SEIJI JOHN MADRIGAL

**SMARTWATCH**

**INTRODUCTION**

The main objective of this project is to build a ‘smart’ wrist watch similar to commercial smartwatches in the market like the apple watch and Samsung gear watch and to be able to fit Arduino parts with different functions to fit in a small form factor casing. This project initially started out as a google glass project, to create smart glasses which mimic the function and build of Google’s 2015 glasses. However, due to the limits of parts size, a higher level of difficulty of 3D design building and limitations in time, I opted to create a smartwatch instead. Additionally, the of this Arduino smartwatch needs to communicate with a smartphone to be able to receive data such as time, currently playing song, and notifications, as well send back data for commands like changing the volume, skipping tracks and play/pause. To achieve this, an application on the smartphone also needs to be created in order to automatically communicate with the smartwatch and persistently process send and receive data. Spotify integration in the app for music control is also included within the application. With this the application must connect with the Spotify Remote API following the manuals shown on the Spotify Developers website. The app also should support and implement the use of Bluetooth Low Energy features to work efficiently with the parts chosen for this build.

**TOOLS AND EQUIPMENT**

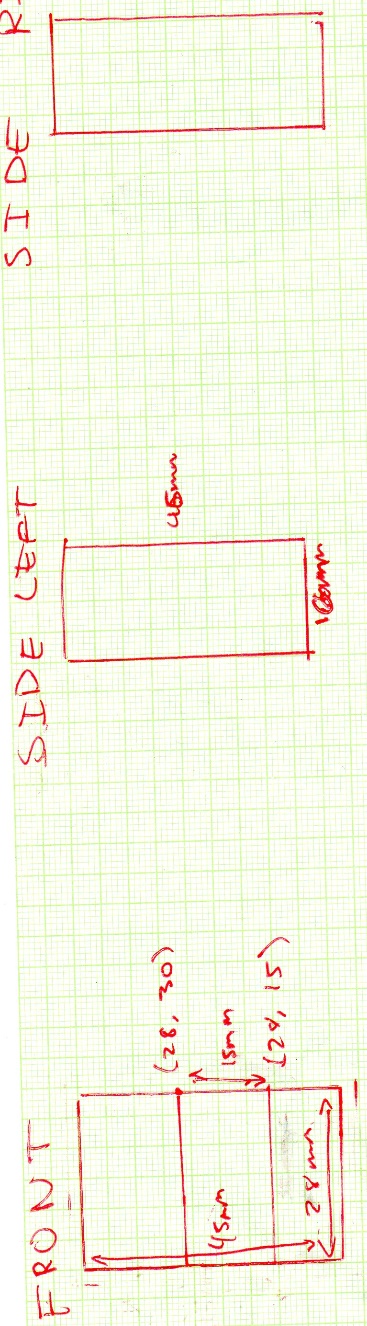
Soldering Iron & Solder – to connect all the wire and parts together.

Wire Stripper – remove external wiring to expose copper for soldering.

Computer – program the Arduino and Android app.

**SAFETY & HAZARDS**

The main concern for safety is when using the soldering iron. The soldering iron is a dangerous tool that can lead to significant burns and injury, the solder can also be harmful if fumes are inhaled or make contact with your eyes, is required to wear safety glasses and maintain distance from the solder and smoke. Sharp tools should also be used in caution, like wire strippers, using the right method of stripping, pushing away from your body to reduce the chance of getting a cut.

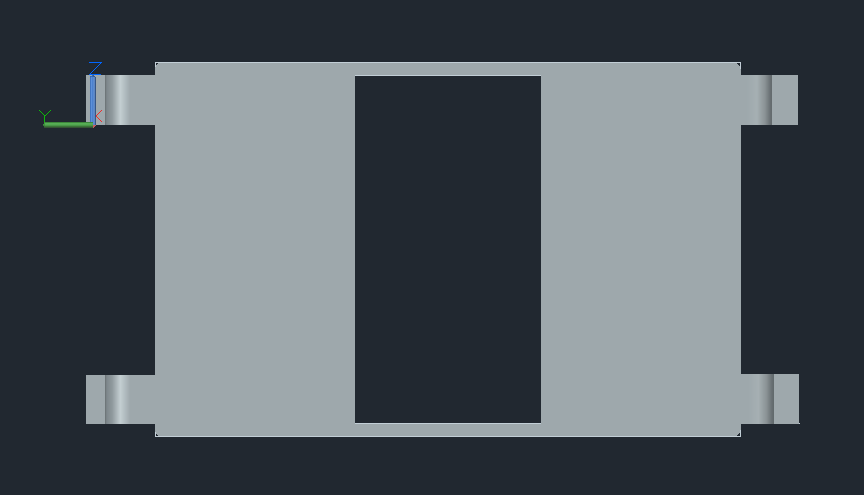
**PARTS**

**CASING**

FIGURE 1: PROTOTYPE SKETCH

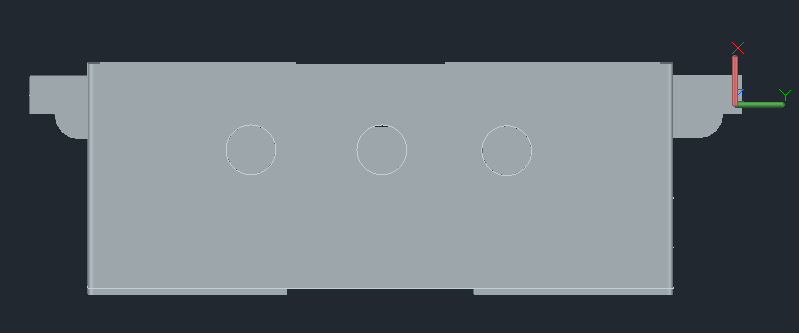
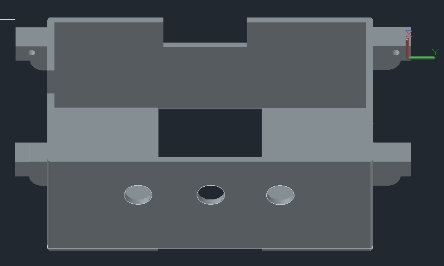
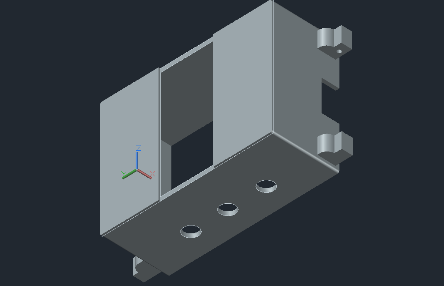
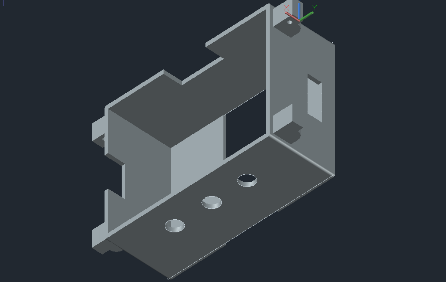
The main in designing the casing for the smartwatch is the size, to be able to fit the user’s wrist and not be too bulky to wear for regular use while also large enough to fit all the stacked components inside the case. The case has a length 45mm 28mm in width with a density of 16mm. The front of the case should have a 15mm gap in the middle so the 0.96” display can be fitted inside the case.

