Project Report for Smart Wearable

Contents

Introduction

Brief Project Description

- A demonstration-based project built to display a smart wearable device's basic framework and functioning. This device tracks steps, shows direction (compass), shows current body temperature and Average Body Temperature. This information is being sent to an android app through a cloud. This device is mainly used to accompany you during your cardiovascular (running/walking) exercise.

Problem Definition

- Exercise is a bodily activity that enhances and maintains physical fitness. The benefits of bodily exercise can range from preventing heart disease to keeping a healthy body mass to even preventing mental illnesses. Unfortunately, not everyone has the time and motivation to follow exercise as a routine or habit. According to a study posted by the World Health Organisation (2018) that used data from hundreds of surveys worldwide, more than 25% of adults lack exercise as a routine.

- Aerobic exercise (Cardio) is usually the most preferred type of exercise as it takes less effort and is simple in practice. The lack of aerobic exercise leads to problems like overweight/obesity, lack of stamina, heart problems, unhealthy moods.

Project Objectives

- This smart wearable device is built to track parameters related to **human movement**.
- As we have mentioned earlier, aerobic exercise, which involves movement like running, walking, jumping, etc., is one of the most essential and straightforward forms of exercise.

Primary Objectives:

- This project's first and foremost objective is to make exercise more manageable as a routine.
- This can be done by helping the person **track** their **exercises** and help them set goals for themselves.
- This device will act as a little companion.

Literature Review

This section will further include the overview, analysis, and summary of the research on the importance of Aerobic Exercise.

We have a research method of observatory and descriptive research.

As mentioned earlier there is a plethora of diseases directly or indirectly linked to the lack of cardiovascular (aerobic) exercise.

Heart disease is almost always linked to lack of aerobic exercise and is also a leading cause of death (Masoudi et al., 2017). In work (Masoudi et al., 2017), there are provisions of clinical data from cardiovascular disease patients. This data provides us a fair understanding of the reasons for the mentioned disease and the specific type of disease. Heart artery blockages, high blood pressure, cholesterol is very abundant in these data sets. Cardiovascular exercise can help prevent the conditions mentioned above.

In work (Eijsvogels et al., 2016), it is described in what ways cardio exercise helps. Put in briefly, aerobic exercise involves our heart beating faster than usual pumping more blood than usual. This increased blood flow can unblock arteries and veins (fatty deposits). Several studies also show that the body can create more connections between different blood vessels that can improve blood circulation even further.

In work (Vaes et al., 2022), A study has been done on the acute changes of the retinal blood vessel diameters which come as a reaction to higher-intensity exercises. This study shows how this helps the blood circulation in the body and help prevent cardiovascular diseases.

Lack of exercise can also lead to depression and feeling of anxiety. In work (Chaddha et al., 2016). There is a description of the link between hormone release and physical exercise. Additionally described in the work, exercise can help the person get distracted off problems in their personal life such as divorces or bad experiences, etc.

Lack of exercise can also easily lead to excessive fat build-up and storage.

The facts and observations from the literature we have reviewed helps us understand the importance of creating solutions to the mentioned problem and provides us great motivation to pursue and attempt in making our solution to this.

Project Planning

This section will include an overview of the project plan, Work Break-Down Structure (WBS) and a Gantt chart to visualize the prior mentioned information.

Our project development method was the **Waterfall method** but with **testing** at each stage that its required. We follow a pretty linear process of development

Again, this project aims to create a smart device that can calculate your bodily parameters and make the information available to you anywhere through an android application and connection to the database.

The parameters we plan to measure is **bodily movement** (i.e., Steps taken, Idle time, Sleep/Rest schedule), **body temperature** and an average **body temperature.**

An additional feature included is the **compass**.

A microcontroller-based board will be used which has onboard **sensors** which feed raw **data** to a **data-processing** script that turns to **data into information**. This information will be **uploaded** to the **cloud** and **retrieved** through the **android application**.

This project requires us to work in coordination to build each element of this framework.

The proposed system will be described in detail in the next (**Design**) section.

Project Deliverables

Deliverables
Requirements: Functional • Count steps. • Detect Idle time. • Detect body temperature. • Calculate average body temperature. • Show direction (compass). • Data sent to cloud. • Data retrieved from cloud. • Data portrayed as information on android application. • The Application needs to be compatible with older generations of android. Non-Functional • The device should be easy to use. • The application user interface must be friendly and understandable. • The application needs to be light and take less memory/space.
Basic Framework
Main Device [step counter, compass, temperature sensor, Idletimer]
Cloud Realtime-Database
Android Application
Project Report

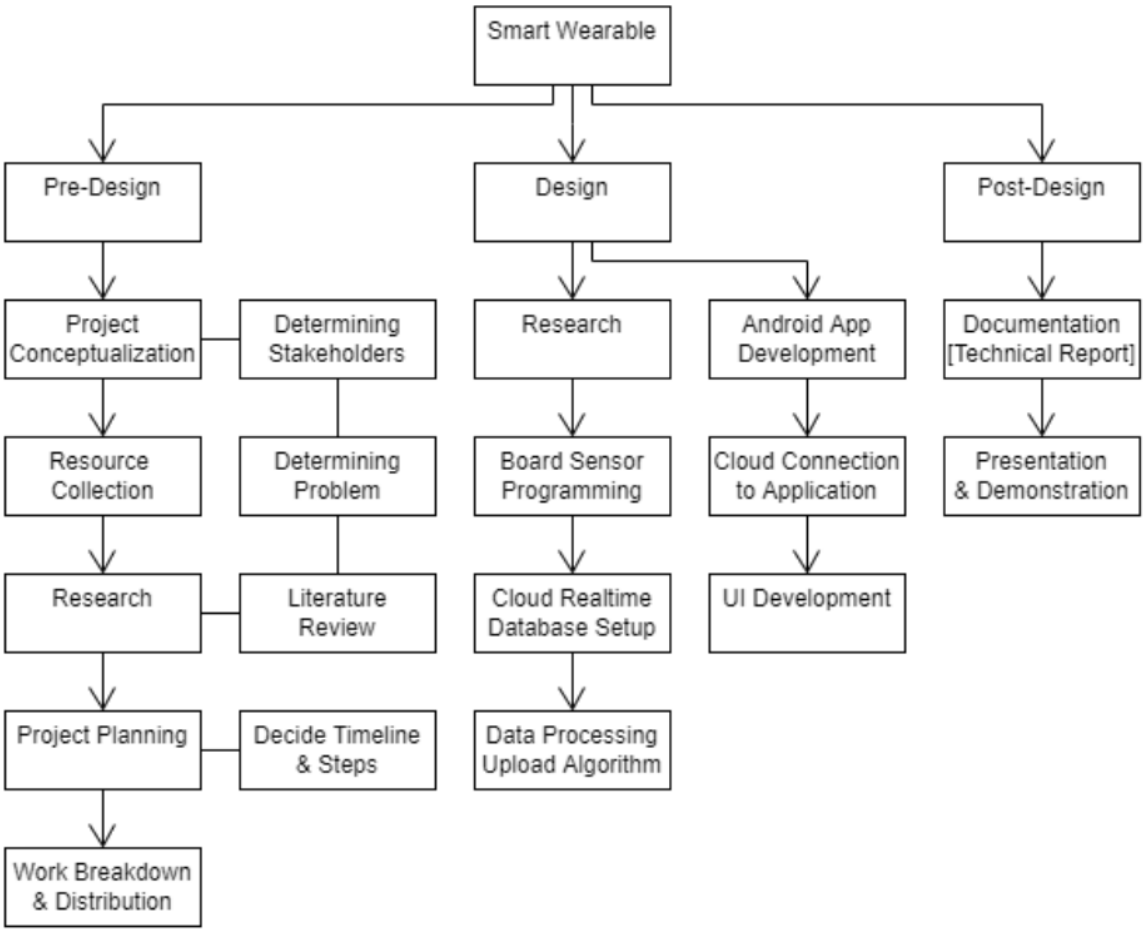
Project Timeline (Gantt Chart & WBS)

Project Timeline given on next page (8).

Task	Start Date (DD/MM)	Duration (Days)	End Date (DD/MM)
Pre-Design:	20/02	11 D	03/03
Project Conceptualization and Planning	20/02	5 D	24/02
Documentation [Technical Report Writing]	24/02	7 D	02/03
Resource Collection	25/02	2 D	26/02
Research & Literature Review	26/02	4 D	1/03
Project Timeline Planning	01/03	2 D	02/03
Work Break Down and Distribution	02/03	1 D	02/03
Design:	03/03	6 D	08/03
Research [Framework, Technology, Software]	03/03	2 D	04/03
STM32 Board Programming	04/03	1 D	04/03
Testing [Obtain Raw Sensor Data]	04/03	1 D	04/03
Cloud Real-Time Database Preparation	04/03	2 D	05/03
Documentation [Technical Report Writing]	04/03	5 D	08/03
Data Processing Algorithm & Upload Algorithm Scripting	05/03	3 D	07/03
Testing [Convert Data to Info and Upload] & troubleshoot	07/03	2 D	08/03
Android Application Development:	08/03	6 D	13/03
Basic Application Build	08/03	1 D	08/03
Cloud Database Connection to Application	08/03	2 D	09/03
Documentation [Technical Report Writing]	08/03	5 D	12/03
Information Retrieval from Database	09/03	2 D	10/03
UI Development	11/03	2 D	12/03
Testing [Information Display & Update Speed]	12/03	2 D	13/03
Project Finalization	11/03	6 D	16/03
Documentation Finalizing & Proof-Reading	11/03	6 D	16/03
Final Testing [System Testing]	12/03	4 D	15/03
Preparation [Demonstration & Presentation]	13/03	2 D	14/03

Project Timeline

Work Breakdown Structure given on next page (9).



Work Breakdown Structure

Design

This section will consist of a proposed system with a description of the hardware and software used, design matrix and a mind map.

Proposed System & Hardware/Software Description

Various sensors will be used to sense raw data which can be turned into information.

- Counting steps- Accelerometer.
- Idle Time- Accelerometer.
- Temperature- Temperature sensor.
- Compass- Magnetometer.

The microcontroller-based board, STM32L475 consists of these sensors and can be used for this project implementation and will provide us with the raw sensor data.

Sensor data is to be processed through Python script to become information and then be uploaded to Firebase Real Time Database (A google cloud service).

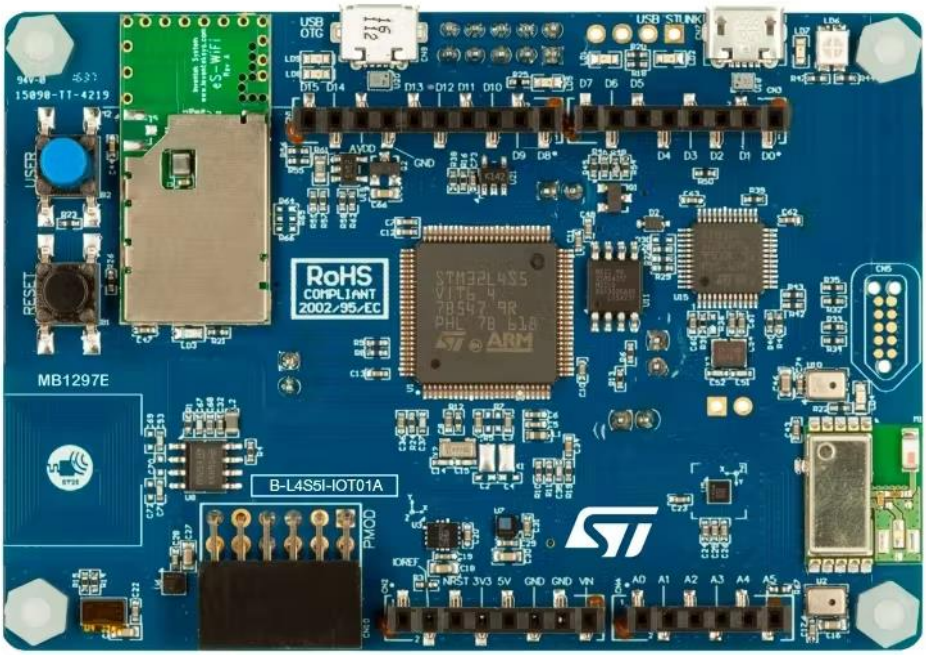
Android Studio will be used to develop the application. Android studio can implement the Software Development Kit for apps provided by Firebase thus establishing connection between the application and the cloud.

