HEALTH MONITORING DEVICES USING IoT

A PROJECT REPORT

Submitted in partial fulfillment of the
requirement for the award of the
Degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

by

Aditya Harichandar (17BEC1085)
Raya Swethaa (17BEC1176)

Under the Guidance of

Dr. Pradeep Narayanan. S

SCHOOL OF ELECTRONICS ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127

June 2021

CERTIFICATE

This is to certify that the Project Work titled "***HEALTH MONITORING DEVICES USING IoT***" that is being submitted by ***Aditya Harichandar (17BEC1085) and Raya Swethaa (17BEC1176)*** is in partial fulfillment of the requirements for the award of **Bachelor of Technology in Electronics and Communication Engineering**, is a record of bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same iscertified.

Dr. Pradeep Narayanan. S

Guide

The Project Report is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

Head of the Department
B. Tech. (ECE)

DEAN
School of Electronics Engineering

ACKNOWLEDGMENT

We sincerely thank our Dean, **Dr. Sivasubramanian. A**, and Head of Department, **Dr. Vetrivelan. P** for giving us this opportunity, their support, and providing us with the required facilities to complete this project.

We would also like to express our immense gratitude to our project guide, **Dr. Pradeep Narayanan. S**, who guided us all through the project for the past few months and also for his help, suggestions, insights, and directions in the completion of this project work.

We want to express our sincere gratitude, appreciation, and resourcefulness of our friend **Mr. Aman Shaji (17BEC1164)**, who was initially a part of the team but unfortunately had to quit halfway due to unforeseen circumstances, for helping us to develop the Android Application **"Dr. Mask"**, which is a part of this project and in providing us with moral support to complete this project work.

We would also like to thank **our family members** to whom we are greatly indebted for bringing us up to this point and for giving us the moral support and help to work and complete this big project whilst in the middle of a global pandemic.

At last, we would like to thank all the teachers and friends who have always been helping and encouraging us throughout the year and in the satisfactory completion of the project.

This acknowledgment of gratitude provides us with an opportunity to thank all those who have lent us a helping hand. This project has been a product of motivation and encouragement from various sources.

Aditya Harichandar
(17BEC1085)

Raya Swethaa
(17BEC1176)

ABSTRACT

There had been rapid advances in the medical field with the Internet of Things (IoT) technology; still, there are many diseases that cause deaths in younger people. Respiratory diseases are known to be the second most deadly diseases. By proper monitoring and provide necessary help at the right time can prevent such deaths. The process of monitoring will be made easier using this system which doesn't get in the way of the user in their daily life. This also helps to decrease the need for beds as doctors can monitor real-time online without the need of staying close to the patient or user. The proposed model/system aims to monitor the respiratory health of people by observing the change in the respiratory system, heart rate, and blood oxygen level and alert the user to take precautions. This model monitors the breathing/respiration rate, Blood oxygen level (SpO_2), and Beats per minute (Bpm) of the patients all through the day in their daily life to give an early alert to make the user get ready with his medication or do the needed. This data of breathing rate gives us information about the upcoming asthma attack and data of blood oxygen level gives information about the hypoxia problems a person can face. This information can help early detection of the emergency that one can face and can be administered with the medication at the correct time. This system helps user or patient avoid fall due to dizziness caused due to less oxygen reaches brain cells, less oxygen reaching body cells called hypoxia and breathlessness caused by an asthma attack and cross them safely.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Section Number	Abbreviation	Full Form
1	App	Application
2	Bpm	Beats per minute
3	SpO_2	Blood oxygen percentage
4	CAN	Controlled area network
5	EN	Enable
6	GPIO	General- Purpose Input/Output
7	GSM	Global System for Mobile Communication
8	GUI	Graphical User Interface
9	Hb	Hemoglobin
10	HR	Heart Rate
11	IoT	Internet of Things
12	PWM	Pulse Width Modulation
13	RR	Respiratory Rate
14	SoC	System on Chip
15	UART	Universal asynchronous receiver-transmitter
16	URL	Uniform Resource Locator
17	USB	Universal Serial Bus
18	WBSN	Wireless Body Sensor Nodes
19	Wi-Fi	Wireless Fidelity

CHAPTER 1

INTRODUCTION

Internet of Things (IoT) made its debut in the industry; it gained a huge boost in applications over various fields. In medical and healthcare applications IoT has become one of the dominant fields to make the process more automated and intelligent [1 – 4]. It also reduces the need for intermediator or personnel who has to overlook the process. Healthcare monitoring becomes automated now with the help of IoT and even a layman can operate it to monitor their loved ones. Further, it helped doctors to get a digitized analysis of the patient before they make an appointment and prescribe the solution by saving more time.

People may die at any age due to illness; in that respiratory disease become the second most deadly disease across the globe. It may occur accidentally and leads to an increase in death numbers on one particular day. Asthma disease is one among them, where it causes lungs to malfunction due to trachea concentration and leads to breathing shortness or inability to breathe. Hypoxia is another common respiratory disease that is identified when there is less flow of oxygen into the body tissues through the blood. By this, if enough oxygen is not reached to the brain cell one may felt dizzy and it leads to various issues like cardiac arrest, less blood flow...etc., [5 – 8]. If proper care is not taken to these patients at that time, it will end either in death or head injury if they fainted and fall accidentally. An IoT-based system can prevent this which consists of ESP32, Humidity, and Pulse Oximeter Sensors, which collects data of blood oxygen level and Bpm from the wrist and humidity from the near nasal area. It is used to sense the action of breathing IN or OUT as specified in [9]. Further, the captured data can be sent to cloud computing and later to the application (App) developed. This system can save lives and ensured the right medication at the desired time [10 - 12].

Through sensor monitoring and by analyzing the data collected from the person it is easy to alert him/her if the data shows any change in values either in humidity or in blood oxygen level. It will save the person's life by making him/her conscious to use an inhaler or through alarm/notification nearby people can give first aid if he/she fainted, unfortunately. An asthma attack can be prevented and immediate notification through SMS will be sent to the doctor and the caretaker of that person. Through the analytical values, it prevents the accidental falls and alerts the guardian/caretaker to take necessary steps as early [13, 14].

1.1 Objectives

The following are the objectives of this project:

- To develop a system that can be used in the daily life of all conditions like hot weather, rainy conditions or doing workouts, etc.
- The proposed system will have a mask and wrist band combo with utmost protection is taken while using at various ambiances.
- For cloud base, Google Firebase is used to visualize the user/patient condition through the developed app or web page both to the doctor's and guardian and by the user itself.
- Once an abnormality is detected an alarm/notification, SMS, or email will be sent to the doctor and guardian as an alert.

1.2 Background and Literature Survey

A few of the prior works regarding the same field are analyzed to gain insight to aid our research.