Figure 2: 3D DESIGN SKETCH



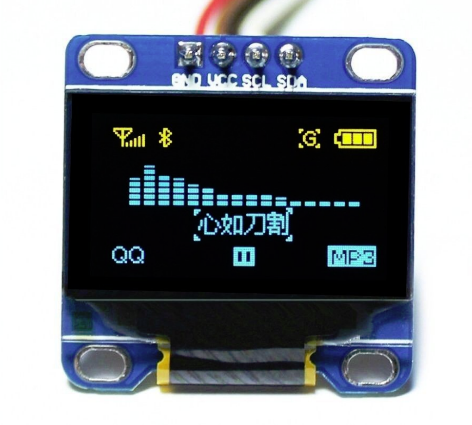
SIDE RIGHT

FRONT

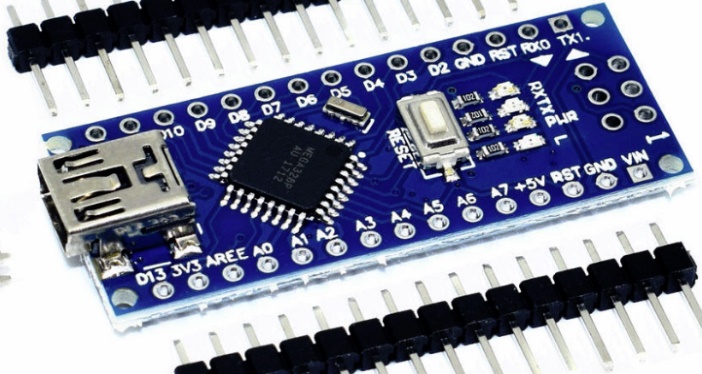


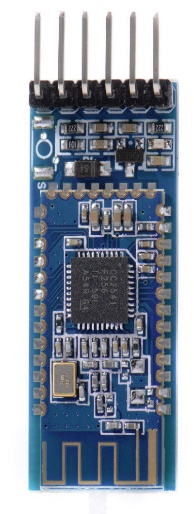
This 3D model of the casing is design in AutoCAD 2018. This is where button holes are included in the design on the left side of the case to be used for the buttons to navigate the screen. Rectangular cut-outs were also added on the right, top and bottom side for switch, micro-USB and mini-USB connector respectively. 4 chamfered watch strap holders with a chiselled lug hole with a 19 mm distance between them to allow standard 18mm watch strap to be connected to the watch.

**DISPLAY**

This is a 128x64 0.96” OLED Yellow and Blue display. To minimise power consumption, an OLED display should be used for wearable devices such as a smartwatch as only pixels that are on are illuminated, therefore power output is minimised to areas on the screen that is displayed. This can extend the battery life of the smartwatch allowing it to last longer during the day making this device practical to use. This display connects directly via the I2C bus of the Arduino, pin A4 & A5 for the Arduino Nano (varies on Arduino models). A prebuilt graphics library can also be used to program the display, the library chosen for this build is the U8glib library for the SSD1306.

**ARDUINO**



The centrepiece of the smartwatch is the Arduino that processes the code created on the Arduino compiler and communicates with the Bluetooth module and the display. The microprocessor embedded on the Nano is an Atmel ATmega328 CPU with a clock speed of 16MhZ. It has 32KB of flash storage and consumes 19mA of power. The Nano has 12 digital pins and 7 digital pins. This is more than necessary as only 2 digitals pins and 5 analog pins will be used for this project.

**BLUETOOTH MODULE**

The Bluetooth module I used is the HM-09 Bluetooth 4.0 Serial Module. This module allows serial communication between Arduino and a Bluetooth LE capable device (android phone). A Bluetooth Low Energy module is slower than a regular Bluetooth module, however it consumes less power during stand-by time using around 90-400 micro amps of current. Additionally, Bluetooth LE in Android 4.3 and above allows easy use of Bluetooth APIs to easily integrate low energy scanning, discovering services and manipulating individual characteristics. Connecting the Bluetooth module to the Arduino, the transmit pin of the module can only take a voltage of 3.3V, a voltage divider to convert 5V from the Arduino to 3.3V is needed.

**BATTERY & CHARGING MODULE**

Battery used for this smartwatch project is a 3.7V 100mAh lithium-polymer battery. Although there are other batteries with higher capacities ranging from 350-500mAh with similar form factors, however other batteries are from China and would’ve taken a few months to ship to Australia. For the charging module, the TP4056 Charging board that uses micro-USB input.

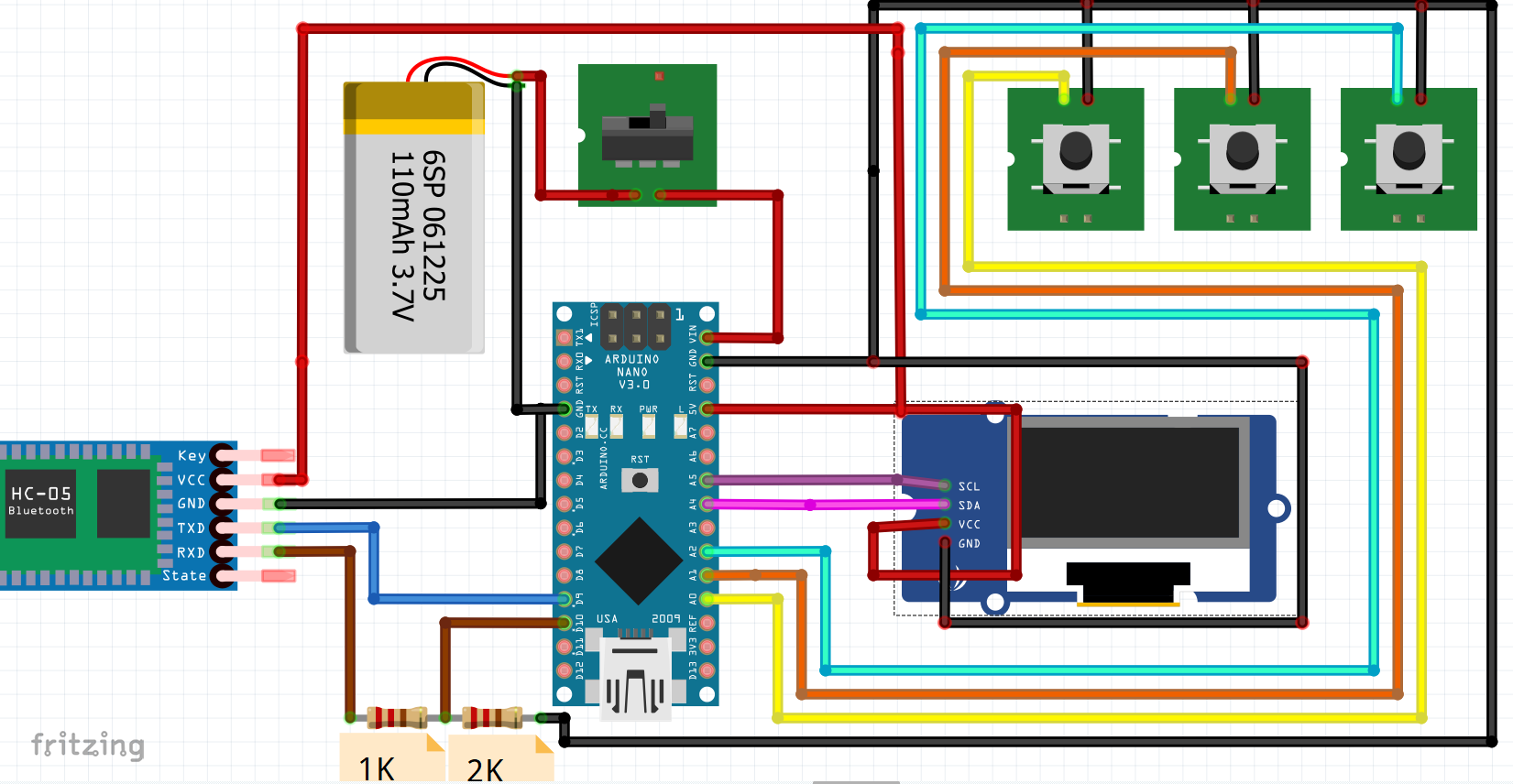
**BUTTONS & SWITCH**

Push buttons are used to interact with what is on displayed on the screen, 3 buttons will be needed which will be up, middle and down button respectively. There will be 2 different states of button press, single press and long press. The up and down button will be used for controlling the volume on single press and previous and next on long press state. The middle button will be used to navigate between the different sub-menus on the screen from the Home screen to the Music page on single press and Play/Pause music for long press. The switch will be connected in series with the battery and the Arduino to switch the Arduino On/Off.

