will be the space where your menu will show. Make these changes to the arrangements properties:
    - Rename the component “Options_menu”

- Align Horizontal and Align Vertical - 1
- Height - 50 pixels
- Width - either Fill parent or 100 percent

26. Next, add a button inside of the first horizontal arrangement (Options_menu). This button, when pressed, will open the menu and allow other buttons that lead to other screens to be pressed. Make these changes to the buttons properties:
    - Rename the component “Container”
    - Align Horizontal and Align Vertical - 1
    - Height and width - 45 pixels
    - Add an image of your choosing that will represent the container. The preset height and width will ensure the image stays at a suitable size for the menu.
    - Make sure Enabled and Visible boxes are checked.
27. Beside the Container button you just created, add a new horizontal arrangement. This arrangement will be the space where your buttons will appear on the screen. Make these changes to the arrangements properties:
    - Rename the arrangement to “Button_menu”
    - Height - Automatic
    - Width - Fill parent
    - AlignHorizontal/Vertical - 1
    - Make sure the visible box is NOT checked. This will give you the toggle effect once the code blocks are created.
28. Finally, inside the Button_menu arrangement you created, add in order: A Horizontal Menu, followed by a button, followed by a Horizontal menu and so on. The buttons being added will, when pressed, allow the user to navigate to other screens, and the arrangements being added will space out the buttons so they look nicer. For the manual, we will use 3 buttons and 3 arrangements. Make these changes to the Arrangements:
    - AlignHorizontal/Vertical - 1
    - Height - automatic
    - Width - 20 pixels

    And these changes to the buttons:

    - Height/Width - 45 Pixels
    - Make sure Enabled and Visible boxes are checked
    - Add images of your choosing to represent the different buttons you have
29. Adjust the colours to your liking and readjust the alignments and sizes to your liking and you now have a toggle menu designed! Now, we have to create the blocks that will get the menu to actually work. Go to the Blocks screen on MIT App Inventor and grab a “to _, do” block from Procedures blocks. Rename it “togglemenu”.

30. Grab an “if then” block from Control blocks and modify it (by pressing the blue settings button on the block) so it becomes an “if _, then_, else if_, then_” block. (By adding an “else if” to the “if” block that pops up on the viewer, the block will change to the desired block).

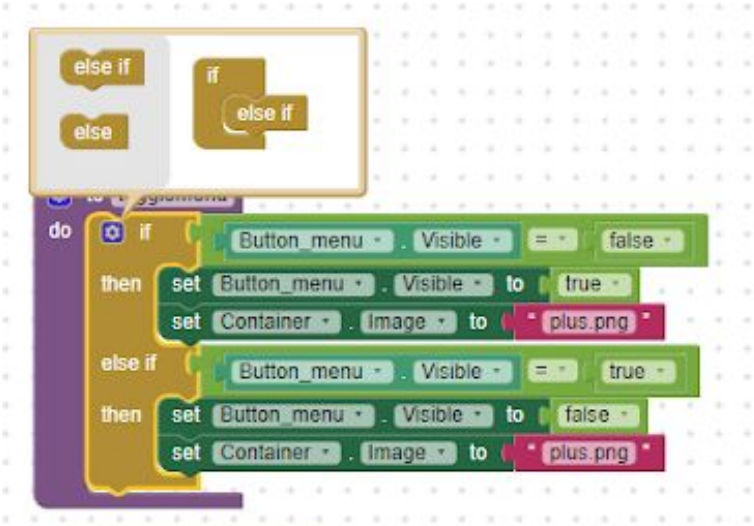


Figure 29. What pressing the blue settings button does and what the if block should look like in the viewer

31. Add 2 “ _ = _” blocks from Logic blocks and add one to the if and else if parts of the block on the viewer. Add a “false” logic block to the if statement, and a “true” logic block to the else if statement. Go to the “Button_menu” arrangement you created in the blocks section, click on it and grab 2 “Button_menu.Visable” blocks and add them to the “_=_” blocks on the viewer.
32. For the “then” statements, we are going to add a “Button_menu.Visible” arrangement block, and a “Container.image” button block to each of them. Add a “true” logic block to the first “Button_menu.Visable” arrangement block, and a “false” logic block to the next one. Next, go to text blocks and add a “_” block to the “Container.image” button blocks on the viewer. Rename the “_” blocks to the name of the image you added to the Container button in the designer stage of MIT app inventor. (For example, we used plus.png, so we typed plus.png to the “_” block).
33. In the blocks list, go to the 4 buttons you should have: Container, and buttons 1-3. Grab the “when _.Click” blocks from each of the buttons and place them neatly away from each other.