Information from the cloud can be retrieved to the android application.

Note that this project's scope is to create an instance of the wearable device, which means the apparatus will not necessarily be fit to be worn but if miniaturised will be perfect.

The STM32L475 MCU Board:

It is an entry level IoT node for the STM32l475 Micro-controller unit.



The board's general specifications include:

- 80 MHz/100 DMIPS
- ARM Cortex M4 core
- 1 Megabyte Flash Memory
- 128 Kilobytes of SRAM
- Wi-Fi, NFC, BLE, Sub-GHz (868 MHz) bands.
- Sensors: Gyro/Accelerometer/Magnetometer, Proximity, Pressure, Humidity, Temperature.
- Arm Mbed Enabled
- Ability to program it with Mbed Online Compiler (ide.mbed.com/compiler).

Accelerometer:

- The accelerometer is designed to detect or measure static/dynamic acceleration using electrochemical properties within.
- An inevitable issue with measuring or detecting acceleration this method is that is that your measures can be affected by the force of gravity constantly acting upon it. This

can technically be turned into a useful feature however can be challenging to be used as a step counter.

Magnetometer:

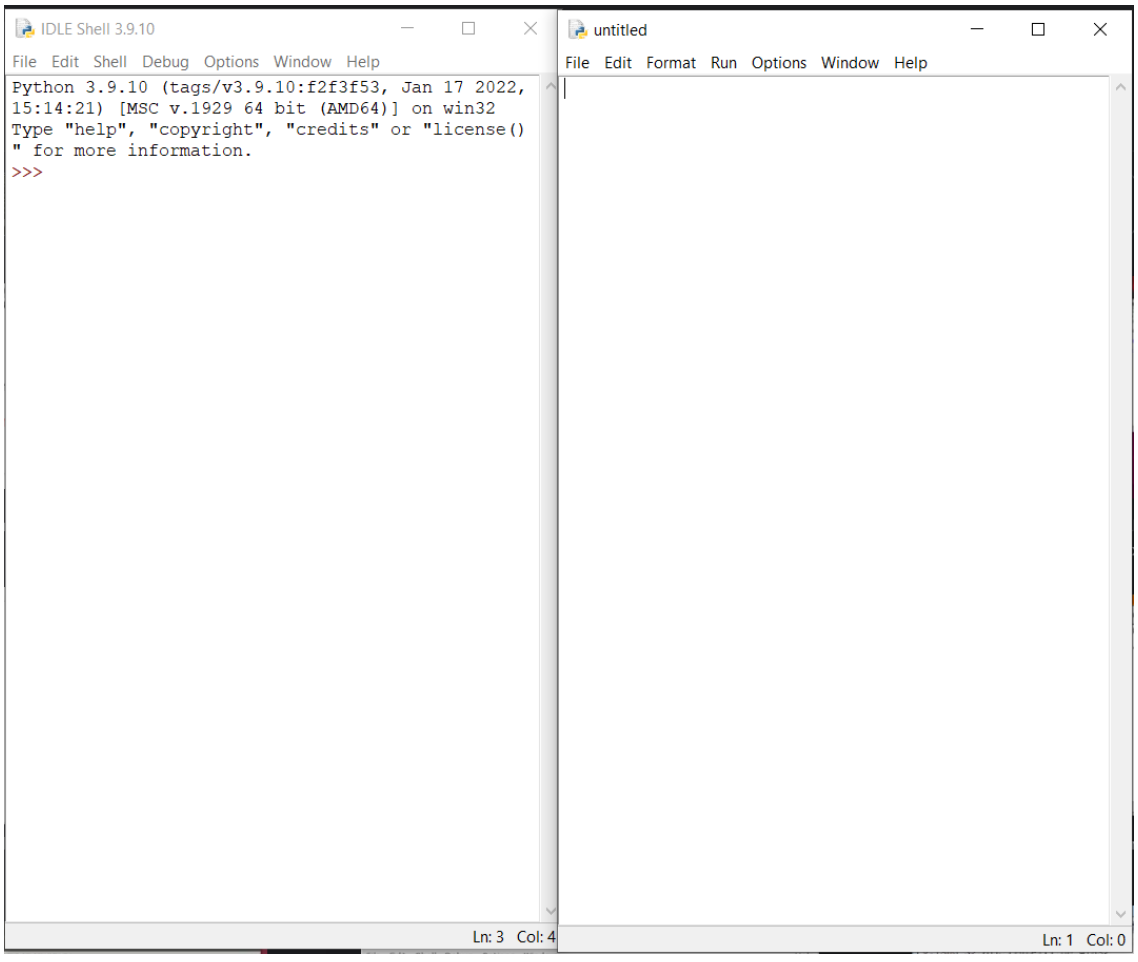
- The magnetometer is used in our case to measure the earths magnetic field, it is calibrated to each direction while the board is oriented horizontally with button side facing the sky.

Temperature Sensor:

- Temperature sensors are usually made of two metal pieces that change physical properties according to the temperature and conduct different amounts of electricity. This analog change can be converted to digital data which will be used to show temperature.

Python:

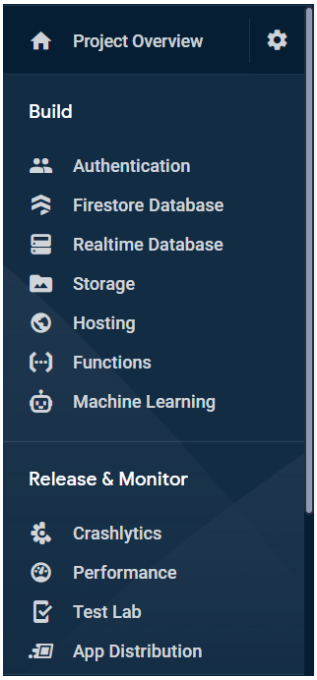
- Python is an open-source high-level programming language mostly used to automate tasks, build software or websites and in our case conduct data analysis and upload to cloud.
- The properties and ease of use in the language makes it suitable for our project.
- The Default IDLE will be used to build scripts.



Firebase Realtime Database:

Google's Firebase platform allows developers to build mobile and web applications. Founded in 2011, it was initially an independent company. Google acquired the platform in 2014, and now it is their flagship app development platform.

The firebase console consists of a decent amount of default applications of which we use the Real-Time database.



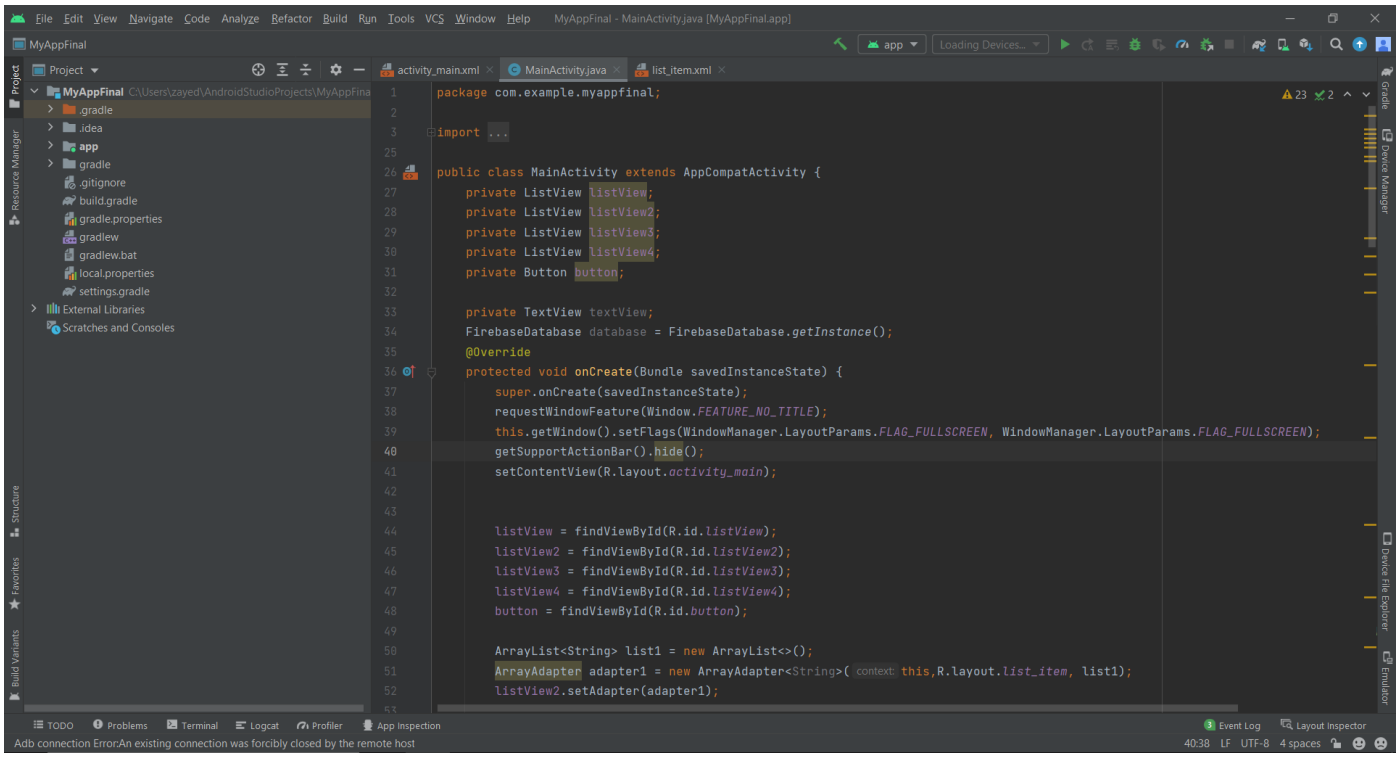
Below is a snip of the database structure in the realtime database tab.

Firebase provides with an SDK consisting of the project and the database, this SDK allows us to implement firebase into our Android Studio environment and application.

Android Studio:

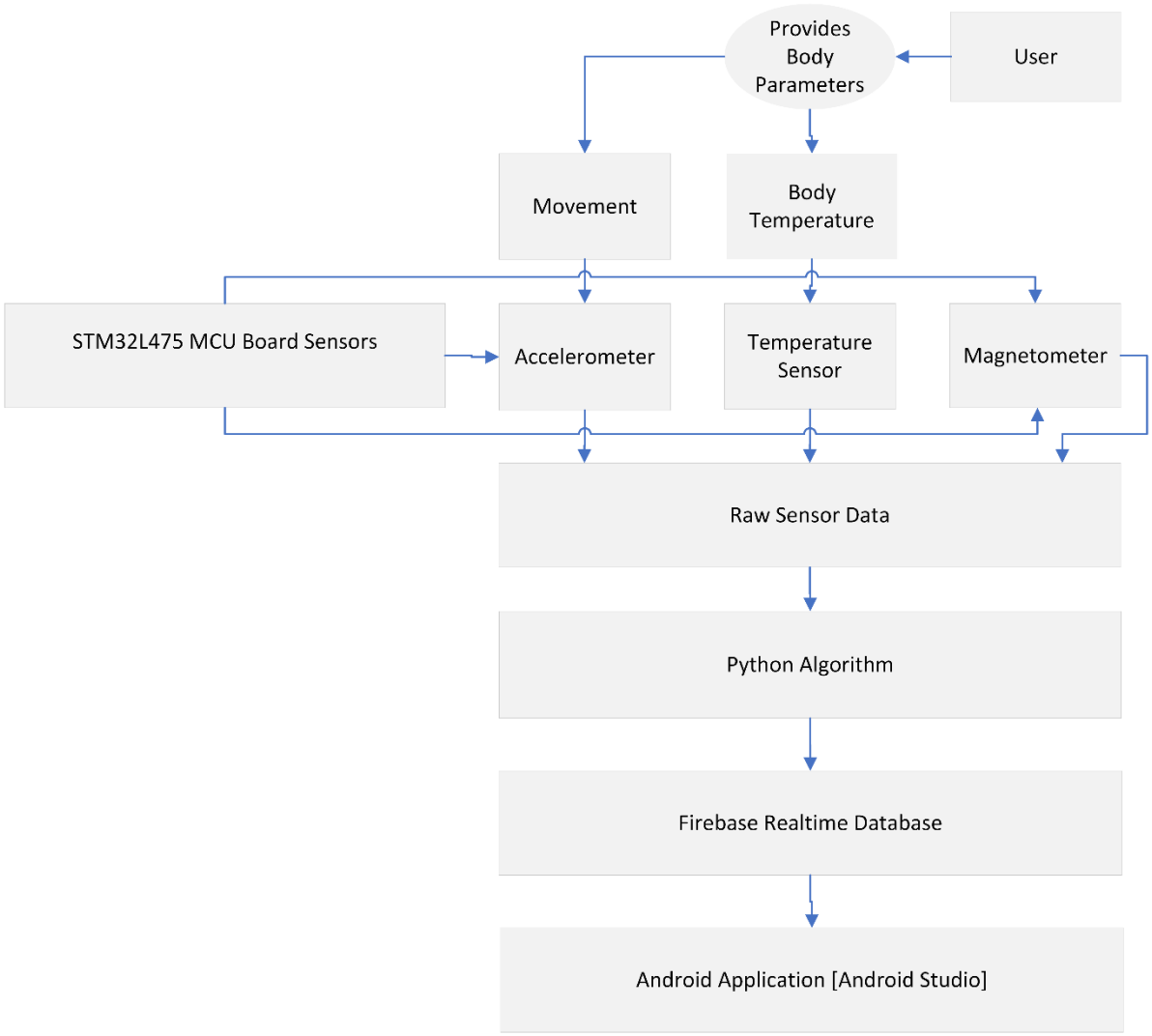
Android Studio is an official integrated development environment for Google's Android operating system. Built on JetBrains IntelliJ IDEA software, it is designed specifically for Android development. The software is free to download & use.