Wireless Body Sensor Nodes (WBSN), storing data in public clouds can improve analysis and reduce storage problems, efficient encoding techniques can be used as a viable option [1]. The efficiency of the low-cost wearable device should be the ideal industry device [2]. The scope of the Internet of Things (IoT) in healthcare and the challenges it faced are analyzed [3]. Using "ThingSpeak" as a web user interface allows easy visualization and analysis of the data collected [4]. The usage of IoT net as topology and the importance of accuracy sensors are discussed [5]. The authors proposed a system that can be used at any place that stores the collected data in the cloud which can be viewed by professional health advisers. The system uses temperature, blood pressure, and ECG sensors which are connected to the ARM Microcontroller and LCD. Blood Pressure (BP) values and pulse rate are displayed in LCD and if it exceeds the threshold value an alert buzzer would be triggered. They used a GPRS module to transfer data to the cloud. Cloud computing technology is used to increase the capacity of the shared resource of the data collected in an expeditious and secured way through the internet. It reduces storage time and cost for this model without any need for large servers [6]. Monitoring in far rural areas can be done by LoRaWAN (IOT4HC),

where it's providing a 33 KM coverage area. This method comes with low cost and low power usage and suitable for e-health monitoring compared to Global System for Mobile Communication (GSM) [7]. One's health can be continuously monitored using advanced sensors which can be either embedded into the patient body or worn by them [8].

The Respiratory Rate (RR) monitoring along with the merits and limitations of each method are discussed [9]. The pulse oximetry, activity tracking, pulmonary ventilation, and air quality assessment are the four main important parts in respiratory healthcare which could be measured via smart wearable devices. The merits and demerits of these wearable devices are also discussed [10]. The asthma patients are monitored using the breath rate. Different ways are used to transmit the captured data securely. The extraction part includes the details of the breath rate for different ages. A cloud-assisted IoT framework is developed in this paper for asthma patients. In this system, different sensors and devices are used to record the respiratory rate. Patient's data are collected, recorded, and monitored through cloud technology and big data. Using a cloud service, the transmitted data are restored, enhanced, and encrypted through a network connection. If mismatched in the data shows there is a chance of mental disorders, heart attacks, social embarrassment, or physical damage to the patients. Authentication and security occurred through the signals are embedded with encryption/ watermarking. RR is then calculated using a simple algorithm. The signal sent to the cloud are extracted, analyzed, and sent to the healthcare professional where they will take care of the patient's health [11].

A device to assist the people who are affected by asthma, using the help from the sensors data like dust, humidity, temperature, and barometer will be collected and uploaded to the cloud for further analysis with the concerned doctor and patient's caretaker [12]. Remote monitoring of pulse oximetry with low-energy consumption, a low-cost device through Bluetooth technology used here to monitor blood oxygen partial saturation (SpO_2) and Heart Rate (HR) through open-hardware solutions for data transmission and reception from a cloud server. It uses 'HTTP requests' for clouding [13]. Embedded Systems with IoT provide a solution to the constant and non-invasive measuring of the cardiac values with the help of a technology known as pulse-oximetry. Pulse-oximetry technology uses a medical sensor that creates a photoplethysmogram to detect the oxygen saturation level and changes in blood volume in the tissues. A clip-like pulse oximeter is used to get the SpO_2 level and then it calculates the proportional value using an operational amplifier circuit with SpO_2 reading. With the help of a lookup table, light absorption for different wavelengths is captured, and

computes the heart rate from it. This heart rate is sent to Cloud and displayed in LCD. The cloud checks for irregularities and if yes activates the alert system which uses the GPS and sends SOS or alert message of Google maps link to the patients nearest dear ones [14].

Table 1.2.1 Literature Survey

S. No.	Author Name	Title	Journal & Year of Publishing	Inference
1	Avrajith Ghosh,Arnab Raha and Amitava Mukherjee	Energy-Efficient IoT-Health Monitoring System using Approximate computing.	Jan. 2020 Elsevier, Internet of Things	Wireless body sensor using IoT, storage in public clouds and techniques to reduce data redundancy and improve efficiency.
2	Ahmed Abdelgwad, Ahmed Khattab and Kumar Yelamarthi	IoT- Health Based Monitoring system for Active and Assisted living.	Jul. 2017 International Conference on Smart Objects and Technologies for Social Good	Cloud computing techniques and efficiency with low-cost wearable device architecture.
3	Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci and Silvana Andreescu	Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges.	Jun. 2015 IEEE International Conference on Services Computing At: New York, NY	Scope of IoT in healthcare and the challenges. Usage of hybrid cloudlets, intelligent IoT systems and future IoT analysis.
4	Md. Milon Islam, Ashikur Rahaman and Md. Rashedul Islam	Development of Smart Healthcare Monitoring System in IoT Environment.	May. 2020 SN Computer Science	Smart based IoT healthcare monitoring using 'ThingSpeak" as web user interface
5	D.Shiva Rama Krishnan, Subhash Chand Gupta and Tanupriya Choudhury	An IoT based Patient Health Monitoring System.	Jun. 2018 International Conference on Advances in Computing and Communications Engineering At: Paris, France	Importance and accuracy of LM35, LM358 and double low operational enhancer for better performance.
6	Andrea Aliverti	Wearable technology: role in respiratory health and disease.	Jun. 2017 Breathe (Sheff) At: Milan, Italy	Four main areas of interest for respiratory healthcare via smart wearable devices and its merits and problems.

7	S Lakshmanachari, C. Srihari, A.Sudhakar and Paparao Nalajala	Design and Implementation of Cloud based Patient Health Care Monitoring Systems using IoT.	2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS) At: Hyderabad, India	Mobile sensor used to collect data like noise, location, acceleration, temperature. ARM7DTDMI is used.
8	Afef Mdhaffar, Tarak Chaari, Kaouthar Larbi, Mohamed Jmaiel and Bernd Freisleben	IoT-based Health Monitoring via LoRaWAN.	Jul. 2017 IEEE EUROCON 2017 -17th International Conference on Smart Technologies At: Orchid, Florida	Monitor using *LoRaWAN* (IOT4HC), low cost, low power, secure method, low e-health monitoring compared to GSM.
9	M. Sathya, S. Madhan and K. Jayanthi	Internet of things (IoT) based health monitoring system and challenges.	Feb. 2018 International Journal of Engineering and Technology	Advanced sensor usage worn or embedded form. Challenges on implementation.
10	Haipeng Liu, John Allen, Dingchang Zheng and Fei Chen	Recent development of respiratory rate measurement technologies.	Aug. 2019 Physiol Meas.,	RR monitoring along with the merits and limitations. Future RR monitoring methodologies advantage and limitations.
11	Syed Tauhid Ullah Shah, Fiazan Badshah, Faheem Dad, Nouman Amin and Mian Ahmad	Cloud-Assisted IoT-Based Smart Respiratory Monitoring System for Asthma Patients.	Nov. 2018 Applications of Intelligent Technologies in Healthcare	Asthma monitoring using breath rate, ways to protect, transmit and extract.
12	S. Mohanraj and K. Sakthisudhan	An Internet of Things based Smart Wearable System for Asthma Patients.	Mar. 2019 International Journal of Recent Technology and Engineering (IJRTE)	Assist asthma patients using sensors like dust, temperature, humidity and barometer and data transferred to cloud.
13	Leonardo Juan Ramirez Lopez, Gabriel Puerta Aponte and Arturo Rodriguez Garcia	Internet of Things Applied in Healthcare Based on Open Hardware with Low-Energy Consumption.	Jul. 2019 Healthc Inform Res	Remote monitoring using pulse oximeter, low-energy *Bluetooth* device, operated via 'HTTP requests' from a cloud server.
14	Dhanurdhar Murali, Deepthi R Rao, Swathi R Rao and Prof. Ananda M	Pulse Oximetry and IOT based Cardiac Monitoring Integrated Alert System.	2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI) At: Bengaluru, Karnataka, India	Continuous or non-invasive measuring of cardiac values using pulse oximeter and photoplethsmogram.