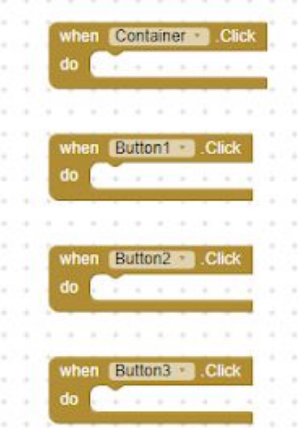


Figure 30. The "when_.Click" blocks placed neatly

34. For "Container.Click", go to the Procedures buttons and add a "call togglemenu" button to the "Container.Click" block.
35. For buttons 1-3, go to the Control buttons and add a "open another screen screenName" button to each of the buttons blocks. Attach an "_" text block to the ends of the "screenName" Control blocks. Inside those texts boxes, add the individual names of the screen you wish to send the user to in each of the "_" blocks.

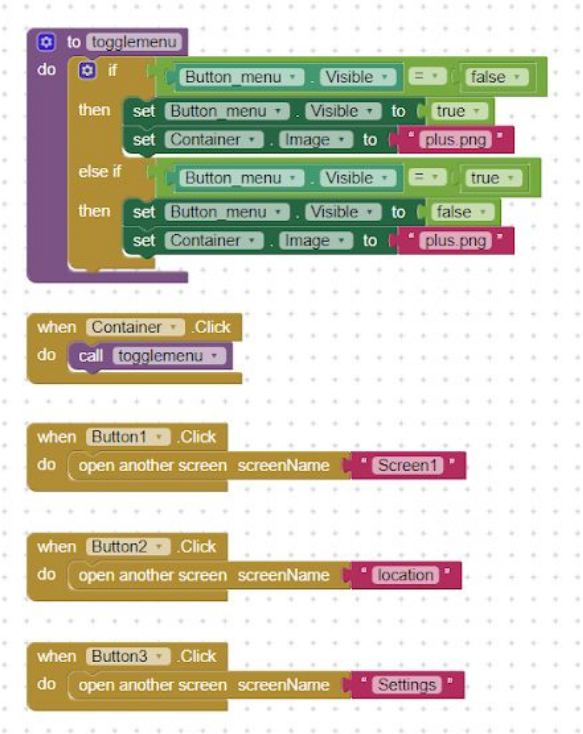


Figure 31. Visual of what the Code Blocks should look like for the toggle menu

2.3B – Arduino Code

2.3B.1 BOM (Bill of Materials)

The arduino code did not require any materials to create, as it only needed the Arduino IDE software application to write the code in, as well as a device to download that software onto to use, such as a laptop. Writing the code only requires minimal materials, but to ensure the code works and that blood oxygen readings are being sent from the MAX to the arduino nano, to the bluetooth module and displayed on your app, the inner circuit must be built. To construct this sub-circuit, you need the following materials: wires, MAX30100 chip, arduino Nano, 4.7kΩ resistors, HC-05. (Refer to section 2.2.3.1 for circuit assembly). It is also possible to observe blood oxygen levels on the serial monitor of the Arduino IDE software, instead of setting up the bluetooth connection using the HC-05.

Material	Price (CAD$)
MAX30100	$12.00
Arduino Nano	$7.00
4.7kΩ Resistors	$0.00
HC-05	$11.00
Wires	$1.46
Total Estimated Price:	$31.46

Table 4. Tabulation of the complete bill of the software code materials.

2.3B.2 Equipment list

- Device to run Arduino IDE software
- Arduino IDE Software

Optional:

- MAX30100 Chip
- 4.7kΩ Resistors
- Arduino Nano
- Wires

2.3B.3 Instructions

The first step in constructing the arduino code is to download the MAX30100 library. There are several versions available online (Libraries used can be found in Appendix II).