**PLANNING**

|  |  |  |
| --- | --- | --- |
| PROJECT DEVELOPMENT SCHEDULE | | |
| WEEK | CATEGORY | SPECIFIC DETAILS |
| 1 | ANDROID | Create a basic UI and code a program log for app debugging. |
| 2 | ANDROID | Use the Bluetooth LE API on the app and testing. |
| 3 | ANDROID | Add a service class to the app that allows the program to continuously run in the background. |
| 4 | ANDROID | Use Spotify Remote API and include it within the app. Test functions such as play/pause, previous/next, etc. |
| 5 | ARDUINO: SOFTWARE | Include all the libraries needed for different module. U8glib (screen), One Button (push buttons), Software Serial (Bluetooth Module). |
| 6-7 | ARDUINO: CASE 3D DESIGN | Develop the casing for the smartwatch on AutoCAD and 3D print the prototype design. |
| 8 | ARDUINO: WIRING & | Wiring different parts together before soldering all the parts to test functionality. |
| 9-10 | ARDUINO: PROGRAMMING | Program the Arduino board to display & move text on the display screen, read and write serial Bluetooth messages and add functionality to the buttons. |
| 11 | ARDUINO: SOLDERING & ASSEMBLY | Soldering all the wires together and fitting all the parts into the case. |
| 12 | FINAL TESTING | Testing for any malfunctions and bugs. |

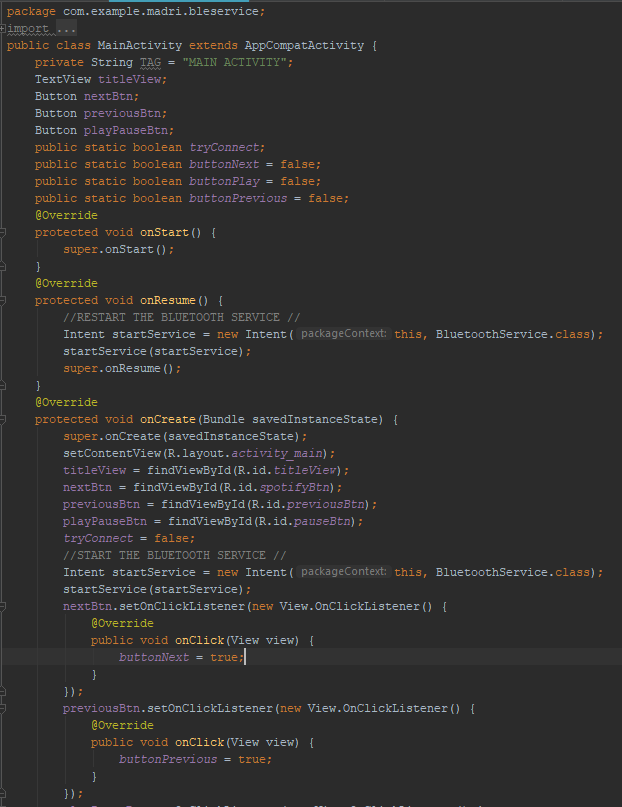
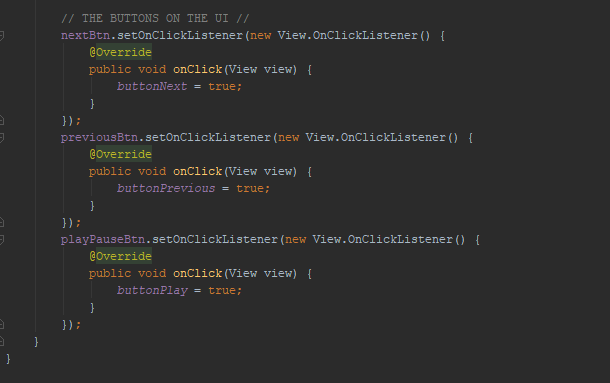
**SCHEMATICS**

****

*This is NOT a complete wiring diagram of the build. The program, Fritzing, does not include a charging module object, the charging module is what connects the battery for charging. The charging module just connects to the battery in parallel.*

**SOFTWARE DEVELOPMENT**

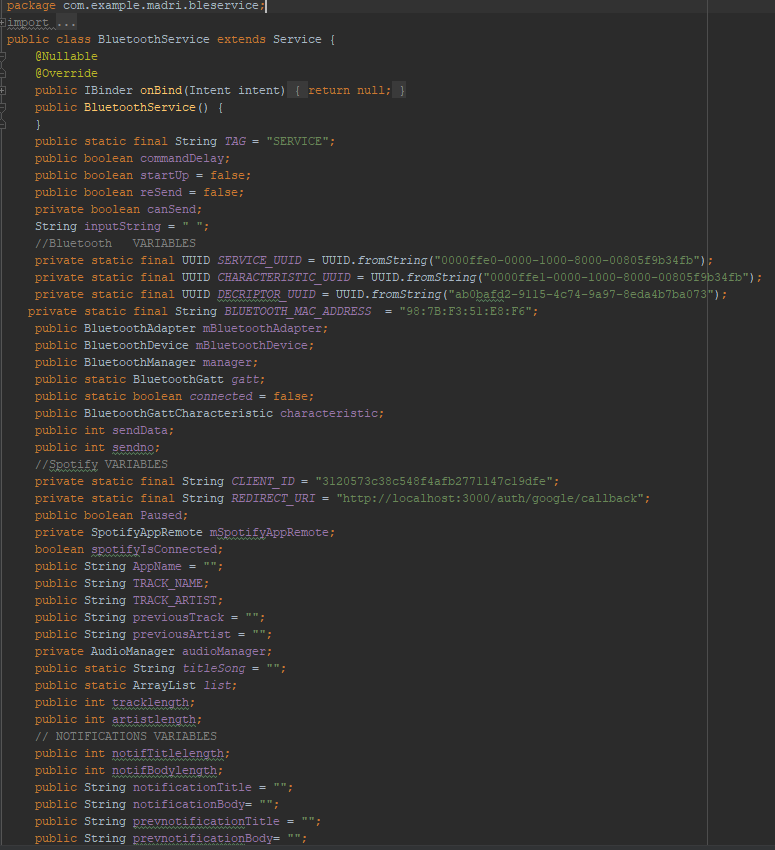
**ANDROID APP: BLE SERVICE**

Main Activity – this is the main UI activity of the BLESERVICE app. The main purpose of the activity is to start the BluetoothService.java service using an Intent; an operation to start a new activity on the App. The UI includes three buttons used for debugging previous/next & play/pause functions of Spotify.