1.3 System Overview

The proposed system is feasible to use in everyday life by all ages people since it incorporates the use of a face mask which has become normal nowadays and along with the wrist band for a long duration. Additional sensors are used in the system proposed to monitor the user throughout their day in all conditions like summer, winter, and during regular workouts. Heat resistant material is used to cover the components used in the system to avoid heat damage on the skin. Hydrophobic acoustic mesh is used to allow airflow alone and protect it from dust and water. It can be used during their routine life without finding any convenience to the users and decreasing the risks mostly faced by asthma and hypoxia patients. Once all the data are captured through the sensor will send it for further analysis through the cloud. The graphical representation analyzed from the cloud can be visualized through the app on their smartphone or through the webpage. Analyzed data found it to be an emergency case immediate notification/alarm will be notified to the users through an alert to take immediate medication and a flashlight on-screen to take the help from the near ones. Immediate SMS or email is sent to the guardian and doctor to understand the situation of the user to take necessary steps to protect them as soon as possible. This will avoid accidental deaths to the user through the faster operation with help of the app developed.

1.4 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the block diagram, flow chart, and hardware component explanation in detail along with the comparison and selection tables.
- Chapter 3 explains the design methodology of the circuit connection which is later converted to the wearable device. It also gives an idea of how to set up Google firebase.
- Chapter 4 gives the cost involved in the implementation of the project and analyzes its economic feasibility.
- Chapter 5 explains the Graphical user interface of the developed application and compiles the results obtained after the proposed system is implemented.
- Chapter 6 concludes the work with discussions about the results obtained and their future implications.

CHAPTER 2

PROPOSED MODEL

This chapter describes the proposed model in detail from the block diagram explanation to flow chat how each part of the model functions. It also includes the details of each hardware component used with its technical standards, pin details, and other technical specifications. The explanation of the preferred/used feature of each component, comparison, and choice are given clearly, which gives a clear understanding of the idea behind the proposed model.

2.1 Block Diagram

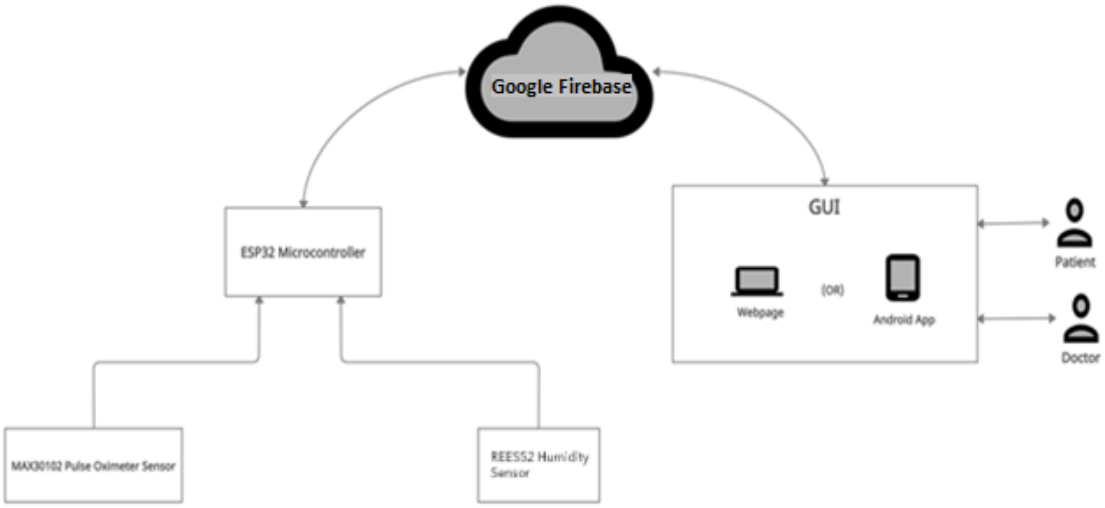


Figure 2.1.1 Schematic Diagram of the Proposed System

Figure 2.1.1 shows the basic block diagram of the proposed model, in which there are 3 layers; the lower layer is the MAX30102 a pulse oximeter sensor and REES52 a humidity sensor, the second layer is ESP32 Microcontroller and the Graphical User Interface (GUI), Application (App) or Website and the top layer is the Google Firebase Cloud.

MAX30102 sensor module gets the Bpm and the pulse oxygen level and the humidity sensor gets the humidity values from the breath. MAX30102 sends data to the Microcontroller using the I2C mode of communication and similarly, REES52 sends data to ESP32. ESP32 microcontroller is connected to the Google Firebase Cloud and sends the data payload in real-time using the MQTT protocol. The data can be visualized in a graphical format in the Google

Firebase Cloud and also be visualized via a webpage or an app that is connected with it. The alerts are sent to the user, doctor, and attender in case if there are any abnormal values or patterns in the Bpm, SpO_2 level, or breathing rate values.

2.2 Flow Chart

Figure 2.2.1 shows the detailed flow chart of the proposed model. In the first layer, sensors are receiving values from the user body, and the received data is sent to the ESP32 microcontroller as per the block in the second layer. In the next layer, data obtained from ESP32 is sent to Google Firebase Cloud and from Google Firebase data is sent to the app. In the app a graphical representation is generated, which can be visualized. Now the data will be analyzed and checked, whether the humidity value is more than the threshold value, or SpO_2 level is more than the threshold value, or Bpm is abnormal to very low or higher than the threshold range. If all the conditions are NO, continuous tracking on the values will be checked. If it finds it to be YES, through any of the conditions an alarm is triggered to make the user take their medication or to rest at the near place. The alarm will notify the nearest people to do the first aid to the user if the condition is so worst, which all three categories go down. Immediate data transfer will be sent to the user's doctor and attender through SMS or email with high priority notification. The alarm can be deactivated by the user if they have taken necessary medicine and back to normal. The system will start checking the data again for any abnormalities. This process continues till the use wears the devices.