The MAX30100 library allows you to access functions that utilize the MAX chip. This library contains functions such as the spO2 calculator, which is what is needed for this project. You will also need two other libraries. To establish a bluetooth connection, the SoftwareSerial library as it is necessary for serial communication on any digital pins. Lastly, the Wire library is needed for sending and receiving data over a net of devices or sensors.

The code was made to be as simple as possible as MIT made all the higher level logic decisions. The purpose of the code is to get the heart rate and blood oxygen levels and send them to our LifeLine app.

```
#include <Wire.h> // Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.
#include "MAX30100_PulseOximeter.h" //MAX30100 library that contains all functions.
#include <SoftwareSerial.h> //Bluetooth connection library.

SoftwareSerial lifeLine(0, 1); //initializes bluetooth connection.
//Numbers (0,1) correspond to RXT an TXT pins.

#define REPORTING_PERIOD_MS     1000
PulseOximeter pox;//initializes pulse oximeter function.

uint32_t tsLastReport = 0; //set to 0 initially to be used in the loop later on.

// Callback (registered below) fired when a pulse is detected


void setup()
{
    Serial.begin(9600);
    lifeLine.begin(9600);

    // Initialize the PulseOximeter instance
    // Failures are generally due to an improper I2C wiring, missing power supply
    // or wrong target chip
    if (!pox.begin()) {
        Serial.println("FAILED");
        for(;;);
    }
    else {
        Serial.println("SUCCESS");
    }

}

void loop()
```

```
void loop()
{
  int pulseox;
  int heartrate;
  //Variablea to save pulse ox & heartrate readings in.
    pox.update();//Calls for an update

    // Asynchronously dump heart rate and oxidation levels to the serial
    // For both, a value of 0 means "invalid"
    if (millis() - tsLastReport > REPORTING_PERIOD_MS) //checks if the last report made was more than 1 second ago.
    {
        Serial.print("Heart rate:");
        Serial.print(heatrate=pox.getHeartRate());//Uses a function from the MAX library to get heartrate.
        Serial.print("bpm / SpO2:");
        Serial.print(pulseox=pox.getSpO2());//Uses a function from the MAX library to get blood oxygen level.
        Serial.println("%");
        if(lifeLine.available()) //Checks to see if the bluetooth connections is available first.
        {
          lifeLine.write(pulseox);//Sends pulseox to bluetooth.
          lifeLine.write(heartrate);//Sends heartrate to bluetooth.
        }

        tsLastReport = millis();//saves the time of the last reported reading as the current time.
    }

}
```

Figure 32. (a) Part One of the Arduino Code (b) Part Two of the Arduino Code

3. How to Use the Prototype

The device is very simple for the user to operate and consist of two parts, the device and phone application. The final device should be worn on the wrist with a close fitting so that the sensor is flush against the skin and is not moving around. The adjustable watch strap will allow for the user to be comfortable and accommodate for their wrist size.

There is an easy to use toggle on and off switch to optimize battery life and we will recommend users to keep the device on only when it is necessary. This will help to reduce risks of false readings and reduce the burden on the user regarding recharging the device. Nevertheless, without this precaution the battery life should last for approximately 24 hours just in case the user forgets to turn it on, then they can keep it on for the whole day and turn it off at night. If the device runs low on battery, it can be charged using a micro USB cable via the Powerboost.