Android Studio includes an virtual device manager that helps us emulate an android device within the environment, makes application development a lot easier and faster.



As for choices we had mainly choice of cloud. So, we used the following matrix to conclude on the best option for us. The rest of our design choices were only based on what looks suitable for us to work with.

Criterion:	Weight (/10)	Amazon Cloud (AWS)	Firebase Google	Azure
Ease of use	10	5	9	6
Budget-Friendly	8	8	9	5
Ease of Integration	8	4	9	8
Range of Features	3	9	6	7
Performance and Reliability	6	8	8	9
Security	3	8	8	9
Storage Space (amount)	2	9	9	9


Provides Body Parameters
User
Movement
Body Temperature
STM32L475 MCU Board Sensors
Accelerometer
Temperature Sensor
Magnetometer
Raw Sensor Data
Python Algorithm
Firebase Realtime Database
Android Application [Android Studio]

Mind Map

Implementation

In this section the implementation of our system will be mentioned in the same order as mentioned in the project timeline.

Our first step was to gather sensor values from the STM32L475 MCU board. The board will be connected to our Windows-based computer through USB and display from Serial Port onto a Serial Monitor

Below is the code uploaded to the board.

```
#include "mbed.h" // Includes the mbed header, this is available on the mbed compiler.
#include "stm32l475e_iot01_tsensor.h" // These libraries are also available
#include "stm32l475e_iot01_magneto.h" // in the BSP library
#include "stm32l475e_iot01_accelero.h" // in the mbed compiler libraries.
DigitalOut led(LED1); // set an LED on the board as digital output LED.
int main()
{   float sensor_value = 0; // Declaring a float value in variable sensor_value
    int16_t pDataXYZ[3] = {0}; // declaring an integer_16t variable consisting of 3 //values
    float pGyroDataXYZ[3] = {0}; // creating a float variable GyroDataXYZ consisting of //three values
    BSP_TSENSOR_Init(); // Initializing temperature sensor
    BSP_MAGNETO_Init(); // Initializing magnetometer
    BSP_ACCELERO_Init();// Initializing accelerometer
    while(1) {
        led = 1;
        sensor_value = BSP_TSENSOR_ReadTemp(); // SensorValue set to the temperature //reading from the temperature sensor
        printf("\nTP = %.2f", sensor_value); // Formatting the values and printing on //Serial Monitor
        led = 0;
        ThisThread::sleep_for(50); //delay code for 50 milliseconds
        led = 1;
        BSP_MAGNETO_GetXYZ(pDataXYZ); // getting the three Magnetometer values and storing in  pDataXYZ
        printf("MX = %d", pDataXYZ[0]); // Formatting first value from pDataXYZ
        printf("MY = %d", pDataXYZ[1]); // Formatting the second value from ‘’
        printf("MZ = %d", pDataXYZ[2]); // Formatting the third value from ‘’
        BSP_ACCELERO_AccGetXYZ(pDataXYZ); // Getting the values from accelerometer sensor and storing the three values in pDataXYZ
        printf("AX = %d", pDataXYZ[0]); // Formatting and printing
        printf("AY = %d", pDataXYZ[1]);// Formatting and printing
        printf("AZ = %d", pDataXYZ[2]);// Formatting and printing
        led = 0;
        ThisThread::sleep_for(50); // delay for 50 milliseconds (0.05 seconds)

    }
}

```

The output received when this code is uploaded to the board is given below:

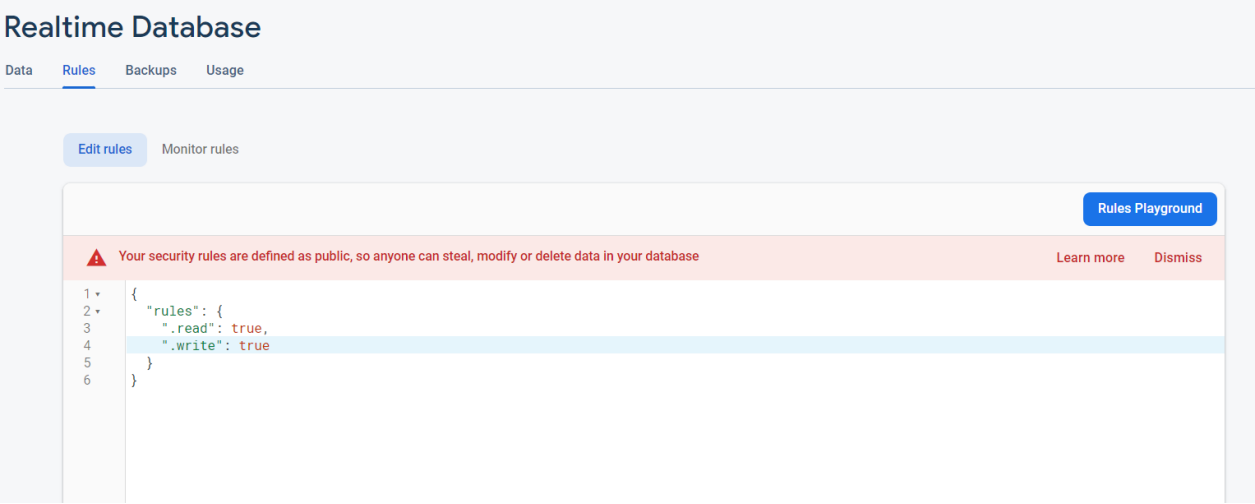
The Serial Monitor used to collect data from the USB port is the Arduino IDE Serial Monitor set at 9600 baud.

```
TP = 29.19MX = 127MY = -308MZ = 618AX = 42AY = 45AZ = 1023

```

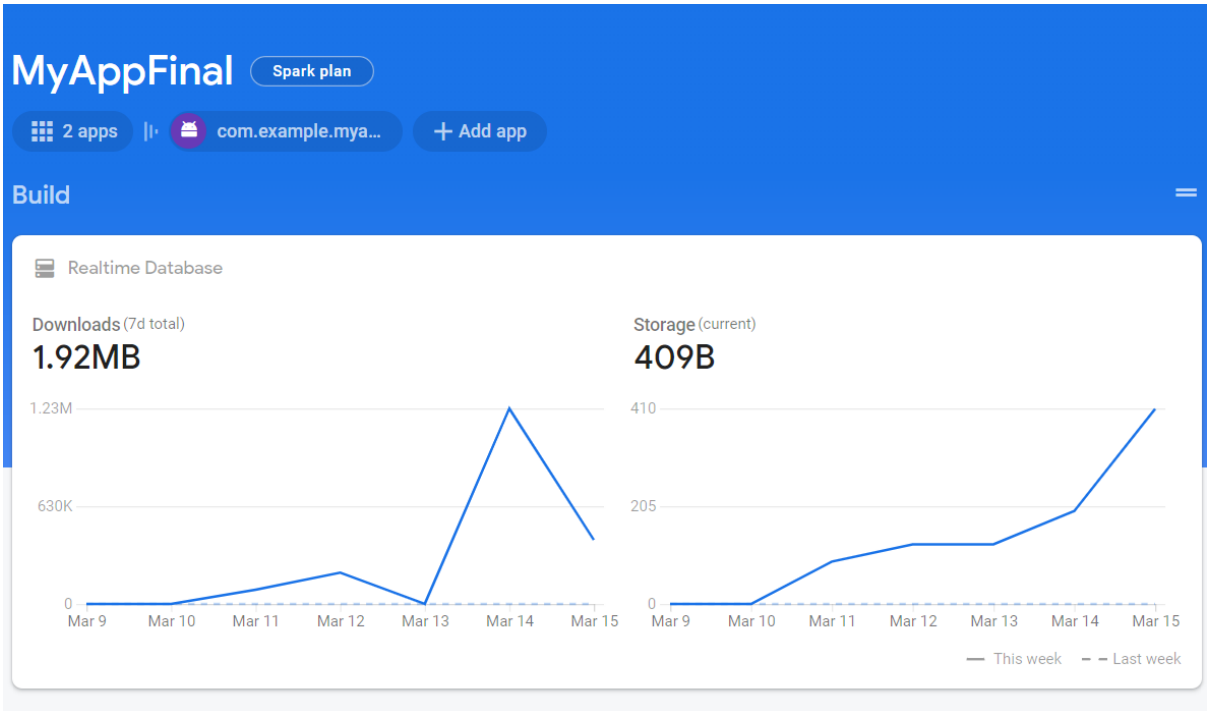
Our next step is to setup the Firebase Realtime Database.

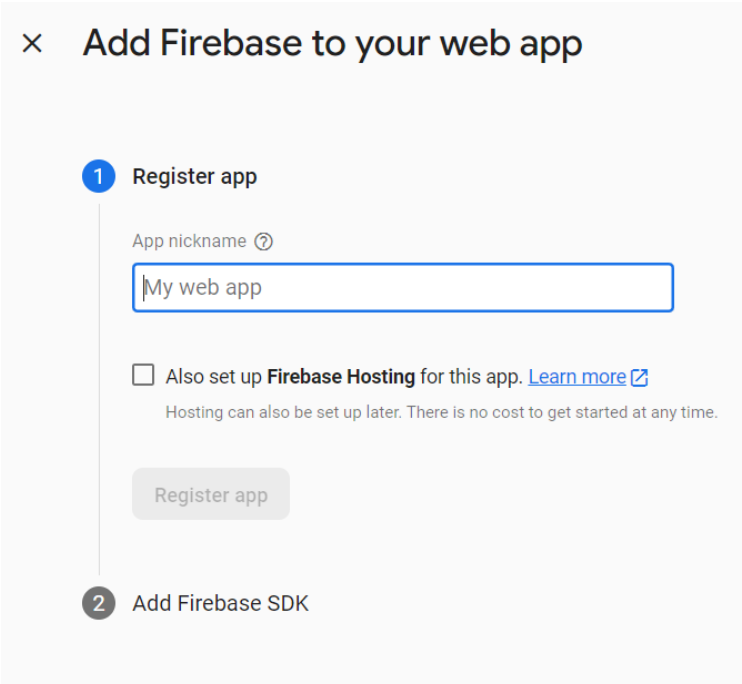
The Setup includes:

- Creating a google account and logging into firebase.google.com with the google account.
- Creating a new project.
- Within the project, creating a Realtime database in the Realtime database tab.
- Setting Rules in the Realtime database to allow access from android.