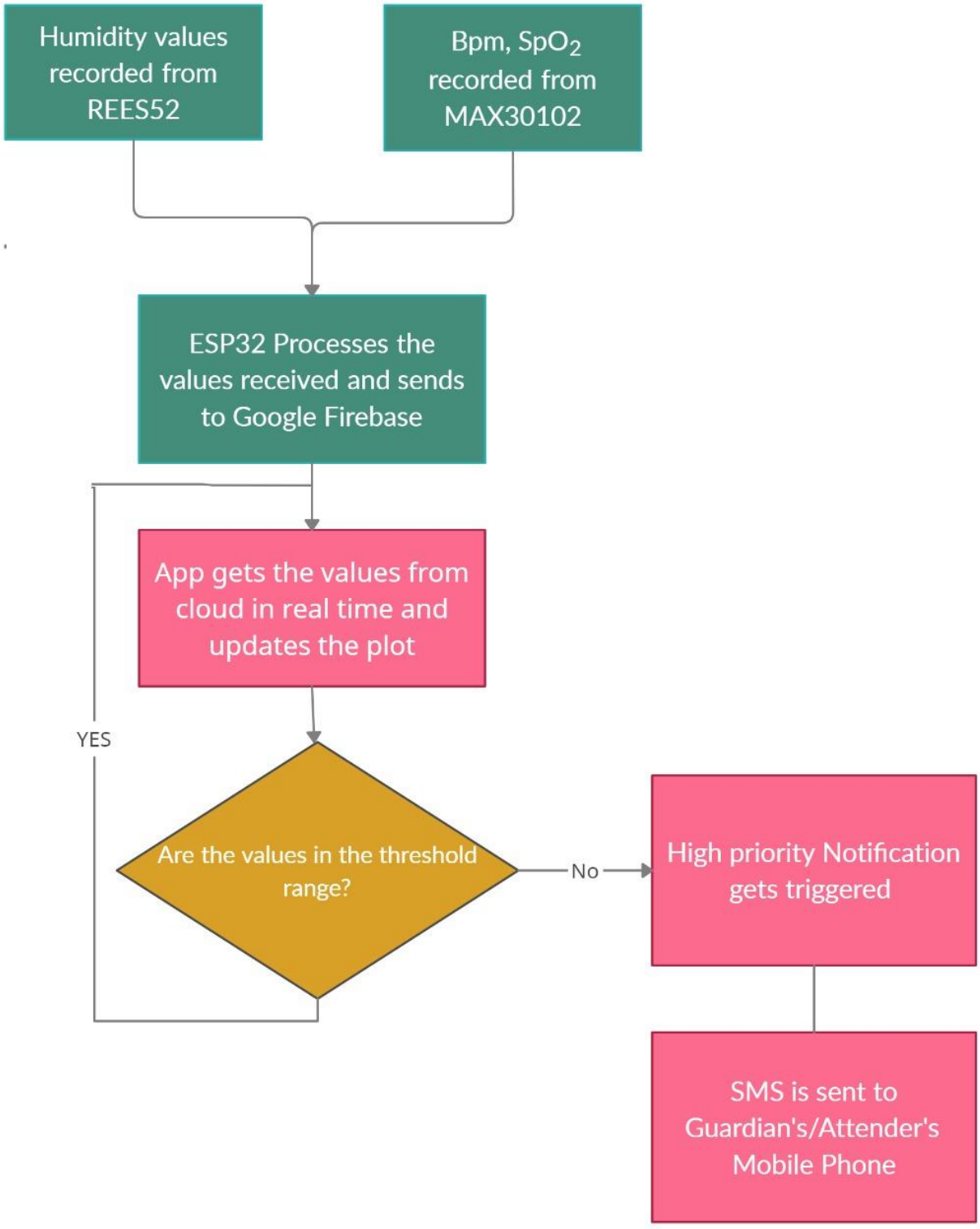


Figure 2.2.1 Schematic Flow Chart of the Proposed System

2.3 Hardware Components

2.3.1 ESP32 – Microcontroller

Figure 2.3.1.1 ESP32 Microcontroller

Figure 2.3.1.1 shows the ESP32 microcontroller which is a System on a Chip (SoC) series with Wireless Fidelity (Wi-Fi), dual-mode Bluetooth. It has a clock rate of up to 240 MHz. It has an inbuilt power amplifier, low-noise receive amplifier, filters, and power management modules. It is low cost and ultra-low power consumption, which requires supply voltage between 2.2 V to 3.6 V.

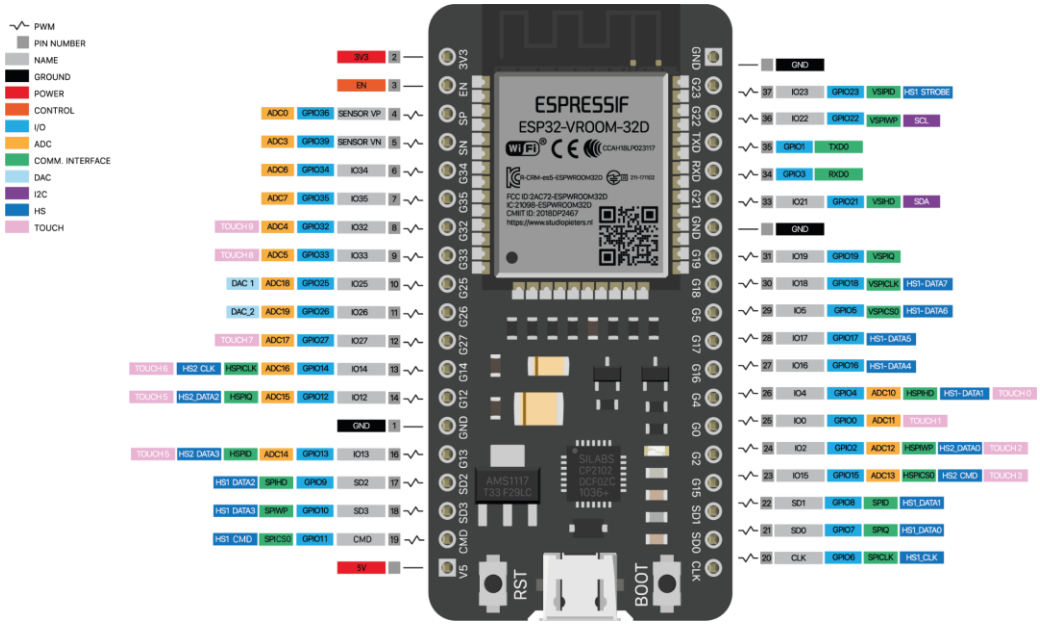


Figure 2.3.1.2 Pin Diagram of ESP32

It has 39 digital pins: 34 pins are General Purpose Input / Output (GPIO) and the rest 5 pins are only input pins. It has Pulse Width Modulation (PWM) signal generation through 16 channels, as per the color-coding is shown in figure 2.3.1.2. It has 3 SPI, 3 Universal Asynchronous Receiver-Transmitter (UART) (3 sets total of 6 pins of Rx and Tx, any of the 34 GPIO pins can be programmed and used as UART), 2 - I2C, 2 - I2S interfaces support Controller Area Network (CAN) protocol. It has a reset pin named Enable (EN) making it low, resets the ESP32. The program is uploaded through the B-type Universal Serial Bus (USB) connection and a power supply can be supplied through this pin. USB supplies 5V to ESP32, which has an inbuilt voltage regulator to convert that supply 5 V to 3.3 V. In table 2.3.1.1 pin details are shown.