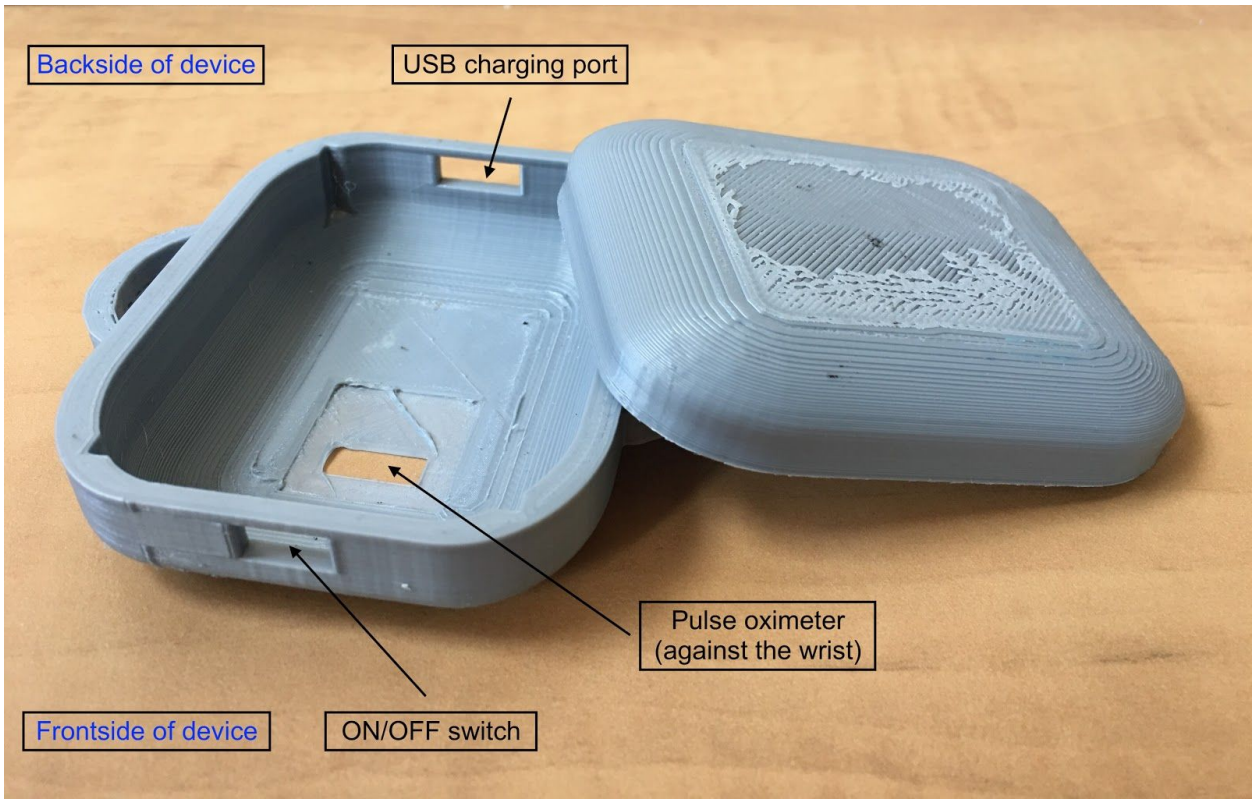


Figure 33. Display of the location of all of the components.

The next part needed is the phone application. Prior to using the device, the user must download the application, LifeLine. At the moment, it is not an official app in the app store and has to be downloaded off of a *MIT AI2 Companion*. This process involves downloading *MIT AI2 Companion* and scanning a QR code that will allow you to download the LifeLine app. However, if this product were to be officialized, the app would be uploaded on the *Google Play Store* and thus only require the user to search the name and download it. Once downloaded, it is very important that the user ensures that all location and text notifications and permissions are allowed for the application. The main screen shows the longitude and latitude coordinates of the user's smartphone device. Underneath, it displays the user's heart rate and spO_2 readings, which are being measured by the device on the wrist and instantaneously updated. At the bottom of the screen, there is a button where the user can use bluetooth to connect the app to the device, which is called "HC-05". When an overdose is detected, the phone will send an alert asking the user if they are having an overdose. The user must respond within 30 seconds otherwise an alarm will go off repeatedly to alert nearby help and can be turned off when the notification button saying, "I am not having an overdose" is clicked.

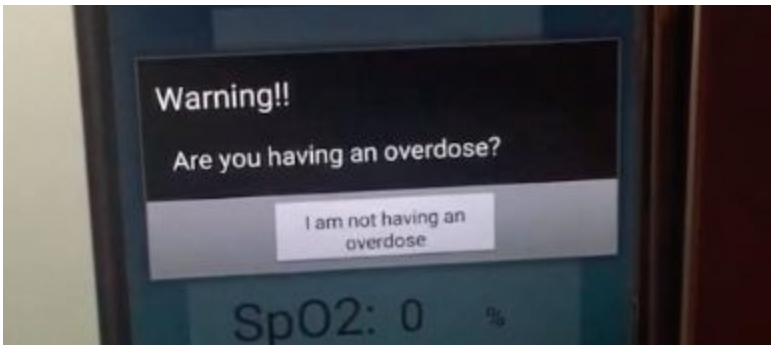


Figure 34. Failsafe system for the application.