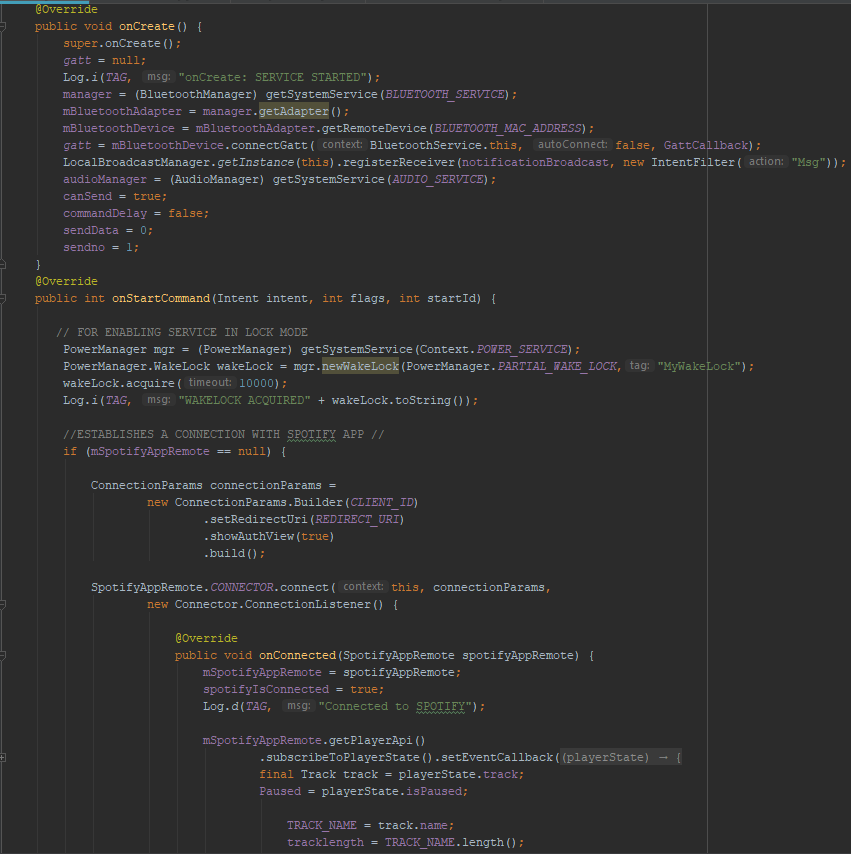
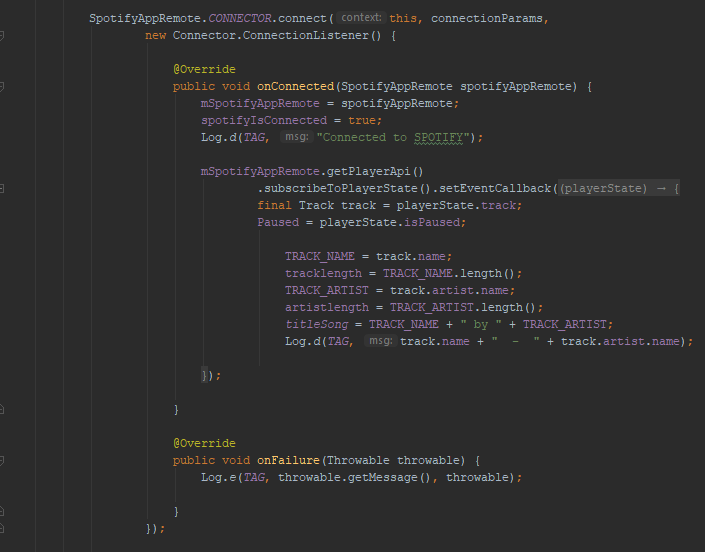
Bluetooth Service – this is the ‘brain’ of the application where all the information is collected and sent from phone to the Arduino Nano.

The initial part of the code is declaring all the variables that will be used. These are separated in different sections, variables for Bluetooth, Spotify and the Notifications.

The Bluetooth variables consist of the MAC Address of the Bluetooth Module and the Universal Unique Identifier (UUID) for the specific service, characteristic and descriptor of the Bluetooth module. The custom characteristic enables the program read and write values.

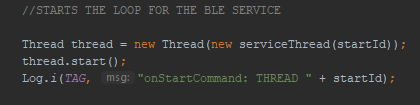
The Spotify variables includes a required *CLIENT\_ID* that is acquired through registering the app through the developer’s website. A default Call back *REDIRECT\_URI* is also required that allows the user to go back to the app one successfully signed in through Spotify. Other variables are used to store the current Track & Artist currently playing on Spotify. Lastly, the notifications variables are used to store the notification header and the notification message.

STARTING THE BLUETOOTH SERVICE

The next section of the code is the *onCreate()* and *onStart()* functions. These two functions are called when the service first initialised by the intent from the Main Activity. In *onCreate()*, objects that declared such as the Bluetooth Adapter, which is the Bluetooth transmitter from the Android phone and the Bluetooth Device which is the Bluetooth Module. The onStart() command includes an attempt to acquire a wake lock which is a feature that allows the service to remain running in the background even when the device is locked. The next section is for checking whether the Spotify Remote is connected, if not it builds a new sign in screen on top of the app to validate the user. The next section of code basically gathers information of the Track and Artist name when the Spotify app is connected and stores them to the variables that were declared previously.

INITIALISE THREAD

The last commands on the onStart() function starts a new thread, a different runnable program that is independent from the main UI thread, responsible for detecting button presses and changing screen values , this separate thread can be run continuously and does not impact operations on the UI thread.

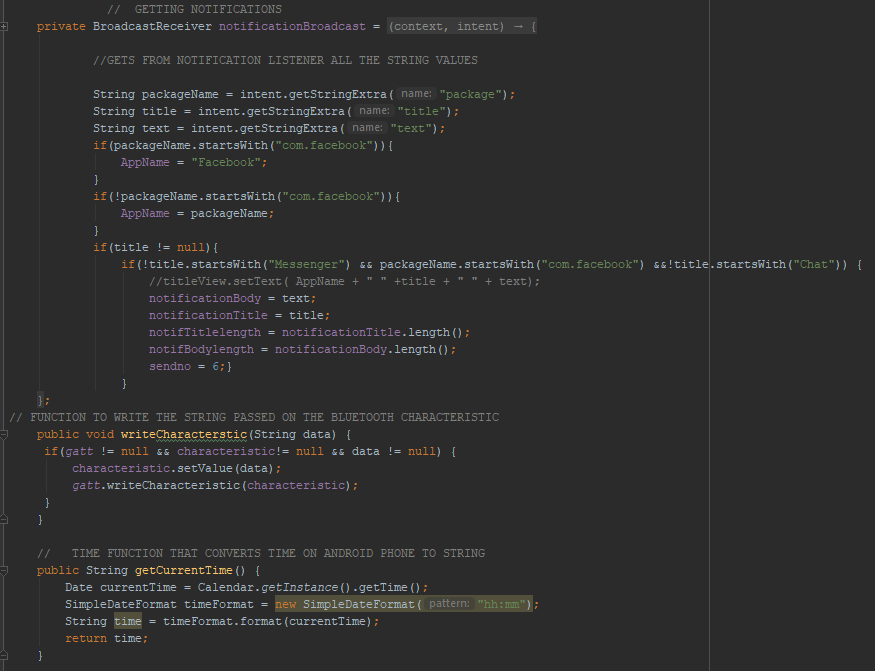


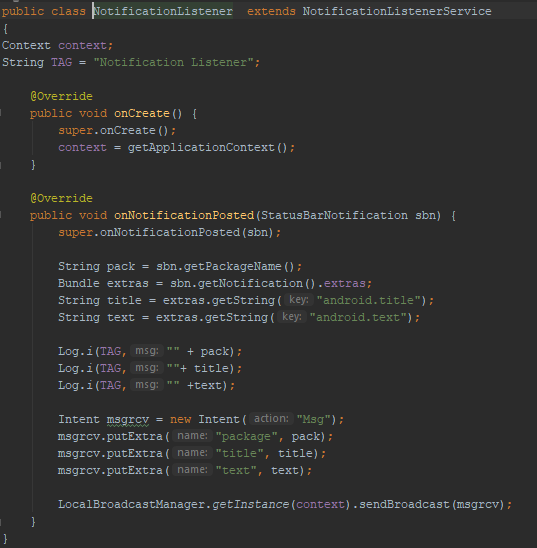
BLUETOOTH CALLBACK

This call back contains many functions that identifies the state of the Bluetooth connection, if the connection has changed the *onConnectionStateChangeI()* will be called and if services are discovered *onServicesDiscovered()* will be called and so on.

BROADCAST RECEIVER & NOTIFICATION LISTENER

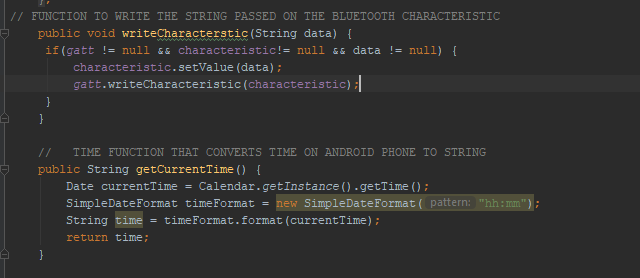
This is a receiver that waits to receive messages from the NotificationListener.java. The Broadcast Receiver uses obtains this information and checks what type of notification it is. In this code, the Broadcast receiver checks whether the notifications are coming from the Messenger app.