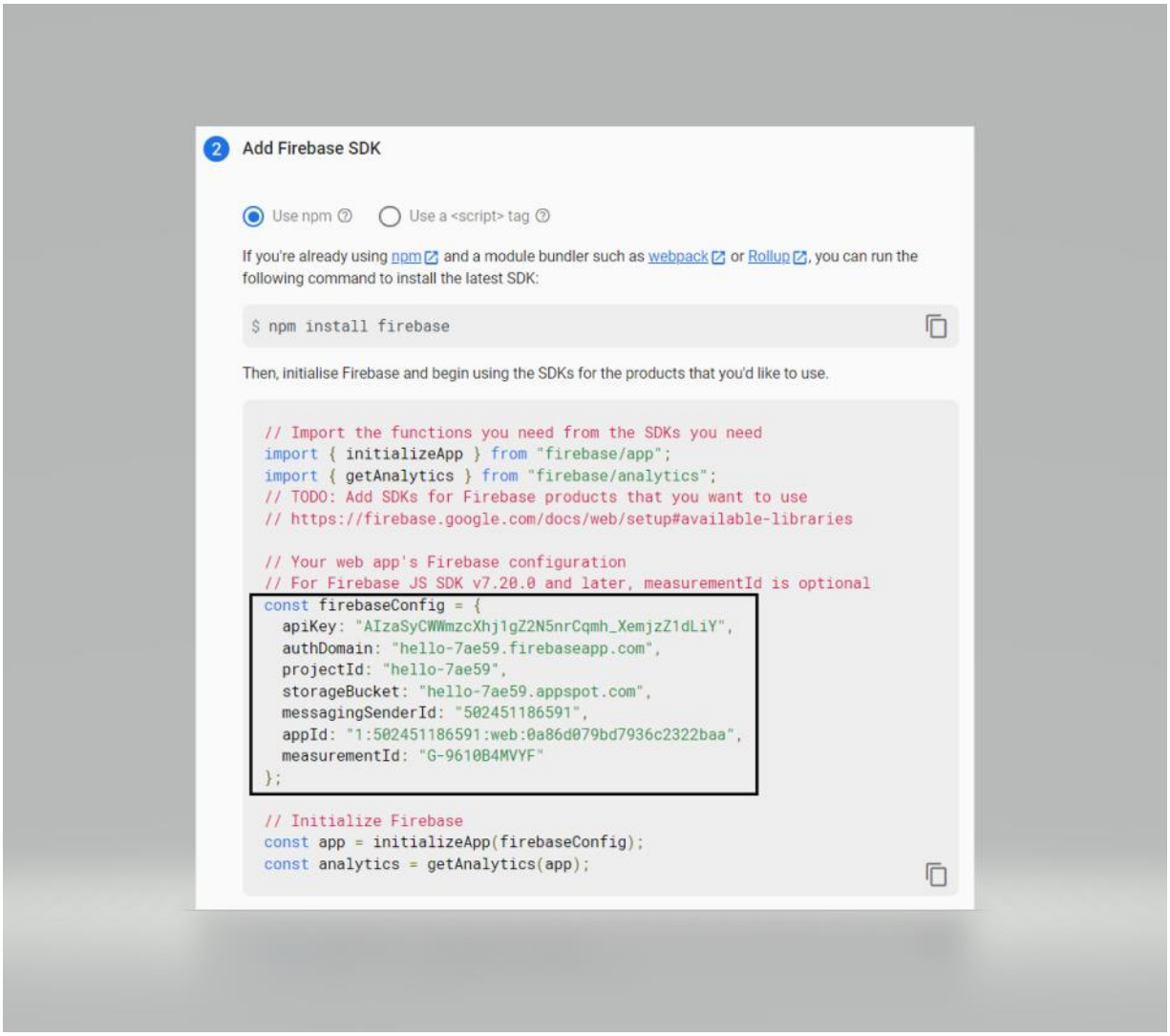


- As seen in the above snip, the read and write values are set to true, this allows access to the database.
- In the project overview tab, Add App (adds application), set name for application, acquire the SDK





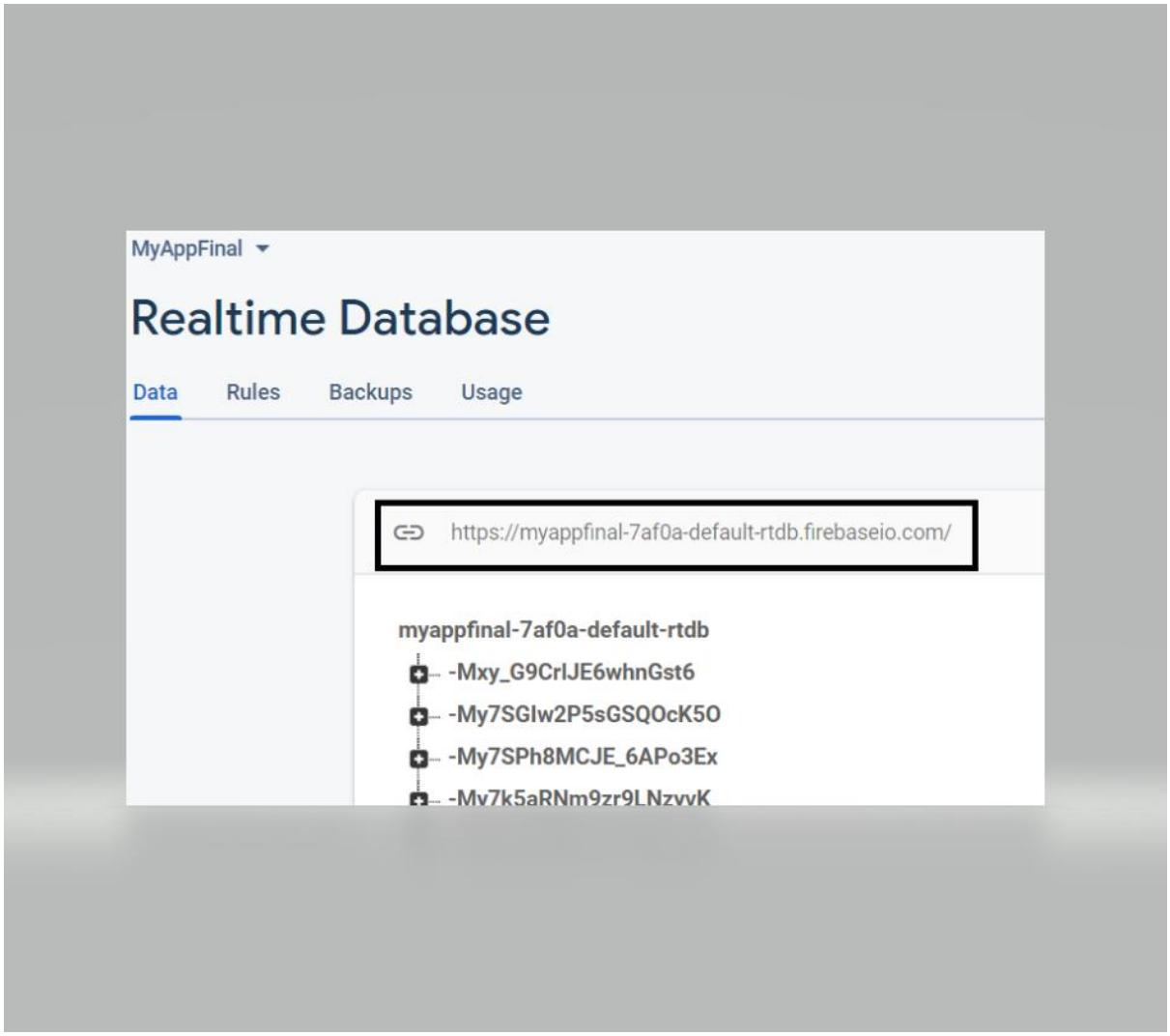
- In the ‘Add Firebase SDK step’, the following will be visible:



- Select and copy the code within the rectangle and we will write it down in our python script as a dictionary like so

```
firebaseConfig ={
    "apiKey": "AIzaSyCHmAg5V0CIfGZhJZwViiA2OCXeFdn09GM",
    "authDomain": "myappfinal-7af0a.firebaseapp.com",
    "projectId": "myappfinal-7af0a",
    "storageBucket": "myappfinal-7af0a.appspot.com",
    "messagingSenderId": "460623334634",
    "appId": "1:460623334634:web:6873f67743df7a93be1b93",
    "measurementId": "G-VVDM998KLD"
    }

```

- The link in the black rectangle in the below snip is the database URL. We need to copy it.



- Add the database URL to the dictionary (“databaseURL”: “putdatabaseURLlinkhere”)

```
firebaseConfig ={
    "apiKey": "AIzaSyCHmAg5V0CIfGZhJZwViiA2OCXeFdn09GM",
  "authDomain": "myappfinal-7af0a.firebaseapp.com",
  "databaseURL": "https://myappfinal-7af0a-default-rtdb.firebaseio.com",#DatabaseURL add
  "projectId": "myappfinal-7af0a",
  "storageBucket": "myappfinal-7af0a.appspot.com",
  "messagingSenderId": "460623334634",
  "appId": "1:460623334634:web:6873f67743df7a93be1b93",
  "measurementId": "G-VVDM998KLD"
    }

```

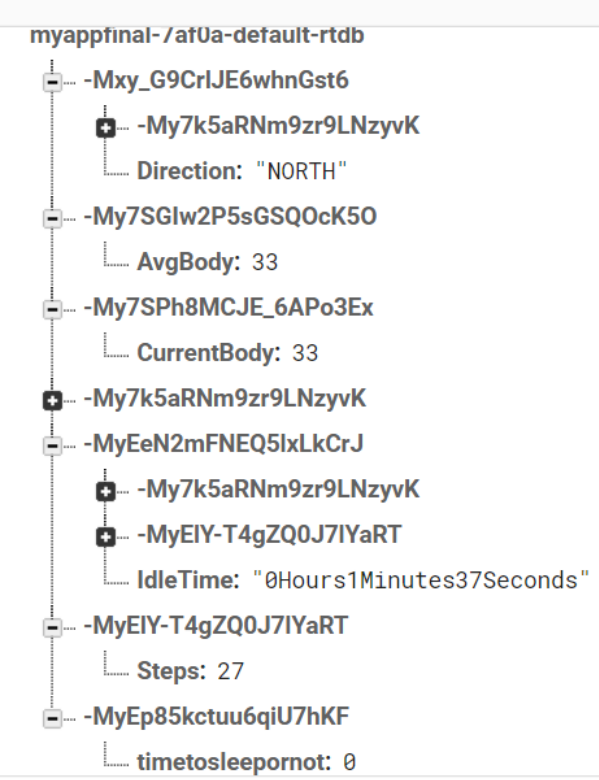
- Import the library module ‘pyrebase’. [can be installed through cmd, pip3 install pyrebase or pip3 install pyrebase4
- Now we establish connection to the Realtime database.

```
import pyrebase

firebaseConfig ={
    "apiKey": "AIzaSyCHmAg5V0CIfGZhJZwViiA2OCXeFdn09GM",
  "authDomain": "myappfinal-7af0a.firebaseapp.com",
  "databaseURL": "https://myappfinal-7af0a-default-rtdb.firebaseio.com",
  "projectId": "myappfinal-7af0a",
  "storageBucket": "myappfinal-7af0a.appspot.com",
  "messagingSenderId": "460623334634",
  "appId": "1:460623334634:web:6873f67743df7a93be1b93",
  "measurementId": "G-VVDM998KLD"
    }
firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()

```

- Create fields with sample data in the Realtime Database

```
import pyrebase
firebaseConfig ={
    "apiKey": "AIzaSyCHmAg5V0CIfGZhJZwViiA2OCXeFdn09GM",
  "authDomain": "myappfinal-7af0a.firebaseapp.com",
  "databaseURL": "https://myappfinal-7af0a-default-rtdb.firebaseio.com",
  "projectId": "myappfinal-7af0a",
  "storageBucket": "myappfinal-7af0a.appspot.com",
  "messagingSenderId": "460623334634",
  "appId": "1:460623334634:web:6873f67743df7a93be1b93",
  "measurementId": "G-VVDM998KLD"
    }
firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()
#For creating Field
databasedata ={"Direction":"NORTH"}
db.push(databasedata)
databasedata ={"AvgBody":"33"}
db.push(databasedata)
databasedata ={"CurrentBody":"33"}
db.push(databasedata)
databasedata ={"Idletime":"IdleTimE"}
db.push(databasedata)
databasedata = {"Steps":"27"}
db.push(databasedata)
databasedata = {"timetosleepornot":"0"}

```

- At every db.push, a branch is created in the Realtime database with the field with the name of the key in the dictionary and the values of the value in the dictionary as shown in the below snip.



Now we are connected to the database and have pushed fields. We will use these fields for our new information which is processed by the data processing algorithm. The current values of each field will be updated by the new information.