Next, the user needs to ensure that two contacts are filled out so that help can be alerted in the case of an emergency. This is done by pressing on the contact one and contact two buttons which will then lead the user to the contacts app on their phone and allow them to pick people. The user should aim to pick a person who is reliable since if an overdose does occur, they will receive a text with your location and the faster they respond, the greater chances of the user's survival. Unfortunately, the app requires internet or cellular data to be connected since otherwise the location might not be properly sent. This is because in order to make it easier for the emergency contact to determine the user's location, our app sends a clickable google maps link instead of longitude and latitude coordinates. This is much more convenient, however, results in the user needing the internet in order for it to determine the location of the coordinates prior to the message being sent. A notification telling the user to turn on their wifi or cellular data will appear on the emergency contacts screen in order to remind the user of this.

(a) (b)

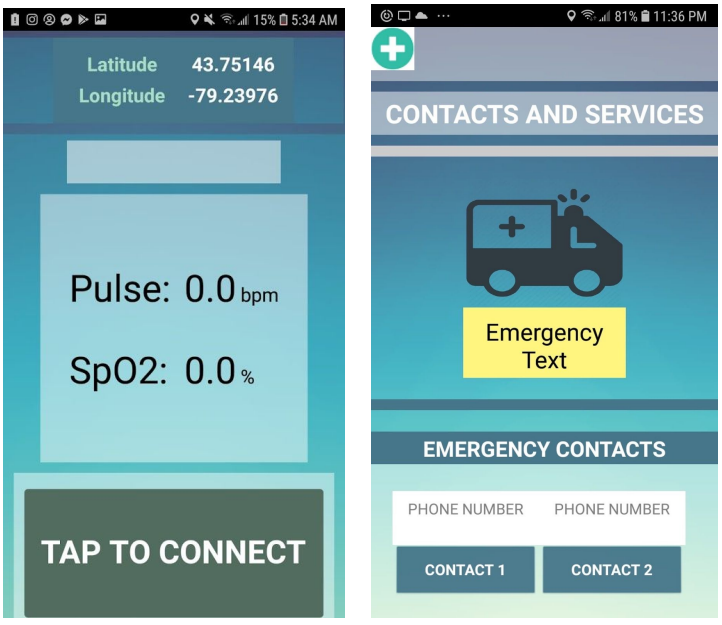
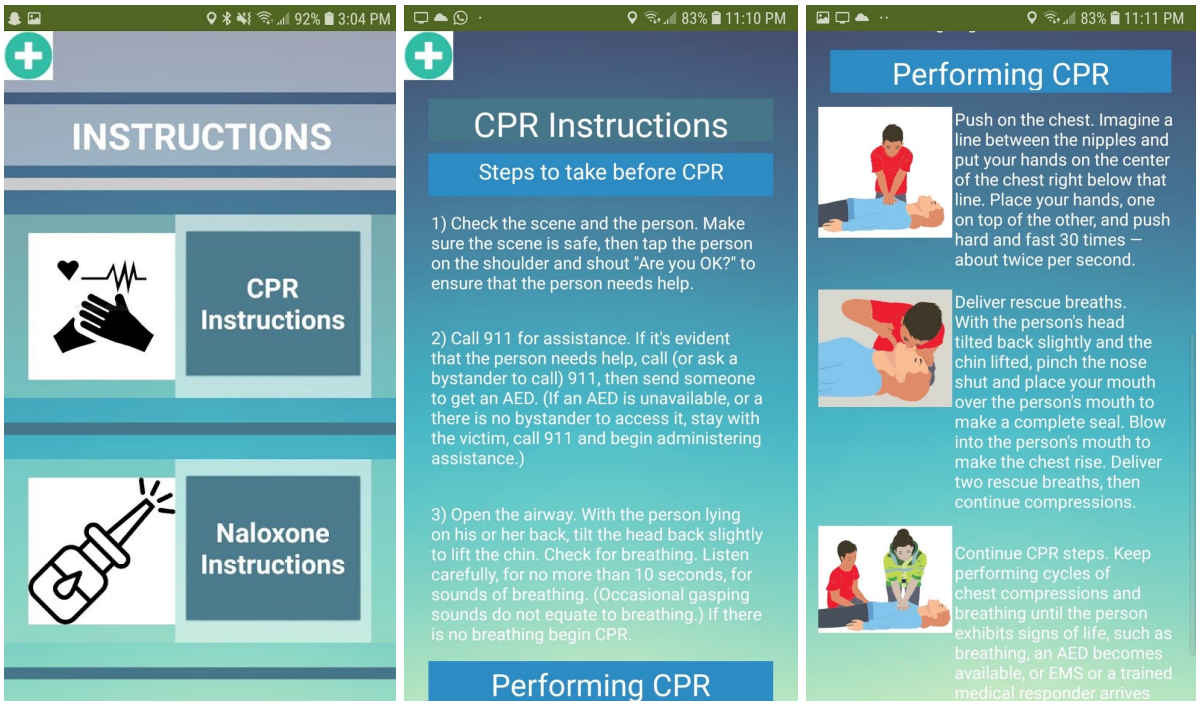