Table 2.3.1.1 Pin Details of ESP32

Category	Name	Details
Power	Micro-USB, 3.3V, 5V, GND	USB or regulated 3.3 V or 5 V can be used to supply power to ESP32 as it has a board regulator to convert to 3.3 V and ground pins.
Enable	EN	Button to reset the ESP32.
Analog Pins	ADC1_0-ADC1-5 & ADC2_0-ADC2_9	12-bit, 18 channels ADC; Analog voltage of 0 - 3.3 V is measured.
DAC pins	DAC1 & DAC2	8 bit, 2 channels, Digital to Analog convertor.
I/O pins	GPIO0-GPIO39	Pin 0 to 33 can be used as I/O pins and the rest are input-only pins.
Capacitive touch pins	T0-T9	10 pins are touch pins /capacitive pads.
RTC GPIO pins	RTCIO0-RTCIO17	18 GPIO pins to wake up from deep sleep mode.
Serial	Rx, Tx	Receive or Transmit TTL serial data.

External Interrupts	All GPIO	Used to trigger the interrupt.
PWM	All GPIO	16 independent channels; all GPIO can be used for PWM through Software.
VSPI	GPIO23 (MOSI), GPIO19(MISO), GPIO18(CLK) & GPIO5 (CS)	SPI-1 communication.
HSPI	GPIO13 (MOSI), GPIO12(MISO), GPIO14(CLK) & GPIO15 (CS)	SPI-2 communication.
I2C	GPIO21(SDA), GPIO22(SCL)	I2C protocol communication.
AREF	AREF	Reference voltage for input voltage.

Table 2.3.1.2 Technical Specification of ESP32

Max operating Frequency	240 MHz
Operating Voltage	3.3 V
DC current on I/O pins	40 mA
DC current on 3.3V pin	50 mA
SRAM	520 KB
Communication	SPI (4), I2C (2), I2S (2), CAN, UART (3)
Wi-Fi	802.11 b/g/n
Bluetooth	V4.2- supports BLE and classic Bluetooth.

2.3.2 REES52 – Humidity Sensor

Figure 2.3.2.1 REES52 Humidity Sensor

Figure 2.3.2.1 shows REES52 is a low-cost humidity and temperature sensor, which gives digital output and also provides high reliability and long-term stability. It has 3 pins Vcc, GND, and Output as shown in figure 2.3.2.2, which detects the ambient humidity. It requires a 3.3 V to 5 V supply and the measurement range of humidity is 20-90% RH and temperature is 0~50℃. It uses a capacitive humidity sensor and thermistor to measure values.

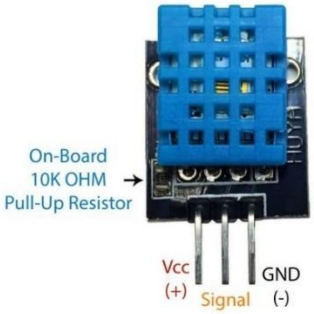


Figure 2.3.2.2 Pin Diagram of REES52

The pin details are as in the table 2.3.2.1 given.

Table 2.3.2.1 Pin Details of REES52

Pin	Description
Vcc	3.3 V - 5.5 V DC or +ve
Output	Serial Output (Digital)
GND	Ground or –ve

The working principle of the humidity sensor is that it uses capacitance to measure the humidity in the air. It has two electrical conductors with a non-conductive polymer film laid between them for the creation of electric fields between the plates. Moisture from the air gets collected on the polymer film and this causes a voltage level change between the plates. This change in voltage is converted into a digital value of relative humidity taking air temperature into account. When the moisture is absorbed by the polymer the ions released in the film increase the conductivity between the electrodes this causes a change in resistance between electrodes, which in turn changes the voltage between them.

2.3.3 MAX30102 – Pulse Oximeter

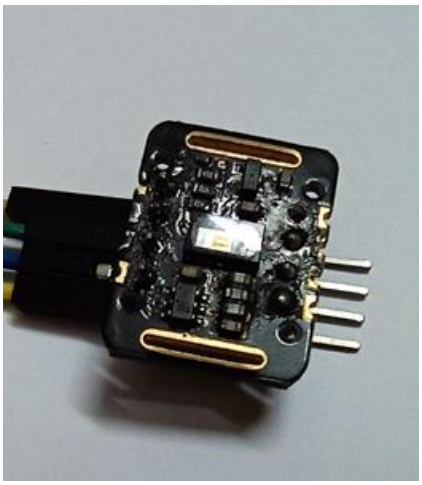


Figure 2.3.3.1 MAX30102 Pulse Oximeter

Figure 2.3.3.1 shows MAX30102 which is an ultra-low-power operation biosensor module, integrated with LEDs, low noise electronics with ambient light rejection, photodetectors, and optical elements. It requires a 3.3 V to 5 V supply and communication is done through I2C protocol. The pin diagram is shown in figure 2.3.3.2.

It is used to measure the heartbeats and oxygen levels in the blood. The sensor consists of 2 LEDs, one emits red light and the other emits Infra-Red (IR) light and a photodetector. The red LED wavelength is 650 nm and an IR LED is approximately in 950 nm wavelength. A photodetector measures the light intensity and the difference will tell the blood oxygen level of the user.

Figure 2.3.3.2 Pin Diagram of MAX30102

The pin details and specifications of MAX30102 are given in table 2.3.3.1 and table 2.3.3.2, respectively.

Table 2.3.3.1 Pin Description of MAX30102

Pin	Description
Vin, GND (2)	Power supply of 3.3 V – 5 V and has 2 ground pins.
SCL	I2C communication clock input.
SDA	I2C communication Data output.
INT	Active low interrupt; connected to external pull up resistor.

Table 2.3.3.2 Technical Specification of MAX30102

LED peak Wavelength	660 nm/880 nm
LED supply voltage	3.3 V – 5 V
Communication	I2C
Operating temperature	-40 °C to +85°C
Dimension	20.3 x 15.2 mm
Weight	1.1 g

The sensor used in the system works on the principle that oxygenated blood absorbs more IR light and allows more red light to pass, whereas deoxygenated blood absorbs more red light and passes IR light.

The sensor uses a reflective method of measurement for heart rate; the LED and photodetector are on the same side next to each other, so, when the light gets emitted, the sensor will reflect some fixed lights absorbed from the body part. If oxygenated blood

increased in the heart beat sensor absorbs more IR light and reflects red light. The observed light will be more, which is captured through the sensor and send back to the system. When the heartbeat doesn't work properly that time the deoxygenated blood becomes high. It makes more red light to be absorbed by the sensor and reflects less IR light. Once the data is analyzed in the graphical representation, the graph shows the peaks for observation of more red light during abnormal conditions. The difference between the normal spike and observed spike will help in the measurement of Beats per Minute (Bpm) or Heart Rate (HR) of the user.