Data Processing & Updating Algorithm

The algorithm first takes the data that's being sent by the board on the serial, and puts the serial data into the variable 'data'

Refer to the code for explanations

```
1.  import serial # pip3 install pyserial
2.  import datetime #pip3 install datetime
3.  import os # pip3 install os
4.  import time #pip3 install pytime or pip3 install time
5.  import pyrebase #pip3 install pyrebase4 or pip3 install pyrebase
6.  ser = serial.Serial("COM7", 9600) # Establishes serial connection to COM7[USB PORT],
    BAUD rate = 9600 and stores serial data
7.  # in variable ser
8.  ser.flushInput()# clear ser
9.  firebaseConfig ={
10.     "apiKey": "AIzaSyCHmAg5V0CIfGZhJZwViiA2OCXeFdn09GM",
11.     "authDomain": "myappfinal-7af0a.firebaseapp.com",
12.     "databaseURL": "https://myappfinal-7af0a-default-rtdb.firebaseio.com",
13.   "projectId": "myappfinal-7af0a",
14.   "storageBucket": "myappfinal-7af0a.appspot.com",
15.   "messagingSenderId": "460623334634",
16.   "appId": "1:460623334634:web:6873f67743df7a93be1b93",
17.   "measurementId": "G-VVDM998KLD"
```

```
    } #Mentioning the properties of our firebase web application including the databaseURL of our realtime database
firebase = pyrebase.initialize_app(firebaseConfig) #initializing the application to variable firebase
db = firebase.database() # storing the database application that was initialized as db
list1 = []
step =0
stepsfinal = 0
avg_temperature = 0
idle = 0
stage = 0
idlehours = []
idleminutes = []
idleseconds = []
endhours = []
endminutes= []
endseconds = []
idlenumber = 0
idletimes = []
idledurationhour = 0
idledurationminute = 0
idledurationsecond = 0
idledurationhours = []
idledurationminutes= []
idledurationseconds = []
sleepstarttimes = []
avg_sleepstarttime = 99
timetosleepnow = 0
while True:
    try:
        ser_bytes = ser.readline() #Reads a line of of the serial data and stores in variable 'ser_bytes'
        decoded_bytes = ser_bytes[0:len(ser_bytes)-2].decode("utf-8") #decodes the line into readable data
        now = datetime.datetime.now()   #Creating a timestamp and storing it in variable 'now'
        now = now.strftime("%Y-%m-%d %H:%M:%S") #Formatting the timestamp
        data = str( "'{}',{}\r\n".format(now,decoded_bytes) ) #storing the timestamp and the readable data into the variable 'data'
# Refer to Serial Output given earlier for this.
#This is a sorting method that picks out a value and assigns it to a corresponding variable
        index = data.find("MX") # For example for Magnetometer X axis values
        # We first find in the serial data 'MX' then find 'MY' and knowing
        # that the value of magnetometer X axis is between these two strings
        # we will take out the value thats between those strings and assign
        # the value to variable 'Magneto_X'
        index2 = data.find("MY")
        index3 = data.find("MZ")
        index4 = data.find("AX")
        index5 = data.find("AY")
        index6 = data.find("AZ")
        index7 = data.find("TP") # Finds TP in the serial data
        Magneto_X = int((data[index+5:index2]))
        Magneto_Y = int((data[index2+5:index3]))
        Magneto_Z = int((data[index3+5:index4]))
        Accelero_X = int((data[index4+5:index5]))
        Accelero_Y = int((data[index5+5:index6]))
        Accelero_Z = int((data[index6+5:]))
        Temperature = int(round(float((data[index7+5:index])))) #picks the value after 'TP' and before 'MX' and rounds it up to
        #integer and stores it in variable 'Temperature'
        current_temperature = Temperature
        t = time.localtime()
        current_time = time.strftime("%H:%M:%S", t)
        #Now we will create time stamps of seconds, minutes, hours
        seconds = time.localtime()
        seconds_time = time.strftime("%S",seconds)
        usableseconds = int(seconds_time)
        #the variable 'usableseconds' holds the current second of the current time
```

```
        #for example the time is 10:25:23, if we mention secondstime = usableseconds
        # then secondstime = 23

        minutes = time.localtime()
        minutes_time = time.strftime("%M",minutes)
        usableminutes = int(minutes_time)
        #the variable 'usableminutes' holds the current minute of the current time
        # for example the time is 10:25:23, if we mention minutestime = usablemintues
        # then minutestime = 25

        hours = time.localtime()
        hours_time = time.strftime("%H", hours)
        usablehours = int(hours_time)
        # The variable 'usablehours' holds the current hour of the current time
        # for example the time si 10:25:23, if we mention hourstime = usablehours
        # then hourstime = 10
#Skip From here if reading code for first time
        #This is where we start calculating the average time at which the user takes rest/sleep
        # sleepstarttimes is the list where all the hour at which the user takes rest/sleep
        # so we will calculate when the user will probably take rest/sleep again by taking an average
          # on the hour that they previously took rest/sleep on
          lensleepstarttimes = len(sleepstarttimes)
          if (len(sleepstarttimes)>0):
              avg_sleepstarttime = sum(sleepstarttimes)/lensleepstarttimes
          else:
              pass
          if (avg_sleepstarttime-1)<usablehours<(avg_sleepstarttime+1):
              timetosleepnow = 1 # if the current time comes close to average rest/sleep time or schedule
              # the variable timetosleepnow will turn to 1 this indicates its time to rest/sleep
          else:
              timetosleepnow = 0 #if not timetosleepnow will be 0 indicating not time to rest/sleep

  #Stop Skip here if reading code for first time
          if (usableseconds%20==0): #Every 20 seconds
              time.sleep(0.01)
              list1.append(Temperature) # the list list1 has the current temperature added to it
              time.sleep(0.01)
          else:pass
          if len(list1)>0: # if list1 has an element or more
                           # we start calculating the average temperature
              avg_temperature = int(round(sum(list1)/len(list1)))
          else:pass

          # Below is the algorithm to count steps
          # This algorithm basically detects if theres acceleration
          #and if theres acceleration it adds steps
          if (bool(70<Accelero_Z<130)== False)and (bool(-850>Accelero_Y>-1030)==False) and (bool(-950>Accelero_X>-1050)==False) and (bool(20>Accelero_Z>-10)==False):
              step = step+1
              steps = step
              stepsfinal = round(steps/3.0) # To further improve accuracy a buffer is added here
              if idle>0.30: #Skip if reading code first time # this is the code responsible for determining if person is NOT idling ANYMORE
                  idle = idle -0.05 # when movements are detected, the buffer starts getting drained
              else:pass
                           #Stop Skip here if reading code for first time
          elif(idle<2): #This is where we start programming the idle.
              #The idea is to judge that the person is not moving if theres no steps detected
              # the variable 'idle' acts like a buffer in between if the person is idle and if the person is not idle
```

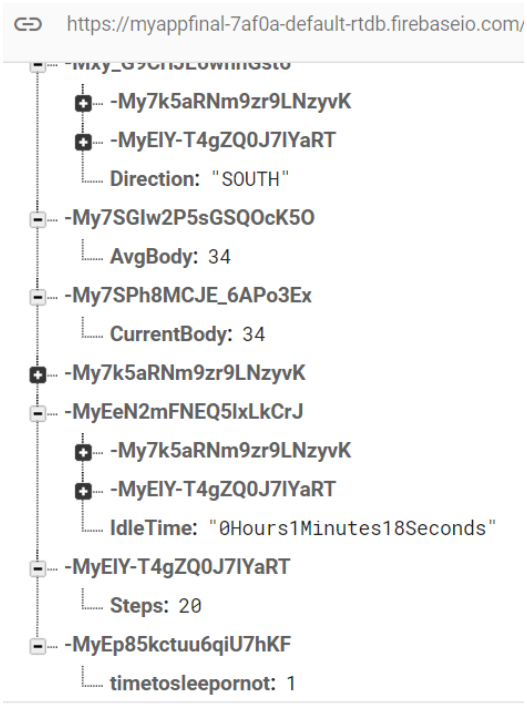
```
137.             # as the person doesnt move the buffer keeps filling up as seen later the
    limit of the buffer has
138.             # been set at 1.7, the rate at which the buffer gets added up is 0.02
139.             idle = idle+0.02
140.         else:pass
141.         if (idle>1.7) and (stage== 0): #We use the variable stage like a switch, its
    states go from 0 to 1 or 1 to 0 only
142.             # so if 'stage' is off or 0 and if idle>1.7 initiating IDLE
143.             idlenumber = idlenumber+1 #idle number is used to calculate which'th idle
    this is as in is it the first idle
144.             # or the second or the third or the nth, the use for this will be
    understood in later parts of the code.
145.             stage = 1 #setting 'stage' as 1 or on. (this is an indication that idle
    WAS started)
146.             idlefinal =1 #indicates that idle has started
147.             idlehour = usablehours #setting the variable 'idlehour' to the hour the
    idle started.
148.             idleminute = usableminutes #setting the variable 'idleminute' to the
    minute the idle started
149.             idlesecond = usableseconds #setting the variable 'idlesecond' to the
    second the idle start
150.             idlehours.append(idlehour)# the list idlehours stores all the hourstamps
    when the idles have started
151.             idleminutes.append(idleminute) #the list idlemintutes stores all the
    minute stamps when the idles have started
152.             idleseconds.append(idlesecond) #the list idleseconds stores all the second
    stamps when the idles started
153.             # the hour, minute, second of when the idle started is stored in the
    respective lists.
154.         else: pass
155.         if (idle<1.4) and (stage==1): #if idle buffer is less than 1.4 (after person
    is NOT idle ANYMORE)
156.                                       # and if idle stage was initiated before this or
    stage = 1 or ON
157.             stage = 0                 # switching back stage to 0 or OFF
158.             idlefinal = 0             # switching off idle status to 0
159.             idle = 0                  # Resetting the buffer
160.             endhour = usablehours     #Storing the hour that the idle ended on the
    endhour variable
161.             endminute = usableminutes #Storing the minute that the idle ended on the
    endminute variable
162.             endsecond = usableseconds #storing the second that the idle ended on the
    end second variable
163.             endhours.append(endhour)  #Appending the ending hour to the endhours list
164.             endminutes.append(endminute) #Appending the ending minute to the
    endminutes list
165.             endseconds.append(endsecond) #Appending the ending second to the
    endseconds list
166.             idlenumberindex = idlenumber-1
167.             # the hour, minute, second of when the idle ended is stored in their
    respective lists.
168.
169.             #below is the algorithm to calculate idle durations
170.             # here we start to calculate how long the person has been idle for and we
    will add each duration to a list
171.             # there will be three lists, that hold how many hours person has been idle
    for, how many minutes, how many seconds
172.             # the basic idea is to subtract the start time from the end time
173.             # i.e. endtime - starttime, of each time bracket (hours, minutes, seconds)
174.
175.             idledurationhour = endhours[idlenumberindex]-idlehours[idlenumberindex]
176.             # the endhours[idlenumberindex] points out to the end hour time of this
    particular idle instance
177.             # the idlehours[idlenumberindex] points out to the start hour time of this
    particular idle instance
178.             idledurationhours.append(idledurationhour) #Appending the difference i.e.
    idle duration in hours, to the
179.             #idle duration hours list which consists of all the idle durations in
    hours.
180.
```

```
            if (idledurationhour>3): #detects rest state as in the person is physically resting (idle for longerperiods)
                # this can be considered as sleep also
                sleepstarttimes.append(idlehour)# if the current idle state is longer than 3 hours it is considered as
                #resting or sleeping and the hour (time) that the rest started will be appended to the
                #list named sleepstarttimes
            else:pass

            if (endseconds[idlenumberindex]>=idleseconds[idlenumberindex]): # our end time - start time algorithm will work
                #properly only if end time is higher than start time i.e.
                # if person starts idle at 10:01:01 and ends idle at 10:02:02 idle duration = (10-10):(02-01):(02-01) = 0:01:01
                idledurationsecond = endseconds[idlenumberindex]-idleseconds[idlenumberindex]
                idledurationseconds.append(idledurationsecond) # append idle duration to its list
            else:
                idledurationsecond = (endseconds[idlenumberindex]+60)-idleseconds[idlenumberindex]
                idledurationseconds.append(idledurationsecond)

            if endminutes[idlenumberindex]>=idleminutes[idlenumberindex]:
                idledurationminute = endminutes[idlenumberindex]-idleminutes[idlenumberindex]
                idledurationminutes.append(idledurationminute)
            else:
                idledurationminute = (endminutes[idlenumberindex]+60)-idleminutes[idlenumberindex]
                idledurationminutes.append(idledurationminute)
            if sum(idledurationseconds)>=60: #if total seconds get more than 60 then it will be counted as a minute and
                #appended to minute and substracted from seconds
                sumidleseconds = sum(idledurationseconds)
                q, mod = divmod(sumidleseconds, 60)
                idledurationseconds.append((q*60)*(-1))
                idledurationminutes.append(q)
            if sum(idledurationminutes)>=60: #if total seconds gets more than 60 then it will be counted as an hour and
                #appended to hours and substracted from minutes
                sumidleseconds = sum(idledurationminutes)
                q, mod = divmod(sumidleminutes, 60)
                idledurationminutes.append((q*60)*(-1))
                idledurationhours.append(q)
            else:pass
            #formatting the durations to strings
            stringidledurationhours = str(sum(idledurationhours))+"Hours"
            stringidledurationminutes = str(sum(idledurationminutes))+"Minutes"
            stringidledurationseconds = str(sum(idledurationseconds))+"Seconds"
            idletimefinalstring = stringidledurationhours + stringidledurationminutes + stringidledurationseconds
            #print(idletimefinalstring)
        else:pass
        #the compass
        # it has been calibrated by checking which physical orientation of the board points to which direction on a real compass
        if 0<Magneto_X and Magneto_X <100 and -5<Magneto_Z and Magneto_Z<25:
            direction = "EAST"
        elif 340<Magneto_X and Magneto_X<500 and -5<Magneto_Z and Magneto_Z<25 and Magneto_Y>-400 and Magneto_Y<-150:
            direction = "NORTH"
        elif 700<Magneto_X and Magneto_X<790 and -5<Magneto_Z and Magneto_Z<25:
            direction = "WEST"
        elif 160<Magneto_X and Magneto_X<300 and Magneto_Y<-700 and Magneto_Y>-1000:
            direction = "SOUTH"
        else:pass