Figure 35. (a) Application main screen and (b) emergency contact screen.

Moreover, there is a section with instructions on how to do CPR and distribute naloxone to the user should the alarm successfully alert nearby help. If the user has naloxone on them, the help will have greater chances of distributing it effectively with the thorough and visual instructions given. Three different types of naloxone distribution are provided, naloxone spray, syringe, and auto-injectable. This will help increase the survival rate of the user since bystanders are the most effective source of help, especially within the approximately 3 minute window before the overdose causes death.

(a) (b) (c)



(d) (e) (f)

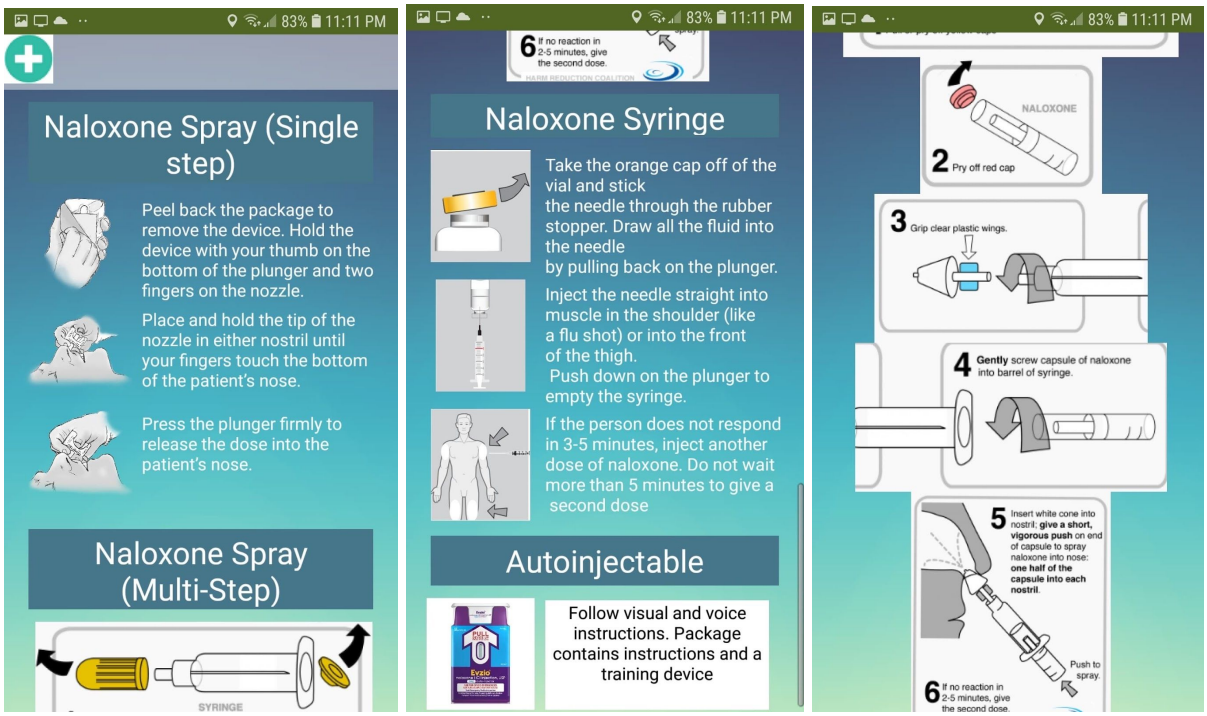


Figure 36. (a) Application settings safety instructions screen, (b), (c), (d), (e), (f) CPR and naloxone instructions screens.