For SpO_2 measurement by observing the Oxygenated Hemoglobin through both lights, one can easily find that oxygenated heartbeat absorbs more in 900 nm range light which falls to IR category and less in 600 nm range, which is for a red light. Similarly, deoxygenated heartbeat absorbs vice versa. So, by comparing the red light and IR light absorbed by oxygenated heartbeat, it is easy to calculate the amount of oxygen content in the blood for any person. The amount of oxygenated heartbeat and deoxygenated heartbeat will change the ratio of IR to Red. Using the lookup table, the ratio for the SpO_2 level can be calculated. In general, if the ratio is 0.5 means 100% SpO_2, 1.0 approximates to 82% SpO_2, and 2.0 means 0% SpO_2.

2.3.4 Lithium Polymer Rechargeable Battery

Figure 2.3.4.1 Li-Po Battery

In this work lithium polymer 3.7 V, 300 mAh rechargeable battery also called Li-Po battery is used. It is a thin, lightweight, and powerful battery that makes them easier to be used even in mobile systems. The approximate size will be 25 × 20 × 4 mm. It has a GND and Vcc to connect to the system, reversing the connection recharges the battery, and sometimes it short the battery too.

2.3.5 Tables of voltage selection

Table 2.3.5.1 and 2.3.5.2 explains the process of voltage selection of each component and battery selection for the proposed model.

Table 2.3.5.1 Voltage Selection for Components

Title	Vcc (Range)	Vcc (Used)	Inference
ESP32	2.2 V - 9 V	3.7 V	REES52 and MAX30102 are powered through the ESP32 microcontroller and they require a minimum of 3.3 V. It has a board regulator to change any voltage to 3.3V to power the board.
REES52 (Humidity Sensor)	3.3 V - 5V	3.3 V	ESP32 microcontroller is supplied with 3.7 V and has a 3.3 V output pin. Hence, we decided on the minimum value of 3.3 V.
MAX30102 (Pulse Oximeter)	3.3 V – 5 V	3.3 V	ESP32 microcontroller is supplied with 3.7 V and has a 3.3 V output pin. Hence, we decided on the minimum value of 3.3 V.

Table 2.3.5.2 Battery Selection for the Proposed Model

Name	Voltage	Inference
Hi-Watt battery	9 V	This battery is chosen initially as its commonly available and relatively cheap. But due to the high voltage, there would be heating issues after prolonged usage, which interfered in the functioning of the sensors. Also due to the bulky in size and weight, it was not preferred to use in our system.
NI-MH battery	3.6 V	This battery is also suitable, but it has a lesser voltage output due to its bulkiness, it wasn't an ideal fit for the design our system required.
Li-Po battery	3.7 V	This battery has less voltage and is smaller in size by a huge margin compared to the previously listed ones and is also a rechargeable type, which is best to fit into the proposed system.

2.4 Summary

Thus, the block diagram, flow chart, and detailed explanation of the components used along with its technical specifications and comparison are discussed in this chapter. The design methodology of the proposed model is presented in Chapter 3.

CHAPTER 3

DESIGN METHODOLOGY

This chapter describes the circuit connection on the breadboard which is later implemented as a wearable device, Firebase cloud setup, and design of the wearable proposed system. It also discusses the constraints, alternatives, and tradeoffs for the proposed system.

3.1 Circuit Connection

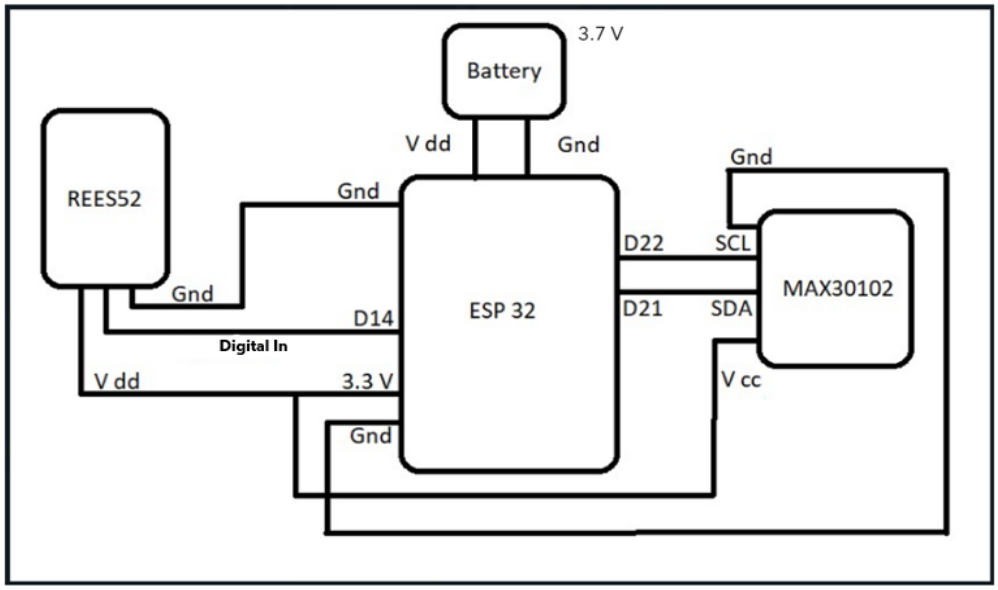


Figure 3.1.1 Circuit Diagram of the Proposed System

Figure 2.4.1 shows the circuit connections of the model. It has 4 components and 3 different connections.

a) REES52 to ESP32

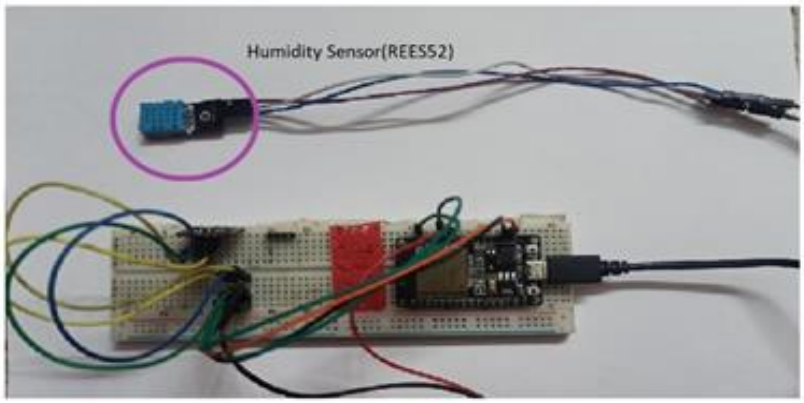
REES52 has 3 pins Vcc, GND, and output pin. Vcc is connected to a 3.3 V output supply pin of ESP32 and the GND pin of the sensor is connected to the GND of the microcontroller to power the humidity sensor from the ESP32. The output pin of REES52 gives a digital output so it is connected to the D14 pin of ESP32, which has a DAC (an inbuilt digital to analog converter) to convert the obtained data into its analog value.