```

```
235.          if (usableseconds%17==0):# Every 17 seconds the fields created earlier will be
    updated with the new information
236.              print("X")
237.              db.child('-MyElY-T4gZQ0J7lYaRT').update({"Steps":stepsfinal})
238.              time.sleep(0.05)
239.              db.child('-Mxy_G9CrlJE6whnGst6').update({"Direction":direction})
240.              time.sleep(0.05)
241.              db.child('-My7SPh8MCJE_6APo3Ex').update({"CurrentBody":Temperature})
242.              time.sleep(0.05)
243.              db.child('-My7SGIw2P5sGSQOcK5O').update({"AvgBody":avg_temperature})
244.              time.sleep(0.05)
245.              db.child('-MyEeN2mFNEQ5lxLkCrJ').update({"IdleTime":idletimefinalstring})
246.              time.sleep(0.05)
247.              db.child('-
    MyEp85kctuu6qiU7hKF').update({"timetosleepornot":timetosleepnow})
248.              time.sleep(0.05)
249.          else:pass
250.      except:
251.          print("*")
```

Data Processing Algorithms Involved and Their uses:

- Parsing raw sensor values, placing them in variables and turning them into information.
- Using Magnetometer value variables to make compass.
- Using temperature values variables.
- Using Accelerometer value variables to detect movement, steps, idle time, sleep/rest time.
- Included with this is also an algorithm that can predict the user's sleep schedule by their sleep patterns note however this is only for the sleep, not wake-up.
- Uploading data to the cloud Real Time database.

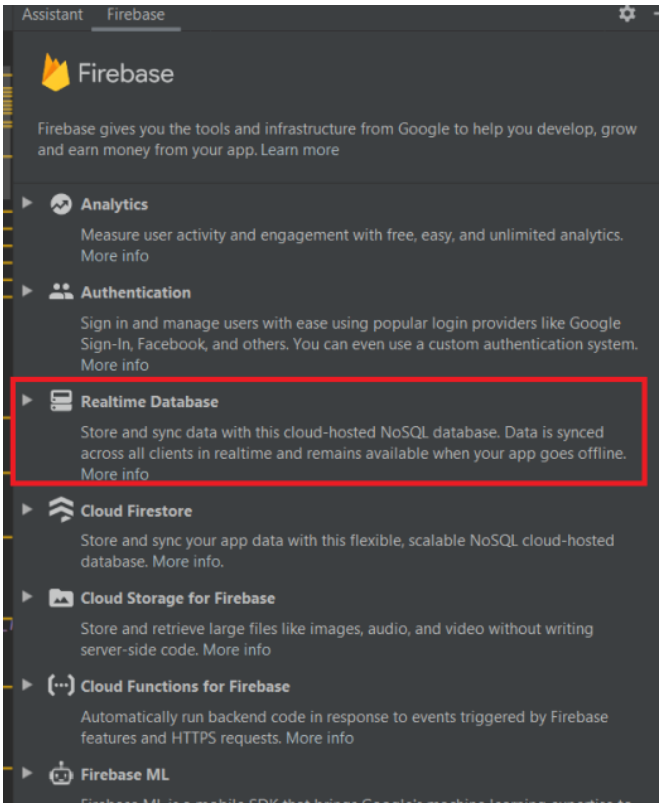


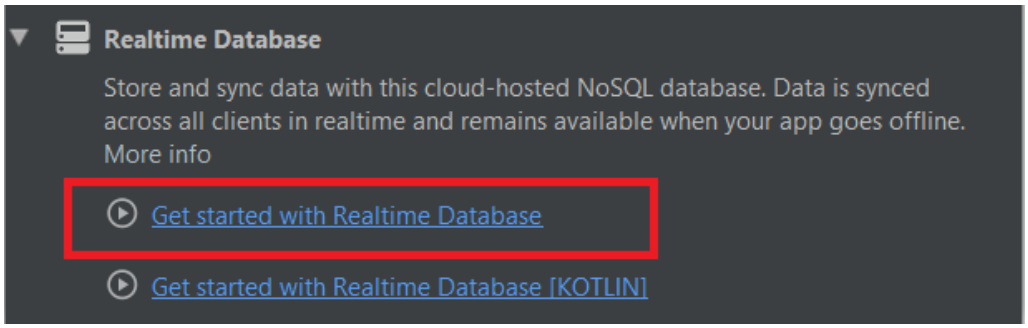
Now we must build the Android application in Android Studio.

- We used an empty activity app [a template usually available when creating a new project]
- After creating the project (empty activity) we will have to connect to Firebase.
- We did this by first logging into android studio with Google Account.

Connecting to Firebase through Firebase assistant

- Assistant is available in the Tools section of Android Studio





After Connecting to Firebase, we add the SDK's it prompts us to add.

In our Android Studio project, we have now activity_main.xml, MainActivity.java

MainActivity.java

```
package com.example.myappfinal;

import androidx.annotation.NonNull;
import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.NotificationCompat;
import androidx.core.app.NotificationManagerCompat;

import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.os.Build;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.sql.Array;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
    private ListView listView; // Creating ListView object
    private ListView listView2; // Creating ListView object
    private ListView listView3;// Creating ListView object
    private ListView listView4;// Creating ListView object
    private ListView listView5;// Creating ListView object
    private ListView listView6;// Creating ListView object
    private Button button;// Creating Button Object


    FirebaseDatabase database = FirebaseDatabase.getInstance();
        //Creating an object of FirebaseDatabase

    @RequiresApi(api = Build.VERSION_CODES.O)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

     requestWindowFeature(Window.FEATURE_NO_TITLE);
// Start of Code to make app full screen

this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        getSupportActionBar().hide();
        setContentView(R.layout.activity_main);
// End of Code to make app full screen
```

```
// Referring each listView object to the listViews from the
activity_main.xml file.

        listView = findViewById(R.id.listView);
        listView2 = findViewById(R.id.listView2);
        listView3 = findViewById(R.id.listView3);
        listView4 = findViewById(R.id.listView4);
        listView5 = findViewById(R.id.listView5);
        listView6 = findViewById(R.id.listView6);
        button = findViewById(R.id.button);

// Referring button object to the button from the activity_main.xml
//Below we create an array list of type string and name 'list'
        ArrayList<String> list = new ArrayList<>();

// We create an adapter object named 'adapter'
// the adapter allows us to connect the listView to the list

      ArrayAdapter adapter = new ArrayAdapter<String>(this,
R.layout.list_item, list);
        listView.setAdapter(adapter); //setting adapter

// We do the same for all listViews with different lists and adapters

       ArrayList<String> list1 = new ArrayList<>();
        ArrayAdapter adapter1 = new
ArrayAdapter<String>(this,R.layout.list_item, list1);
        listView2.setAdapter(adapter1);

        ArrayList<String> list2 = new ArrayList<>();
        ArrayAdapter adapter2 = new
ArrayAdapter<String>(this,R.layout.list_item,list2);
        listView3.setAdapter(adapter2);

        ArrayList<String> list3 = new ArrayList<>();
        ArrayAdapter adapter3 = new
ArrayAdapter<String>(this,R.layout.list_item,list3);
        listView4.setAdapter(adapter3);

        ArrayList<String> list4 = new ArrayList<>();
        ArrayAdapter adapter4 = new ArrayAdapter<String>(this,
R.layout.list_item, list4);
        listView5.setAdapter(adapter4);

        ArrayList <String> list5 = new ArrayList<>();
        ArrayAdapter adapter5 = new
ArrayAdapter<String>(this,R.layout.list_item,list5);
        listView6.setAdapter(adapter5);

// Now we create DatabaseReference object named reference that will access
the database and act as a pointer to the data

        DatabaseReference reference =
FirebaseDatabase.getInstance().getReference().child("-MyElY-
T4gZQ0J7lYaRT");

// We do this for each data field

        DatabaseReference reference1 =
FirebaseDatabase.getInstance().getReference().child("-
```

```
Mxy_G9CrlJE6whnGst6");

        DatabaseReference reference2 =
FirebaseDatabase.getInstance().getReference().child("-
My7SPh8MCJE_6APo3Ex");

        DatabaseReference reference3 =
FirebaseDatabase.getInstance().getReference().child("-
My7SGIw2P5sGSQOcK5O");

        DatabaseReference reference4 =
FirebaseDatabase.getInstance().getReference().child("-
MyEeN2mFNEQ5lxLkCrJ");

        DatabaseReference reference5 =
FirebaseDatabase.getInstance().getReference().child("-
MyEp85kctuu6qiU7hKF");

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
            NotificationChannel channel = new
NotificationChannel("MyNotification","NotificationChannel",
NotificationManager.IMPORTANCE_DEFAULT);
            NotificationManager manager =
getSystemService(NotificationManager.class);
            manager.createNotificationChannel(channel);
        }
// Here we add a value event listener to reference 5 which is a reference
// to one of the fields in the database.
// What this does is waits for change in values
// when value changes the code under  the public void onDataChange gets
executed. In our case its to convert data to string and display it on app/
        reference5.addValueEventListener(new ValueEventListener() {


            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                for (DataSnapshot snapshot5: snapshot.getChildren()){
                    list5.clear();
                    list5.add(snapshot5.getValue().toString());

                }
// notifies the adapter that data has had a change in it.
                adapter5.notifyDataSetChanged();
// This is the code to send notification to user
// Sending them a care message saying they might wanna take rest
// This care message will be sent when the data processing script
// thinks that user usually takes rest [Sleep/Rest schedule prediction]
// list5 contains 1 if the data processing script sends 1 to the firebase
// which means it is time for user's sleep/rest
                if(list5.contains("1")){
// Code for building notification
                    NotificationCompat.Builder builder = new
NotificationCompat.Builder(MainActivity.this, "Notification");
                    builder.setContentTitle("Might Wanna Take A Rest");
                    builder.setContentText("Chill out, Relax!");

builder.setSmallIcon(R.drawable.ic_launcher_background);
                    builder.setAutoCancel(true);
// Code for executing notification
                    NotificationManagerCompat managerCompat =
NotificationManagerCompat.from(MainActivity.this);
```

```
                managerCompat.notify(1,builder.build());
            }
            else {
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });

    reference4.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            for (DataSnapshot snapshot4: snapshot.getChildren()){
                list4.clear();
                list4.add(snapshot4.getValue().toString());
            }
            adapter4.notifyDataSetChanged();

        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });



    reference3.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            for (DataSnapshot snapshot3: snapshot.getChildren()){
                list3.clear();
                list3.add(snapshot3.getValue().toString());
            }
            adapter3.notifyDataSetChanged();
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });

    reference2.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            for(DataSnapshot snapshot2: snapshot.getChildren()){
                list2.clear();
                list2.add(snapshot2.getValue().toString());
            }
            adapter2.notifyDataSetChanged();

        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
```

```
            }
        });


        reference1.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {

                for(DataSnapshot snapshot1: snapshot.getChildren()){
                    list1.clear();
                    list1.add(snapshot1.getValue().toString());
                }
                adapter1.notifyDataSetChanged();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        reference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                for (DataSnapshot snapshot: dataSnapshot.getChildren()){
                    list.clear();
                    list.add(snapshot.getValue().toString());
                }
                adapter.notifyDataSetChanged();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });
// When button is clicked it creates a little temporary watermark message
// of kindness
        Toast.makeText(this, "Hello", Toast.LENGTH_LONG).show();
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this, "HELLO HAVE A GOOD DAY",
Toast.LENGTH_SHORT).show();
            }
        });

    }
}
```