The Notification Listener is a separate class that checks for phone notifications that are on posted on the status bar. This is then sent to the Broadcast Receiver through an Intent.



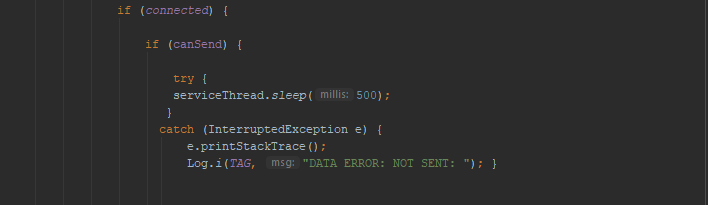
WRITE & TIME FUNCTIONS

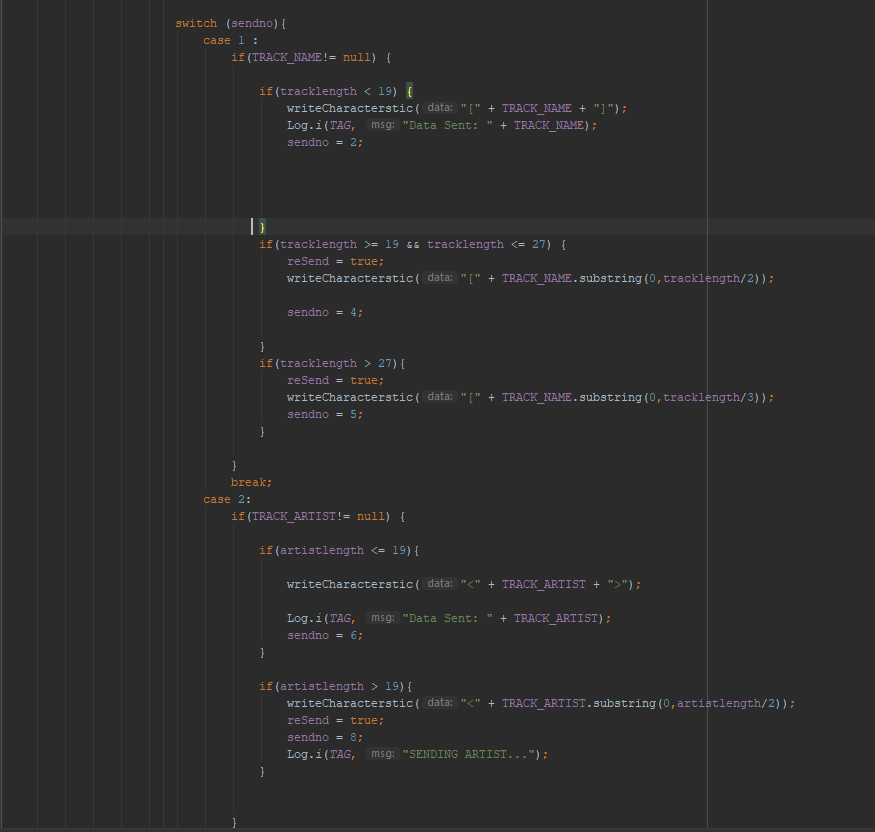
writeCharacteristic() – this function can be called anywhere in the program to pass String values and write them on the custom Bluetooth characteristic.

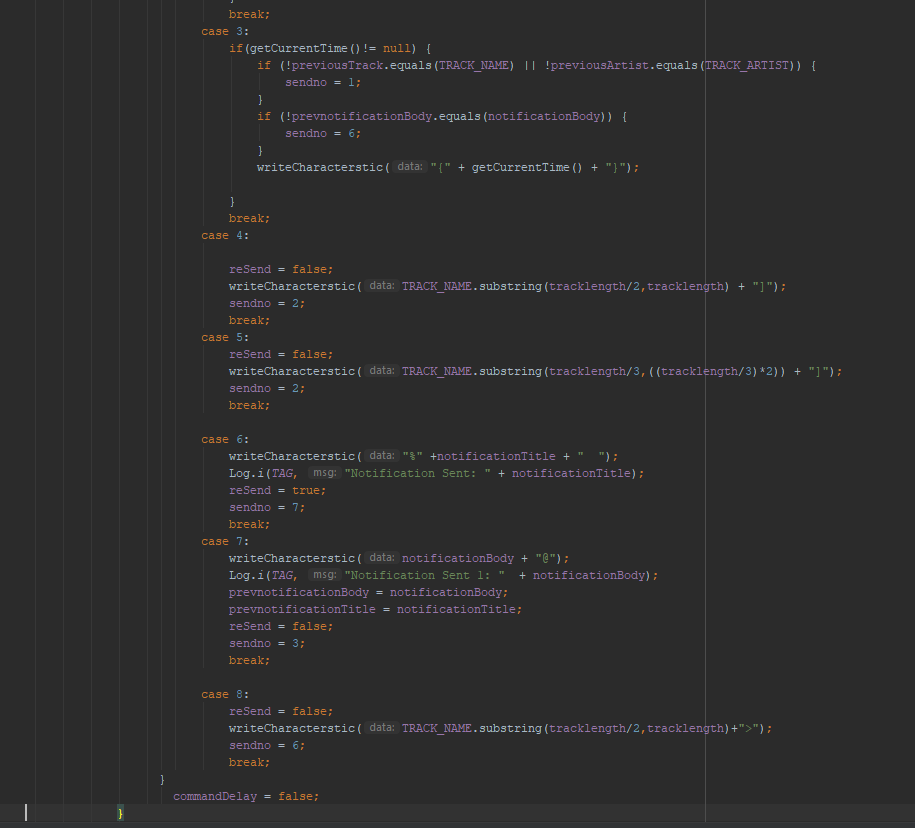
getCurrentTime() – Gets the current time on the Android Phone and converts it to String in a format Hour : Minutes.