A video representation of the application and all its functions can be seen online. It is uploaded on the *MakerRepo* website and *Youtube*. The link for each can be found in the appendix. Along with having the device, we recommend that the user carries a naloxone kit with them as well since it is the fastest and most effective way to save their life.

4. How to Maintain the Prototype

In order to assess the accuracy and durability of the device, several tests were planned to be completed. When success was made with the circuit, the first thing we tested was the accuracy of the MAX30100 chip results by comparing them to a real pulse oximeter at different locations such as the index finger and wrist. We also conducted a test that would simulate an overdose by holding our breaths so that we could determine how accurate the chip was able to detect the drop in spO2 levels. By doing this, we were able to prove the precision of the MAX30100 chip.

(a)

Time(s)	Max30100 (pO2)	Standard Oximeter (pO2)
0	95	97
5	97	97
10	97	98
15	97	97
20	98	98
25	99	97
30	99	98
35	96	98
40	97	98
45	97	97
50	96	97
55	97	96
60	98	96
65	95	95
70	95	95
75	96	95
80	97	97
85	98	97
90	95	96
95	94	94
100	93	94
105	93	92
110	93	91
115	91	89
120	89	87
125	84	86
130	85	85
135	85	83
140	84	83
145	83	82
150	84	81

(b)

Time(s)	Max30100 (pO2)	Standard Oximeter (pO2)
0	98	98
5	97	98
10	98	97
15	96	97
20	95	96
25	98	96
30	97	98
35	97	98
40	96	98
45	96	97
50	95	98
55	95	98
60	95	98
65	97	94
70	97	95
75	96	95
80	94	97
85	94	97
90	92	96
95	91	94
100	94	94
105	94	92
110	93	91
115	92	89
120	85	87
125	88	86
130	84	84
135	85	85
140	82	85
145	81	82
150	81	80

Table 5. Pulse oximetry comparison data from (a) right index finger and (b) right wrist.

Unfortunately, due to the COVID-19 outbreak we had not been able to complete any further testing since we were unable to completely combine our components, the device frame and circuit, together. If we had not encountered this issue, we would have tested for the durability of the device. The test would consist of the complete device being worn on the wrist while enduring different conditions to determine what it can withstand. For instance, testing its resistance to water/sweat, comfort, physical activity, and force. This is important since the device

is intended to be worn on a daily basis. This test would help to determine if any of the materials would need to be replaced and how often.

We assume that the device would need little to no maintenance. However, we had not been able to completely test the circuit with the battery which could possibly bring problems with overheating. Also, with respect to the current final device made, the user must be somewhat cautious in order to avoid getting water and dirt/dust within the device frame since it could cause damage to the components. As mentioned before, working to make the device resistant is something we would have liked to improve on for the future.

There have been a few concerns that need to be considered for improvement and the safety of the user. While testing, we noticed that the oximeter chip would stop working sometimes. Specifically, this means that the circuit would be connected to power but the chip will not recognize this which is evident as the LED would not light up. As a result, we would have to press the reset button on the Arduino Nano. This is obviously worrisome since we do not want the chip to malfunction and leave the user stranded in a life or death situation. However, this unreliability could be expected since we were limited to an $100 budget, thus resulting in us purchasing electrical components that were cheap and of low quality. In the future, it might be best to invest in better materials such that we can ensure the best results.

In addition, another constraint that we would aim to fix for the future is making the phone application available for all phones, not just androids. This is because the MIT app inventor is only compatible with android phones which limits our user audience greatly. This could possibly be solved by using a different software when making the app, however, would cause us to start our work over again.

Furthermore, a risk that needs to be considered is that the safety of the user is dependent on their phone. The basis of our device relies on the ability of the phone application to detect when spO2 levels are less than 90% and thus triggering the failsafe alarm and message to emergency contact. In the case that the phone runs out of battery, the lifesaving purpose of the device is lost, thus leaving the user vulnerable and the device useless. This is something that would have to be greatly considered for improvement by the designer, as well as considered when used by the user.