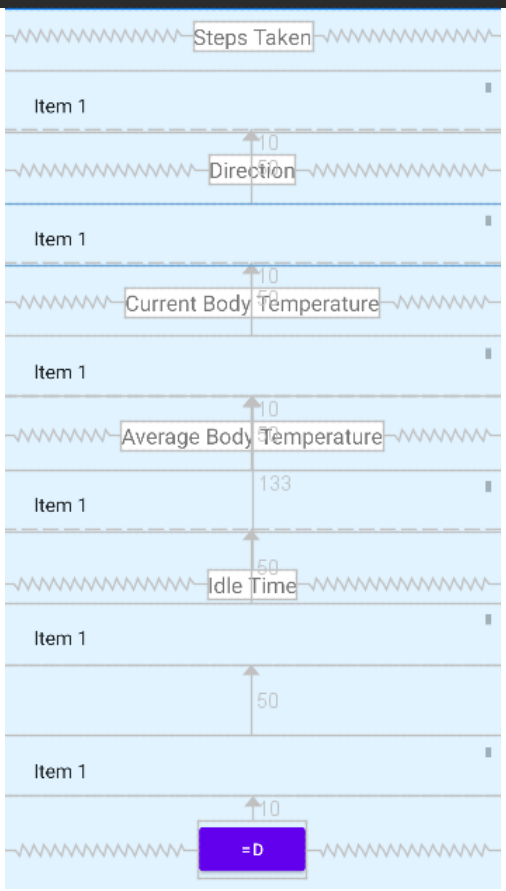
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/material_dynamic_primary95"
    tools:context=".MainActivity">
```

```
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />

    <ListView
        android:id="@+id/listView2"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_below="@+id/listView"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />

    <ListView
        android:id="@+id/listView3"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_below="@+id/listView2"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />

    <ListView
        android:id="@+id/listView4"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_below="@+id/listView3"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />

    <ListView
        android:id="@+id/listView5"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_below="@+id/listView4"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />

    <ListView
        android:id="@+id/listView6"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_below="@+id/listView5"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
```

```
        android:textSize="18dp"

        android:background="@color/cardview_light_background"
        android:text="Steps Taken" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:textSize="18dp"
        android:layout_below="@id/listView"
        android:background="@color/cardview_light_background"
        android:text="Direction" />
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"

        android:layout_below="@id/listView2"
        android:background="@color/cardview_light_background"
        android:text="Current Body Temperature" />


    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/listView3"
        android:layout_marginTop="133dp"
        android:layout_centerHorizontal="true"
        android:background="@color/cardview_light_background"
        android:text="Idle Time"
        android:textSize="18dp" />

    <TextView
        android:id="@+id/textView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/listView3"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"

        android:background="@color/cardview_light_background"
        android:text="Average Body Temperature"
        android:textSize="18dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/listView6"
        android:id="@+id/button"
        android:layout_marginTop="10dp"
        android:layout_centerHorizontal="true"
        android:text="=D" />
```

```
</RelativeLayout>
```



activity_main.xml Design Preview

- We create an additional .xml file in the same location as activity_main.xml.
- This addition file will be named list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/label"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_centerHorizontal="true"
    android:padding="10dp"
    android:textSize="16dp"
    android:textStyle="bold"

    />
```

Evaluation

This section will consist of the tests conducted, testing methods, testing conditions, execution, errors, troubleshooting and outputs.

Testing:

Our testing methodology was chosen as per the requirements of the project. This project needed us to build individual features and then put them together.

Namely the testing methods can be described as **unit testing** and **integrated testing**. First we test a unit for example we test if the board is printing the sensor values we want, then we port this serial data to data sorting & processing algorithm, now we are integration testing. Then our next step is to upload the data to Firebase, so our units become the (board+ algorithm) & (Firebase), after attempting to put them together and testing them out we are back to integration testing again.

Note: The testing schedule is as per the **Gantt chart.**

Obtain Raw Sensor Data:

Test Objective: Code the board to produce raw data in readable formats from the Accelerometer, Temperature sensor, Magnetometer

Type of testing: Unit & Functional

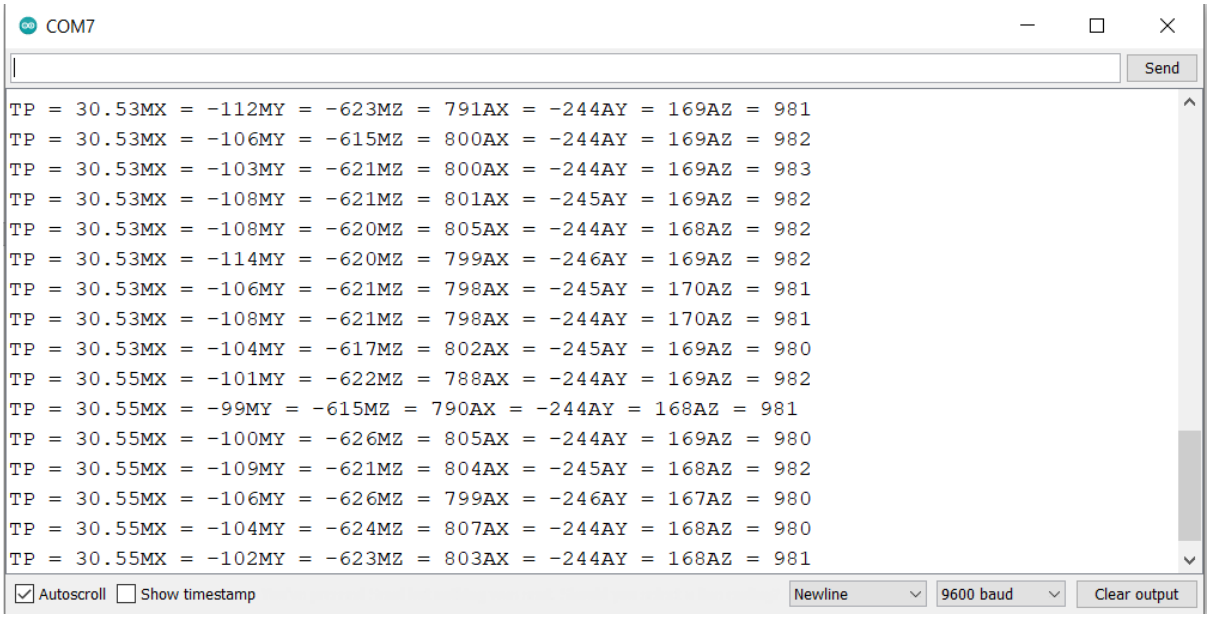
Conditions: The board is connected to the PC and the program for the board is written in mbed.com/compiler online compiler. Board is places away from metal or EM wave emitting obstacles. Arduino IDE Serial Monitor is open and set at 9600 BAUD

Execution: The code below is uploaded to the board and Arduino. This code is a scrapped part of the default sample code available on the online compiler templates.

```
1.  #include "mbed.h"
2.  #include "stm32l475e_iot01_tsensor.h"
3.  #include "stm32l475e_iot01_magneto.h"
4.  #include "stm32l475e_iot01_accelero.h"
5.  DigitalOut led(LED1);
6.  int main()
7.  {   float sensor_value = 0;
8.      int16_t pDataXYZ[3] = {0};
9.      float pGyroDataXYZ[3] = {0};
10.     BSP_TSENSOR_Init();
11.     BSP_MAGNETO_Init();
12.     BSP_ACCELERO_Init();
13.     while(1) {
14.         led = 1;
15.         sensor_value = BSP_TSENSOR_ReadTemp();
16.         printf("\nTP = %.2f", sensor_value);
17.         led = 0;
18.         ThisThread::sleep_for(50);
19.         led = 1;
20.         BSP_MAGNETO_GetXYZ(pDataXYZ);
21.         printf("MX = %d", pDataXYZ[0]);
```

```
22.         printf("MY = %d", pDataXYZ[1]);
23.         printf("MZ = %d", pDataXYZ[2]);
24.         BSP_ACCELERO_AccGetXYZ(pDataXYZ);
25.         printf("AX = %d", pDataXYZ[0]);
26.         printf("AY = %d", pDataXYZ[1]);
27.         printf("AZ = %d", pDataXYZ[2]);
28.         led = 0;
29.         ThisThread::sleep_for(50);
30.
31.     }
32. }
33.
```

Outputs:



Arduino IDE Serial Monitor

Converting Data to Information & Uploading to Database

Test Objective: Be able to put raw sensor data into variables so that it can be used in data processing and updating the initial fields on firebase with the current information.

Type of testing: Integration Testing & Function [Testing Python scripts function and integration with board sensor values from serial]

Conditions: Stable internet connection ensured while board connected to PC and printing raw sensor values. Firebase configurations and SDK applied to Python script. Branches and fields created with sample data (**Pg. 19)**.

Execution: The below python code was executed.

```
1.  import serial # Import pySerial Library
2.  import datetime # Import DateTime library
3.  import time
4.  import pyrebase
5.  ser = serial.Serial("COM7", 9600) #
6.  ser.flushInput()
7.  import time
8.  firebaseConfig ={
9.      "apiKey": "AIzaSyCHmAg5V0CIfGZhJZwViiA2OCXeFdn09GM",
```

```
    "authDomain": "myappfinal-7af0a.firebaseapp.com",
    "databaseURL": "https://myappfinal-7af0a-default-rtdb.firebaseio.com",
  "projectId": "myappfinal-7af0a",
  "storageBucket": "myappfinal-7af0a.appspot.com",
  "messagingSenderId": "460623334634",
  "appId": "1:460623334634:web:6873f67743df7a93be1b93",
  "measurementId": "G-VVDM998KLD"
    }
firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()
list1 = [0]
step =0
stepsfinal = 0
avg_temperature = 0
while True:
    try:
        ser_bytes = ser.readline()
        decoded_bytes = ser_bytes[0:len(ser_bytes)-2].decode("utf-8")
        now = datetime.datetime.now()
        now = now.strftime("%Y-%m-%d %H:%M:%S")
        data = str( "'{}',{}\r\n".format(now,decoded_bytes) )

        #print(data)
        #The Below lines of code find the data
        # and sort the data into integer variables
        # the variables are later used in processing
        index = data.find("MX")
        index2 = data.find("MY")
        index3 = data.find("MZ")
        index4 = data.find("AX")
        index5 = data.find("AY")
        index6 = data.find("AZ")
        index7 = data.find("TP")
        Magneto_X = int((data[index+5:index2]))
        Magneto_Y = int((data[index2+5:index3]))
        Magneto_Z = int((data[index3+5:index4]))
        Accelero_X = int((data[index4+5:index5]))
        Accelero_Y = int((data[index5+5:index6]))
        Accelero_Z = int((data[index6+5:]))
        Temperature = int(round(float((data[index7+5:index]))))
        current_temperature = Temperature
        #print(current_temperature)
        #print(Temperature)
        t = time.localtime()
        current_time = time.strftime("%H:%M:%S", t)
        #print(Magneto_X,Magneto_Y, Magneto_Z,)
        seconds = time.localtime()
        seconds_time = time.strftime("%S",seconds)
        usableseconds = int(seconds_time)
        #print("Usable Seconds: ",usableseconds)

        if (bool(70<Accelero_Z<130)== False)and (bool(-850>Accelero_Y>-1030)==False) and
    (bool(-950>Accelero_X>-1050)==False) and (bool(20>Accelero_Z>-10)==False):
            step = step+1
            steps = step
            stepsfinal = round(steps/3.0)
            print(stepsfinal)
        else: pass
        if 0<Magneto_X and Magneto_X <100 and -5<Magneto_Z and Magneto_Z<25:
            direction = "EAST"
        elif 340<Magneto_X and Magneto_X<500 and -5<Magneto_Z and Magneto_Z<25 and
    Magneto_Y>-400 and Magneto_Y<-150:
            direction = "NORTH"
        elif 700<Magneto_X and Magneto_X<790 and -5<Magneto_Z and Magneto_Z<25:
            direction = "WEST"
        elif 160<Magneto_X and Magneto_X<300 and Magneto_Y<-700 and Magneto_Y>-1000:
            direction = "SOUTH"
        if (usableseconds%17==0):
            print("X")
            db.child('-MyElY-T4gZQ0J7lYaRT').update({"Steps":stepsfinal})
```

```
78.             time.sleep(0.05)
79.             db.child('-Mxy_G9CrlJE6whnGst6').update({"Direction":direction})
80.             time.sleep(0.05)
81.             db.child('-My7SPh8MCJE_6APo3Ex').update({"CurrentBody":Temperature})
82.             time.sleep(0.05)
83.             db.child('-My7SGIw2P5sGSQOcK5O').update({"AvgBody":avg_temperature})
84.             time.sleep(0.05)
85.         else:pass
86.     except:
87.         print("*")
88.
```

Outputs: Data updating on cloud.