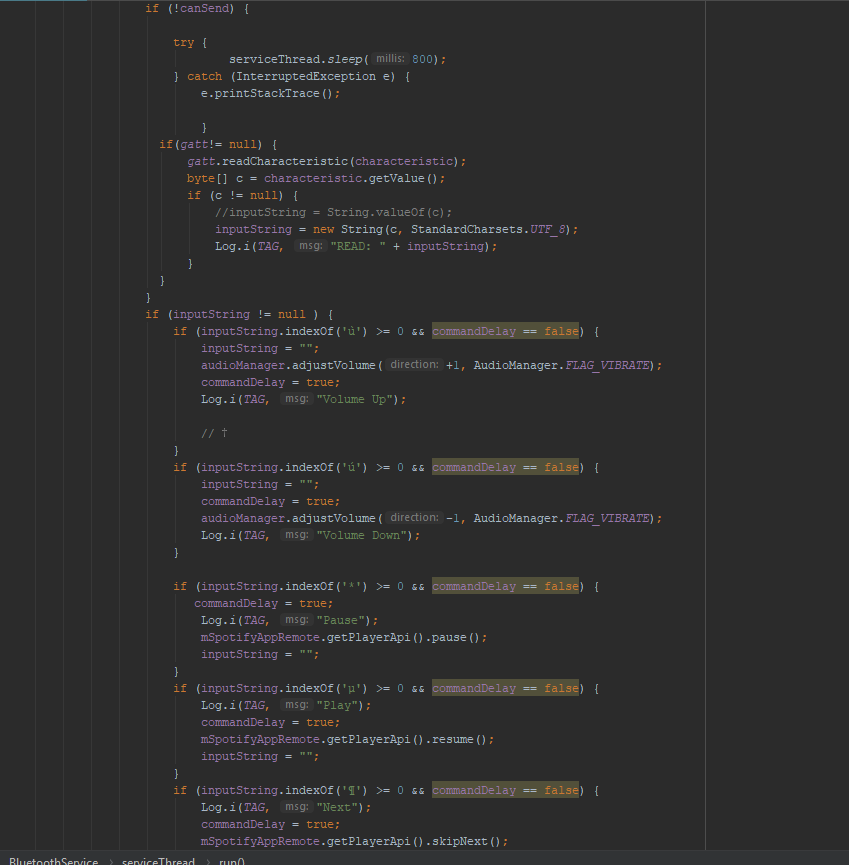
BLUETOOTH THREAD & RUNNABLE

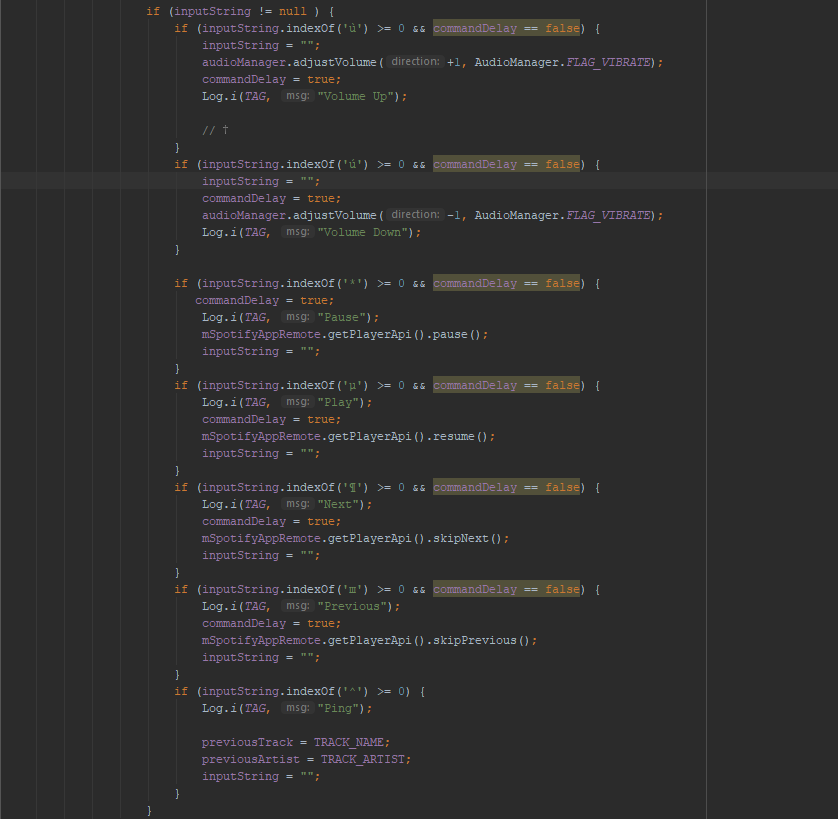
This class *serviceThread* is the thread that was initialised in the *onStart()* function. This Thread has a runnable process that contains a while loop, which runs commands within it indefinitely until the service is stopped.

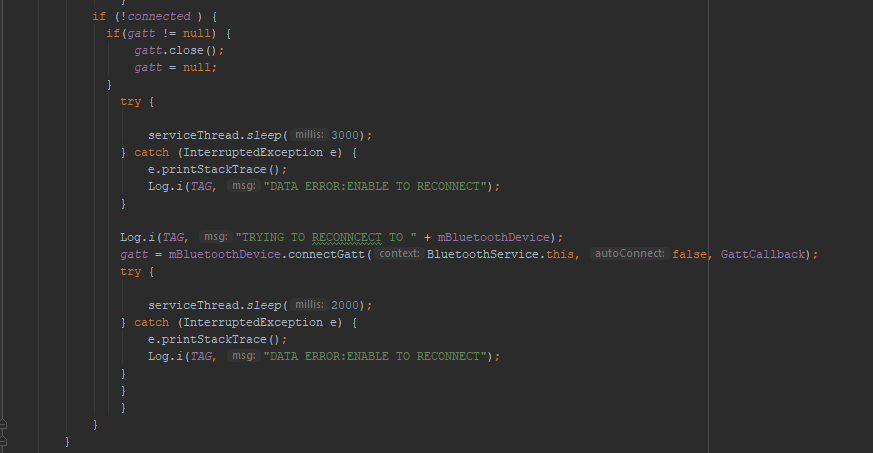
The next block of code within the serviceThread checks whether the Bluetooth module is connected to the phone, when it is connected it then checks if the Boolean variable *canSend* is true. The *serviceThread.sleep()* is basically a delay before the rest of the code is executed, this delay is to avoid data being send on top of each other.

After the delay, a switch statement with different types of data to be sent. The main purpose of this is to send a small chuck of data one at a time due to the limits of the data throughput of Bluetooth serial. The switch statement only executes a case depending on *sendno* integer. This will allow the serviceThread.sleep() to be executed before the next case runs.



This is the block of code is responsible for reading the custom Bluetooth characteristic. This code runs when *canSend* is false which allows this code to run when the service is not writing to the characteristic. The code *uses characteristic.getValue()*, a byte array and converts it to a String with a UTF-8 encoding.

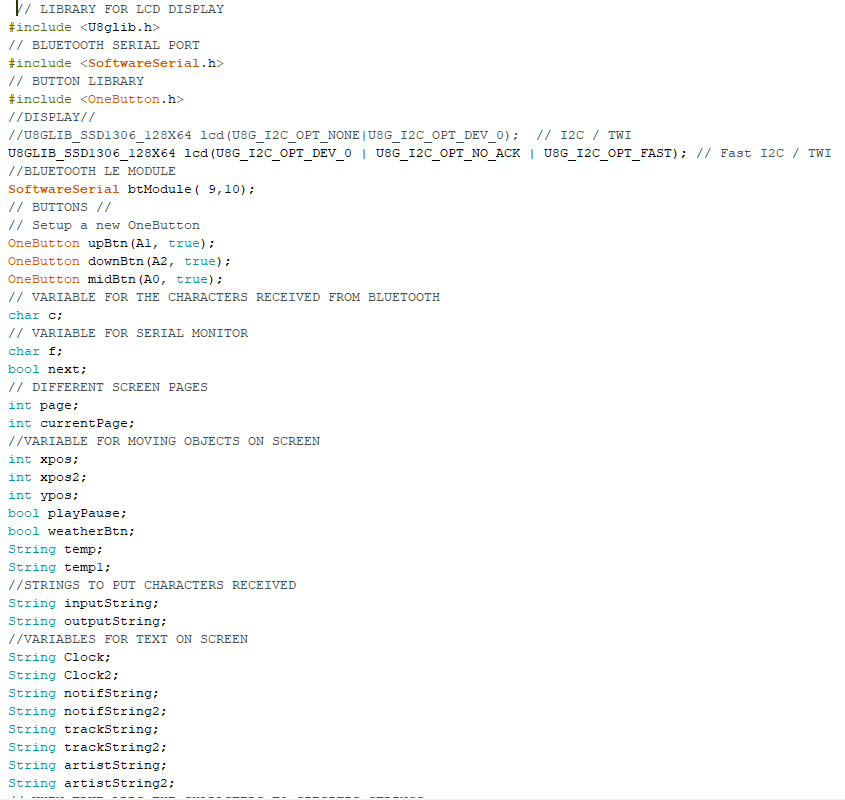
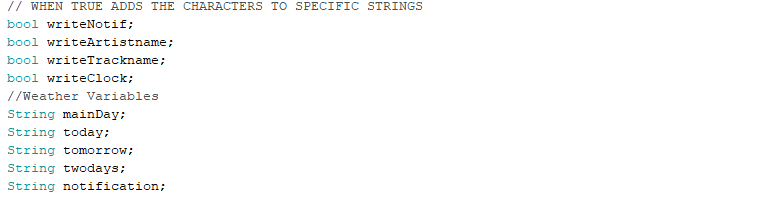
Also, within the while loop, this next block of code checks whether a specific symbol is read from the Bluetooth module. This translates into commands for different functions such as Volume, Skipping Track and Play/Pause.

This Code Block checks whether if the connection is broken. If yes, it will try to reconnect to the Bluetooth module every 5 seconds.

This next code block is when the service is forced to close. It will disconnect Spotify and the Bluetooth device connected to the Android Phone.