b) MAX30102 to ESP32

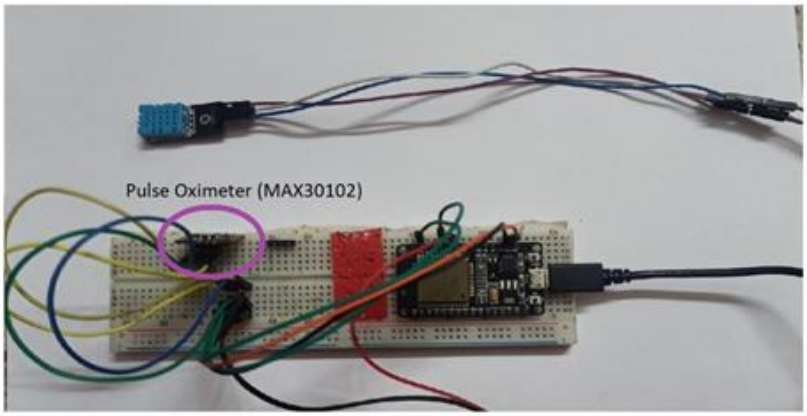
MAX30102 uses the I2C protocol, so, it has 4 pins namely serial clock (SCL), serial data (SDA), Vcc, and GND pin. Vcc and GND pins of MAX30102 have connected parallelly to the 3.3 V pin and GND pin of ESP32, respectively, which will supply the voltage to the pulse oximeter sensor. Similarly, the SDA and SCL of MAX30102 are connected to the SDA and SCL pins of ESP32 which are D21 and D22 pins, respectively. The output of MAX30102 is analog value.

c) Li-Po battery to ESP32

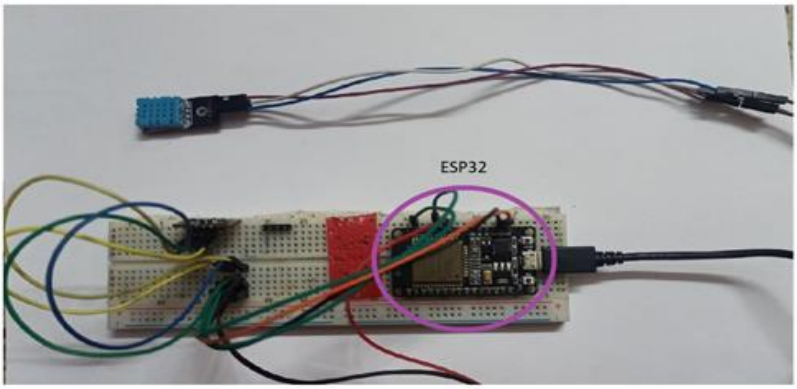
Li-Po battery is used to supply voltage to ESP32 by connecting Vcc and GND pins, respectively. The data received is sent to the cloud using the Wi-Fi module available in the ESP32. As the microcontroller, it is connected directly to the Google Firebase Cloud. Figure 3.1.2 shows the circuit connection of the proposed system.



(a)



(b)



(c)

Figure 3.1.2 (a) Humidity Sensor, (b) Pulse Oximeter Sensor, and (c) ESP32 connected in the Proposed System.

3.2 Setting up, connecting to Google Firebase and Coding for the system functioning

Google Firebase is Google's database platform which is chosen as a cloud source for this project. It is used to store, retrieve and transfer data received from any sources like sensors, android app, or web services. Google Firebase can be developed either as a mobile-based application or web development platform, in this many features are included like cloud messaging and authentication, real-time database. A Real-time database serves the captured data lively to store, analyze and transfer through the internet whenever it retrieved. MQTT protocol is used here to send and receive data from connecting sensors or devices. The database provides sorting, queuing, and multimedia support to the received data. Other features like file storage, messaging, server-side functions, and machine learning are also possible using this cloud-based system. 10 GB of free data will be provided for storage. but only 1 GB of transmission data per day is allowed. ESP32 microcontroller is connected to the user cloud account to send the real-time data and graphical representation for easy visualization and a better understanding to the user, doctor, and attender. Below are the steps to create and connect the firebase cloud with the ESP32 microcontroller. The user must have their account connected with this cloud to serve hassle-free service, whenever the proposed system is used.

1. Open the browser page "https://firebase.google.com". Press 'go to console' and create a new project by giving it a name.
2. Enable the Google analytics for the project if not enabled already. Click my

site and create a project button, now the project is created.

3. Go to project setting-> service accounts-> database secrets and get the secret key which is the firebase authorization key as shown in figure 3.2.1.

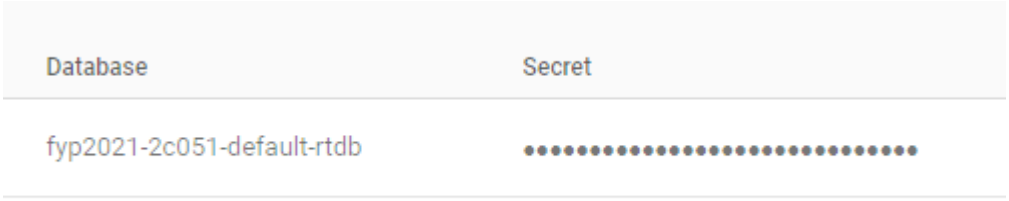


Figure 3.2.1 Secret key

4. Now we need to create the database, Go to Database and create a database by choosing START in Test mode and enable the same.
5. Go to project overview and go to service accounts copy database URL (Uniform Resource Locator) as shown in figure 3.2.2, which is used in the code for connecting the cloud.

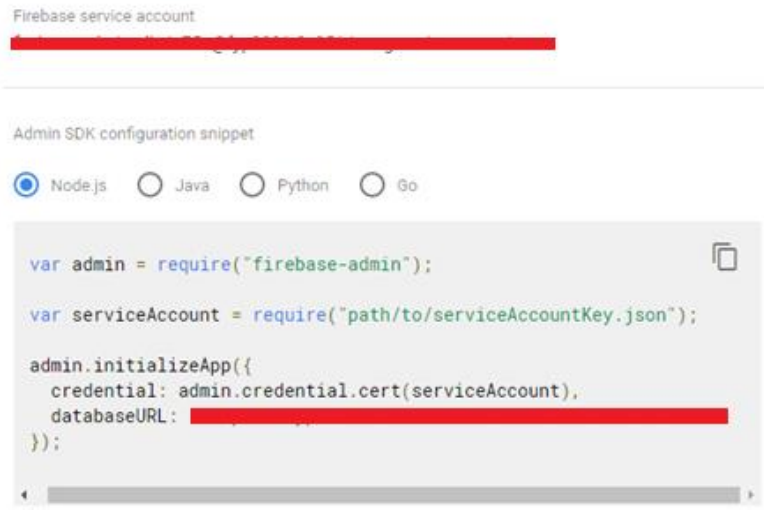


Figure 3.2.2 Firebase Host URL and Service Account Details

6. Download the firebase library for Arduino IDE while using it in ESP32 and include the same in the code as per figure 3.2.3 (a).

```
#include <FirebaseESP32.h>
#include <FirebaseESP32HTTPClient.h>
#include <FirebaseJson.h>

#include "WiFi.h"
```

(a)

```
#define FIREBASE_HOST "[redacted]
#define FIREBASE_AUTH "[redacted]
#define WIFI_SSID "[redacted]"
#define WIFI_PASSWORD "[redacted]"
```

(b)

Figure 3.2.3 (a) & (b) are Codes for connecting ESP32 and Firebase Cloud

7. In figure 3.2.2 (b) the same code is used to connect ESP32 and Firebase by providing the necessary details of the firebase host and secret key.
8. Later the rest of the codes are filled forESP32 functioning with the system, the whole code for hardware system functioning is explained and given in chapter 7 Appendix.
9. In the database, the project created in firebase will show the data uploaded in real-time as shown in figure 3.2.4.