5. Conclusions and Recommendations for Future Work

This manual describes step by step how each subsystem within our device, from the physical to the software aspects, were created. Overall, our design encompasses a discreet and modern solution by having a lifesaving device be disguised as a common watch. The device increases the chances of a user's survival by detecting when their spO2 levels are less than 90%, a characteristic that typically indicates opioid overdose. It provides reassurance to the user by its failsafe notification which when left with no response triggers 2 methods of alerting help, through an emergency text and alarm.

Each component of the device involved their own set of hardships and successes which lead us to the final device we have designed. This project required us to learn and/or develop many new skills such as soldering, 3D printing, and coding. Through this, we had discovered the importance of working ahead of schedule and as a team. Without strong communication and a common team goal, we would not have been as successful with our device.

Unfortunately, due to the COVID-19 pandemic we were not able to complete the finishing touches for the electrical and mechanical aspects of the device. Although the components of the electrical aspect were functional, we were not able to get it onto a PCB board to make it more compact and fit easily into the device frame, which was created for the measurements of the PCB board. For the mechanical part, we were not able to obtain the final piece, the watch strap, thus preventing us from testing the comfort of the device on the wrist. Also, as a result of us not being able to compact the components into the device frame, we were unable to implement a sealing mechanism as well as see how functional the device would be inside the device frame.

As the project was in development, the battery size vs device size quandary manifested itself throughout. This was mainly because the client emphasized the importance for a discreet and comfortable device, but at the same time decreasing the size of the device via decreasing the size of the battery caused electrical complications. These complications were difficult to confront due to lack of time and inexperience.

Moving forward with this project, the watch strap and PCB must be finalized and put together to finalize the plans that were already put in place for this project. Next, changes could be made to improve the device that we have already made. Changes would include a slightly smaller device frame, this is to ensure that the PCB board is very snug in the device and doesn't shake around at all. Also, we would have done more research to find an effective sealing mechanism for the device.

As much as it is a great success that majority of the goals we set regarding the phone application were completed, there are a few changes we would have liked to make that would improve the device for the future. For user experience, it would be beneficial to allow the users to be able to customize their warning values, so they can pre-set their failsafe time (currently set at 30 seconds) and their warning blood oxygen level (currently set at 90%). This includes creating a screen on the device that displays the recorded heart rate and blood oxygen saturation levels such that the user can review them whenever desired. Especially if we could present this in an efficient manner like a graph.

Despite all of the improvements that could be made, the device we have created closely encompasses the vision we have for our final device. As a team, we were able to persevere past all of the obstacles faced to create a device that we are all proud of. We hope that our device will help others to develop a reliable and secure solution to this critical issue.

6. Bibliography

Muhammad H. Rashid, *"Power Electronics, Circuits, Devices, and Applications"*, Third Edition, Pearson Education, Inc., 2004.

APPENDICES

APPENDIX I: Design Files

- MakerRepo Site:

https://makerepo.com/antoniazupu/c12-lifeline-

In this website, all of the project deliverables as well as files, such as the STL files, can be found and used to help in continuing this project.

- Video Representation of the Lifeline application:

https://www.youtube.com/watch?v=VFOqZ02ltkY

This video shows the Lifeline application on the smartphone. All of the features and functions that the application includes are shown here as well as how to use the app.

- https://drive.google.com/drive/folders/1wM0fo2MGWEJffWhuUTrqkFKHpNB9UJ85?usp=sharing

  This drive contains an AIA file for users to download onto MIT App inventor and an APK file for users to download straight to their phones using the MIT App inventor Companion app.

APPENDIX II: Other Appendices

Chosen Watch Strap Option	https://www.kijiji.ca/v-jewelry-watch/ottawa/black-silicone-rubber-watch-strap-with-tool-for-g-shock-watch/1455423931

Arduino Libraries Used	**Link**
MAX30100	https://github.com/oxullo/Arduino-MAX30100
Software Serial	https://www.arduino.cc/en/Reference/SoftwareSerial
Wire	https://www.arduino.cc/en/Reference/Wire

Alarm Sound MP4 Link:
http://soundbible.com/81-Red-Alert.html