Note: This is an older version of the python script so it may not include all the features, this script was made when testing.

Information Display & Update Speed:

Test Objective: Display the information from firebase onto the Android Application

Type of testing: Integration testing

Conditions: Data kept Constantly updating on the database. Stable internet connection.

Execution: Following Code in AcitivityMain.java

```
public class MainActivity extends AppCompatActivity {
    private ListView listView;

    FirebaseDatabase database = FirebaseDatabase.getInstance();
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);

this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        getSupportActionBar().hide();
        setContentView(R.layout.activity_main);
        listView = findViewById(R.id.listView);
        ArrayList<String> list = new ArrayList<>();
```

```
        ArrayAdapter adapter = new ArrayAdapter<String>(this,
R.layout.list_item, list);
        listView.setAdapter(adapter);
        DatabaseReference reference =
FirebaseDatabase.getInstance().getReference().child("-MyElY-
T4gZQ0J7lYaRT");
        reference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                for (DataSnapshot snapshot: dataSnapshot.getChildren()){
                    list.clear();
                    list.add(snapshot.getValue().toString());
                }
                adapter.notifyDataSetChanged();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });
        Toast.makeText(this, "Hello", Toast.LENGTH_LONG).show();
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this, "HELLO HAVE A GOOD DAY",
Toast.LENGTH_SHORT).show();
            }
        });

    }
}
```

activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/material_dynamic_primary95"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_marginTop="50dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp" />
   <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:textSize="18dp"
```

```
        android:background="@color/cardview_light_background"
        android:text="Steps Taken" />



</RelativeLayout>
```

list_item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/label"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_centerHorizontal="true"
    android:padding="10dp"
    android:textSize="16dp"
    android:textStyle="bold"

    />
```

Android Device set to Google Pixel 2 API 32

Errors: 1

Error Description: ‘No Devices Found’ for emulation.

Error Fix: Opening CMD, from the C:\Users\User\AppData\Local\Android\Sdk\platform-tools execute command adb devices

Outputs: On Device Emulator, Steps are being updated as board is physically shaking.

Testing Whole System:

Test Objective: Check if every part of system is working

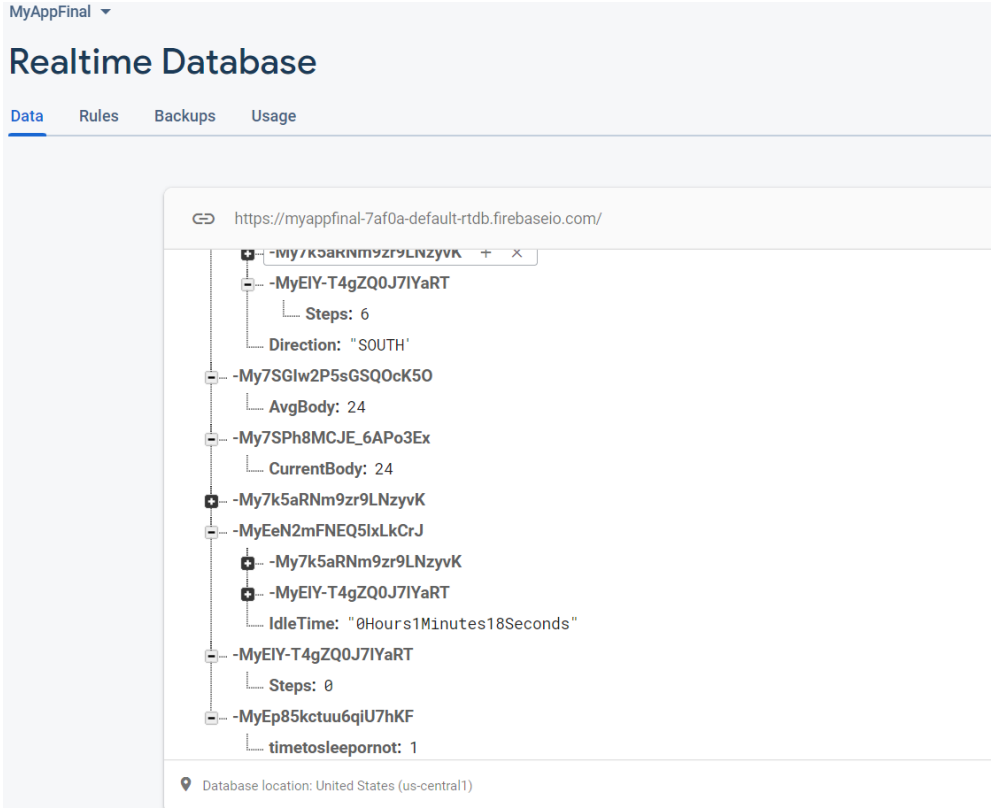
Type of testing: Integration

Conditions: All the conditions set as before.

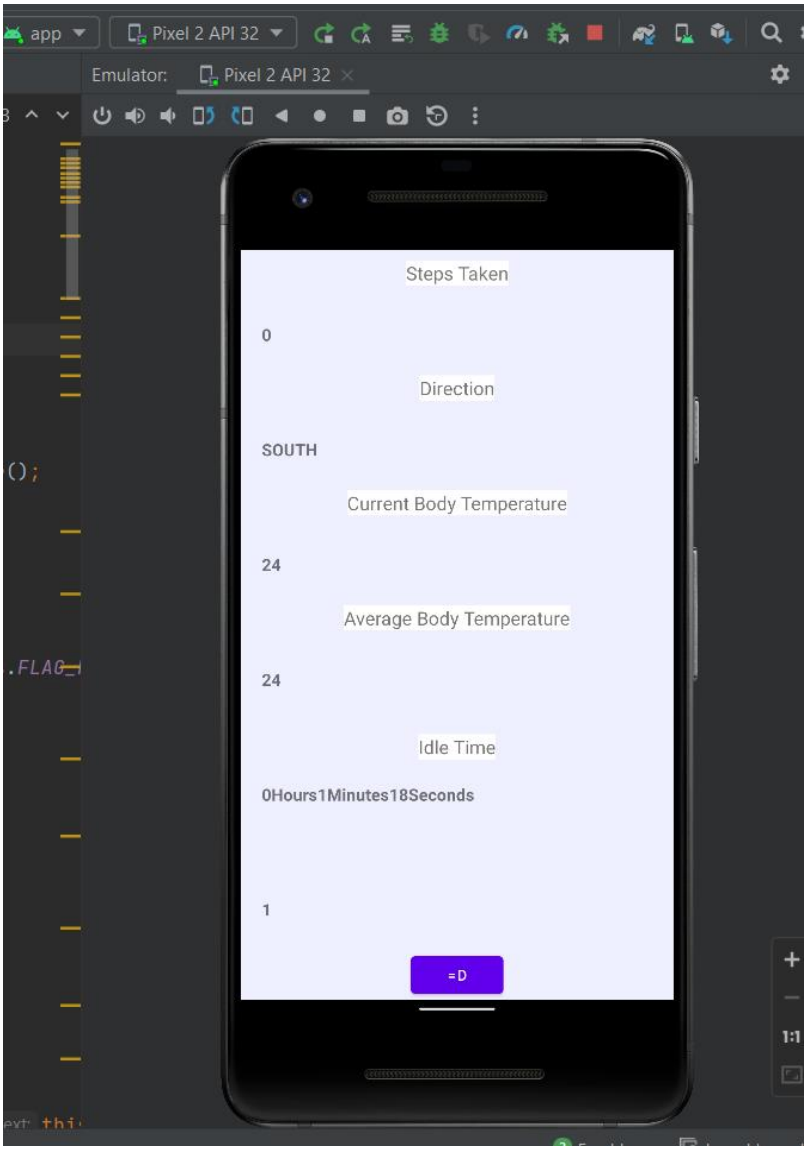
Execution: Connect board, run python script, built app.

Outputs:

Database:



Android App:


app
Pixel 2 API 32
Emulator:
Pixel 2 API 32
Steps Taken
0
Direction
SOUTH
Current Body Temperature
24
Average Body Temperature
24
Idle Time
0Hours1Minutes18Seconds
1
=D

Conclusion

It is evident that not everything is always perfect, however, it is essential to reflect and improve what isn't perfect. This section will talk about our project's flaws and what's missing and might be added in the future.

Limitations:

- The accelerometer [Step Counting] can give inaccurate measures sometimes as only acceleration is not enough for counting steps. Sometimes the effects of gravity or travelling in a car or bike might get counted as steps.
- The magnetometer gives error in readings when there are metal obstacles or EM wave emitting obstacles around it which might lead to an inaccurate compass/direction reading.
- The current apparatus isn't portable as it has to be connected to a PC or the best possible is to connect it to a host that is able to take serial data, run python scripts and communicate with the cloud at the same time.
- The application is not compatible with all versions of Android.

Future Directions:

- Improve User Interface & Aesthetics
- Solve the accelerometer inaccurate readings limitation by using the accelerometer with a gyroscope so that it can detect when the accelerometer is vertical to the ground. When accelerometer is vertical to ground accelerometer acceleration will be ignored as it might be an effect of gravity.
- Detect steps only when a arc type acceleration/movement vector takes place, this way step counting can be a little more accurate.
- Make the device portable by sending data directly from the board to a cloud through MQTT and host the data processing algorithms on the cloud.
- Include a feature that motivates the user to take a walk or stand up when the system detects long idle time in daytime.
- Detect heartbeat using heartbeat sensor.

We have reflected upon the limitations of our current build, and we wish to proceed with fixing these limitations. We engineer a product if it doesn't exist and if it does we look for ways to make it better and solve even more problems.

References

1] Masoudi, F.A., Ponirakis, A., de Lemos, J.A., Jollis, J.G., Kremers, M., Messenger, J.C., Moore, J.W.M., Moussa, I., Oetgen, W.J., Varosy, P.D., Vincent, R.N., Wei, J., Curtis, J.P., Roe, M.T. & Spertus, J.A. 2017, "Trends in U.S. Cardiovascular Care: 2016 Report From 4 ACC National Cardiovascular Data Registries", *Journal of the American College of Cardiology,* vol. 69, no. 11, pp. 1427-1450.

2] Eijsvogels, T.M.H., Molossi, S., Lee, D., Emery, M.S. & Thompson, P.D. 2016, "Exercise at the Extremes: The Amount of Exercise to Reduce Cardiovascular Events", *Journal of the American College of Cardiology,* vol. 67, no. 3, pp. 316-329.

3] Vaes, A.W., Spruit, M.A., Goswami, N., Theunis, J., Franssen, F.M.E. & De Boever, P. 2022, "Analysis of retinal blood vessel diameters in patients with COPD undergoing a pulmonary rehabilitation program", *Microvascular research,* vol. 139, pp. 104238.

4] Chaddha, A., Robinson, E.A., Kline-Rogers, E., Alexandris-Souphis, T. & Rubenfire, M. 2016, "Mental Health and Cardiovascular Disease", *The American Journal of Medicine,* vol. 129, no. 11, pp. 1145-1148.

Appendices

https://drive.google.com/drive/folders/1NZ-TTOImxPUlQAgTcEcvwpctxYe1kO9W?usp=sharing