**ARDUINO SOFTWARE**

LIBRARY AND VARIABLES

The *#include* tells the compiler to include the library set for *U8glib* (display), *SoftwareSerial* (Bluetooth Serial Communication) and *OneButton* (push button detection).The rest of the code declares variables needed for different aspects of the Arduino program, from Bluetooth module, declaring that the Rx and Tx pins are in pin 9 and 10, declaring push buttons connected via A1, A2, A3, and other miscellaneous variables for screen text and animations.

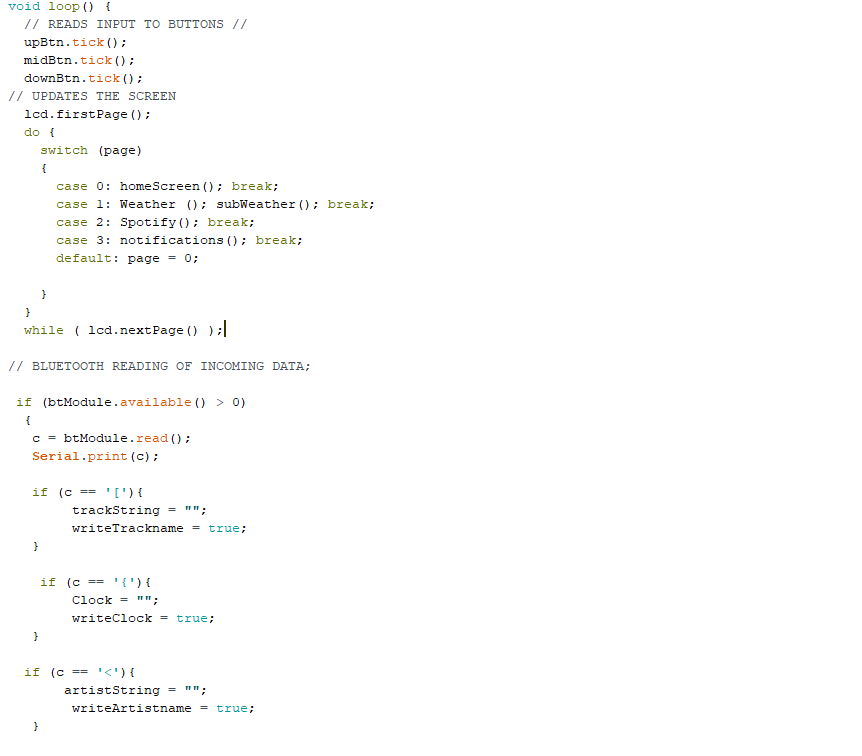
SETUP

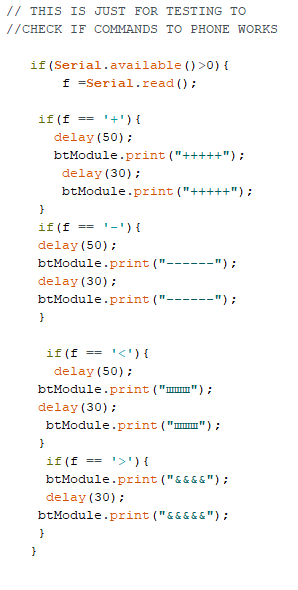
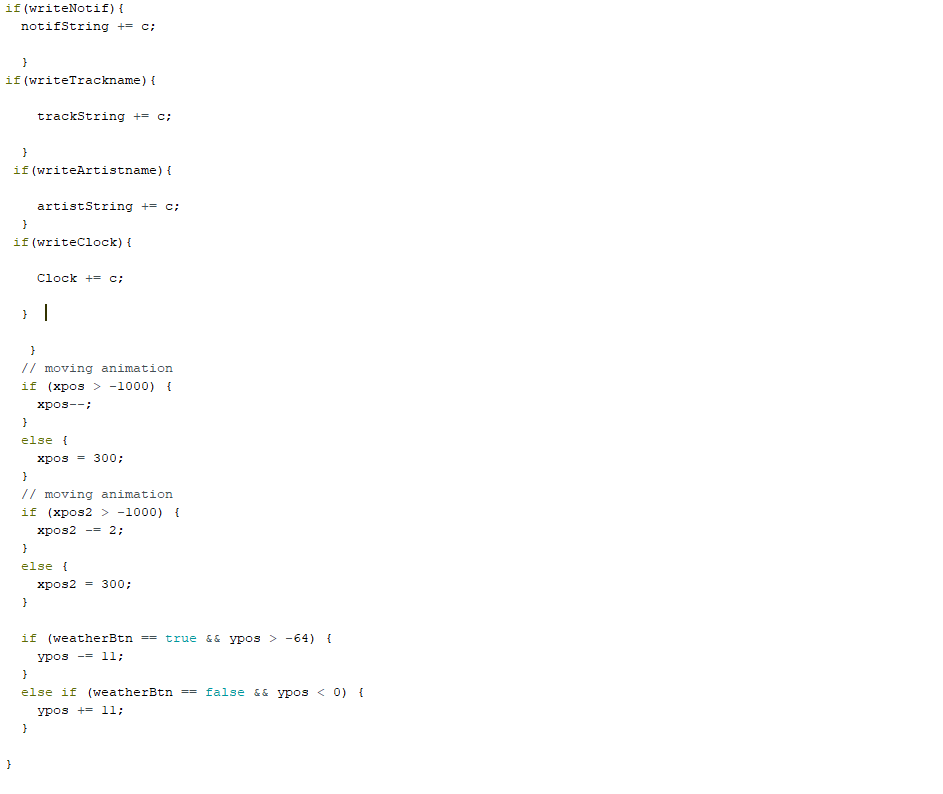
The setup() function on the Arduino is what will be executed when the Arduino Nano boots up. Code inside this only gets executed once code that is needed to be ran only once can be embedded here. This includes setting up the serial communication with the Arduino and Computer through *Serial.begin()* (this is used for testing) and serial communication between Arduino and Bluetooth Module *btModule.begin().* Buttons are also initialised here to have one click and long press features.

The *bootScreen()* function prints out boot text on the Screen.

LOOPER

The loop() is where all the continuous operations are executed.

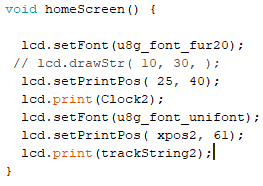
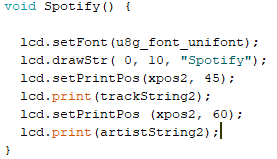
The first 3 lines monitor whether any buttons are pressed. The switch statement updates the screen with the right data depending on the *page* integer variable. The next half of the code detects if there is incoming data from the Bluetooth serial connection. The data is then stored as a character in the variable *c*. There are different starting characters for different types of incoming data. This allows the program to know which String to save the data to, for example, the Artist name will be sent from the Android app to the module with a starting character ‘<’ and an ending of ‘>’, this will then allow the Arduino to know to save it in the *artistString* variable to be stored. Once data received has been saved, the Arduino will send a ping back to the Android that it has finished saving the values, and it will stop sending the same values.

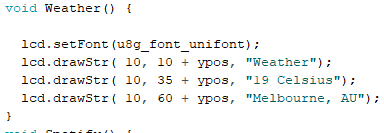


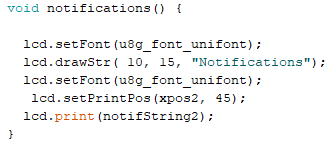
The *xpos* and *ypos* are the integer value of the x-axis and y-axis points on the screen. The x-axis animation allows long texts to be displayed on the screen through moving from left to right whereas the y-axis animation is the animation of opening a sub-page for the Weather page, to see more information.

PAGES

These are separate functions that make up the different pages of the screen. Every page will have different type of information on the screen. The Homescreen will display the time and the current playing track. The Spotify will have the current artist and track name displayed, the Weather & Sub Weather screen will have the temperature and location and the Notifications screen will display messages from Facebook Messenger.



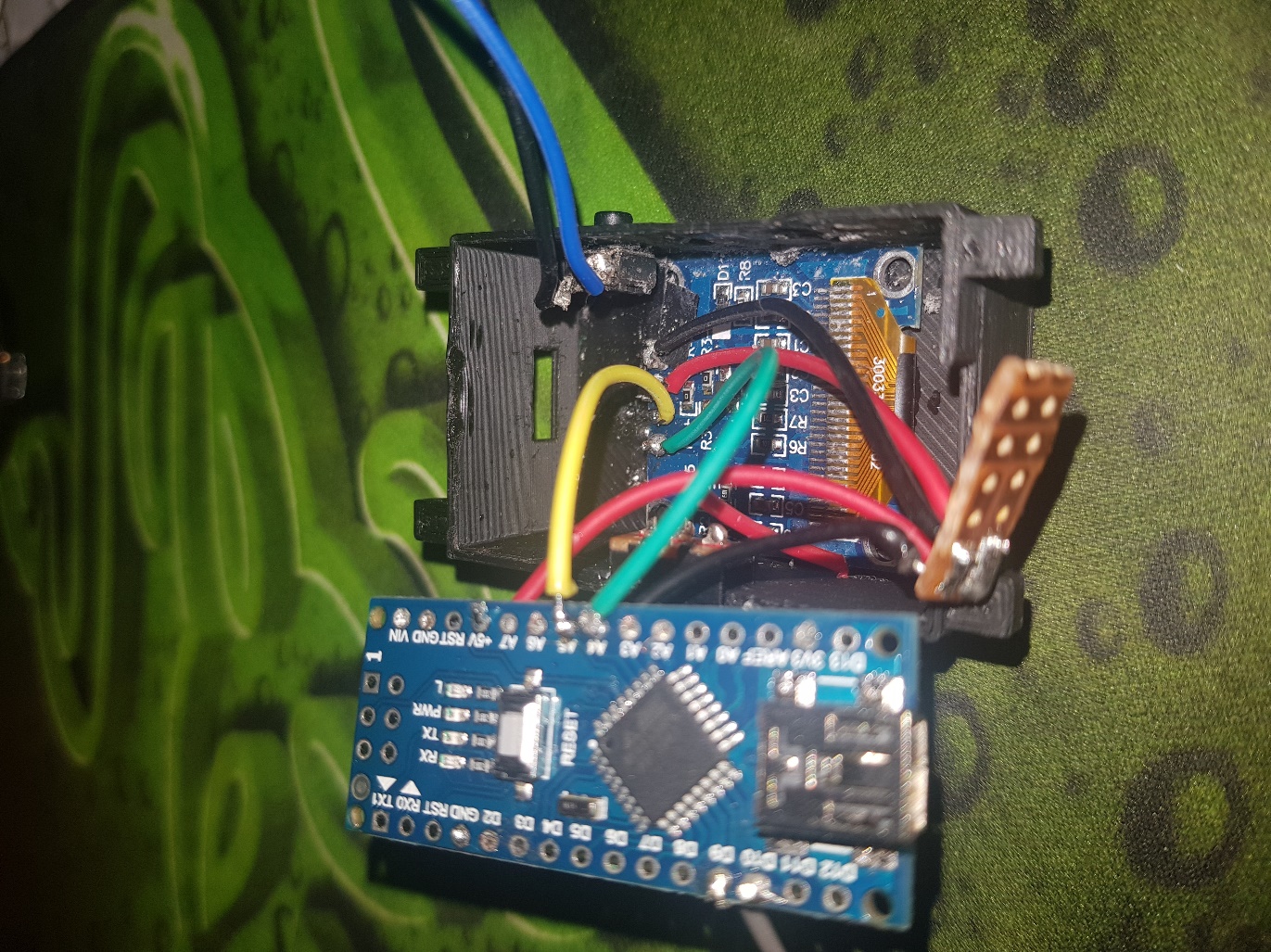




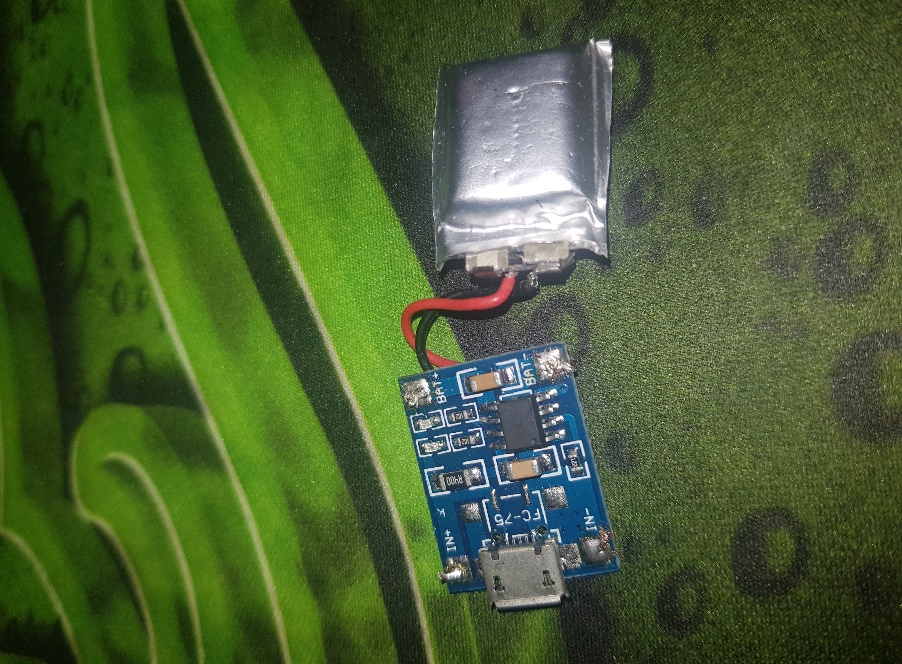
BUTTONS

The rest of these functions are for single press and long press commands for individual buttons. The middle button single press will change the page. On the Spotify page (page 2). The functions of the Up and Down buttons are enabled. Commands inside the up and down buttons and long pressing the middle button can allow the user to change the volume, skip tracks, and Play/Pause.

**ASSEMBLY**

  
1. The first step is to glue the display to the inside of the case using the top and bottom side of the screen. The buttons and the switch should also be glued from the inside of the case. Then the next step is to solder the Arduino Nano and the display together. Follow the wiring diagram, strip each side of a 5 cm wire and solder each end to the Arduino and Display. The Ground and 3.3V supply are soldered on a mini PCB breadboard to organise and remove excess wiring for better fit inside the case.

2. The next step is to solder the battery to the charging module. Connect in parallel to the Arduino VIN pin and Ground. Then when connected place the battery and the charging module on top of the display board, ensure that the micro-USB connector is sitting flush inside the rectangular cut-out, use super glue for securing the connector in place.



3. Using the wiring diagram, strip wire 4 cm in length and solder each side to the Bluetooth Module and Arduino. Place the Bluetooth Module on top of the battery and charging module routing each all the wires to the side. Create a voltage divider with a 2K and 1K resistor to connect to the receive pin of the Bluetooth module.



4. Tuck all the wires and parts inside the casing safely avoid disconnecting soldered wiring. Slowly ease the Arduino Nano into the case. Attach a cover to the backside of the case. In this case I have attached a smooth rubber sheet cut from a mousepad and glued the edges on the case. This makes wearing the watch more comfortable. Then attach any 18mm watch strap.



FINAL PRODUCT



CONCLUSION

Although the project was a success, the both the app and the Arduino software were running properly, there were still many bugs that needed to be fixed for the watch to be working as I envisioned it to work. More time could have been used to develop this watch if I had decided to do this at an earlier stage instead of trying to make the first project, Google glass work. Better parts could have been chosen to make the watch last longer with a higher capacity battery and have a slimmer form factor with using an all-in-one Bluetooth and Arduino board such as the [Feather M0](https://www.littlebird.com.au/adafruit-feather-m0-bluefruit-le). Furthermore, many other features could have been added with this this board with a larger storage such as storing and displaying bitmap images on the screen and making having more content written on the board. The main method that could have been improved in this project is time management, using the time in class to work on the project more efficiently and asking the teacher for help and assistance for aspects of the project that I was stuck on.