Figure 3.2.4 Cloud Database

3.3 Hardware System Design

The proposed model can be used in our day-to-day life of all age groups. In this section, the model sketch and final product formation are shown and explained clearly in detail.

3.3.1 Proposed Model – Sketch

The whole designed system connected in the Face mask and Wrist band as in sketch

format is shown in figure 3.3.1.1 - 3.3.1.3.

Figure 3.3.1.1 Mask with the REES52

The face mask is used to get the humidity values from the breathing of the user to observe the breathing rate. The humidity sensor is connected with the mask and wires are connected down to the neck and run and via an arm to reach the wrist band as shown in the 3D model representation in figure 3.3.1.4.

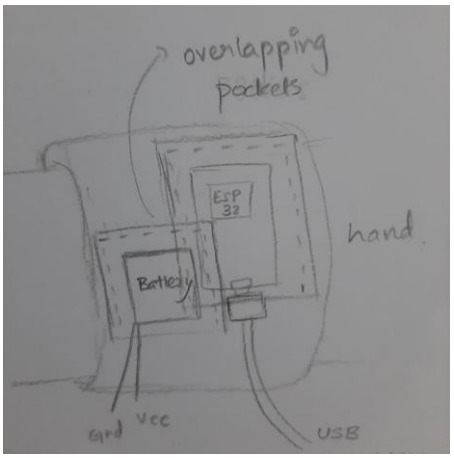


Figure 3.3.1.2 Outer View of the Wrist band

At the outside of the wristband, ESP32 microcontroller and Li-Po battery are placed. Inside the wristband, a Pulse oximeter is placed. To avoid further damage due to water (rain or sweat), components are covered with a layer of Hydrophobic Acoustic mesh, which allows the only airflow and protects dirt and water. Two overlapping pockets are made at the outside

of the wristband to place the ESP32 and battery as shown in figure 3.3.1.2. One pocket with a hole is made on the inside to hold the Pulse oximeter which is shown in figure 3.3.1.3. A hole is created in the presence of an additional layer between the skin and LED used in the Pulse oximeter, which may reduce the system accuracy. To prevent heat damage over the skin, pockets having a layer made up of heat-resistant cloth. In figure 3.3.1.2, a USB connection is made to supply the system and to upload the code, also, it can be used for temporary purposes.

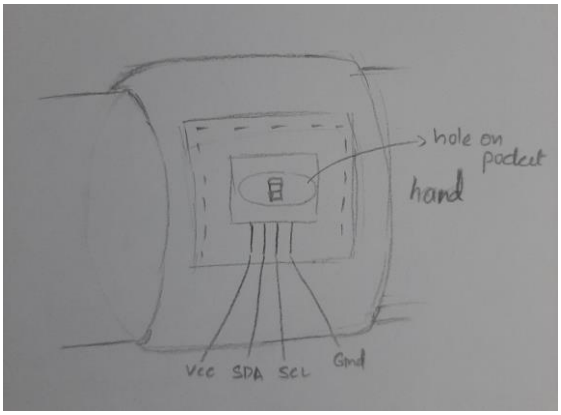


Figure 3.3.1.3 Inner View of the Wrist Band

3D model representation is made by using 3D Paint software as shown in figure 3.3.1.4. It explicates how ESP32 is connected with the Humidity or REES52 sensor in the mask and wire connection through the neck and arm. ESP32, Battery, and MAX30102 are kept properly in the wristband with utmost care.

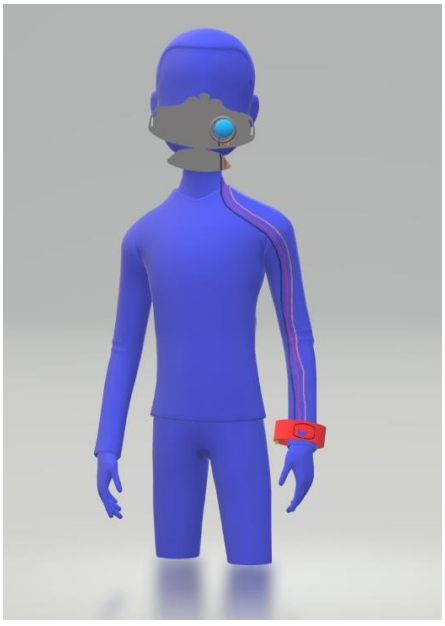


Figure 3.3.1.4 3D Model of the Proposed System

3.3.2 Product Implementation

The sketches from section 3.3.1 that gets implemented and made into a product are shown in this section.

Figure 3.3.2.1 Humidity Sensor on the Mask

In figure 3.3.1.1 the mask is attached with REES52, the sensor is placed at different places inside the mask. Due to less airflow inside the mask, the change in humidity is not significant between breaths. This is because the air keeps circulating inside the mask and even if it moves out the humidity won't change much. So, to solve this problem a sensor is placed outside the mask and is placed at different places. With this, only the proper values are absorbed if we placed the sensor below the nostrils. After placing it outside the mask just below the nostrils, the breath observations are made and shown in figure 3.3.2.1. It helps to get the humidity from the breath and due to proper airflow, a significant difference is observed in the humidity values.

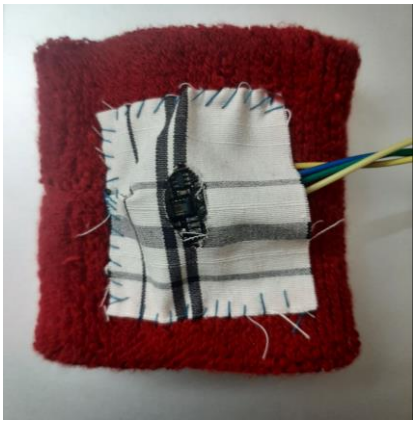


Figure 3.3.2.2 MAX30102 Placed Inside Wrist Band

MAX30102 is placed in the pocket with a hole on the inside of the wristband. It is covered with hydrophobic acoustic mesh and pocket layered with heat-resistant material as explained in figure 3.3.1.2. Similarly, overlapping pockets are also layered with heat-resistant material and covered with hydrophobic acoustic mesh and these pockets hold the ESP32 and Li-Po battery. Figures 3.3.2.2 and 3.3.2.3 replicates the sketches used in section 3.3.1 for the final product implemented.

Figure 3.3.2.3 ESP32 and Battery Placed Outside Wrist Band

The proposed system is shown in figure 3.3.1.4.3D model replication for the user is shown in figure 3.3.2.4, which can be worn by anyone of any age in day-to-day life.

Figure 3.3.2.4 Proposed System used by Different People

3.4 Constraints, Alternatives and Tradeoffs

1. **Constraint:** In the IBM Watson cloud there is a delay of 1 to 2 minutes was found, which makes the system less efficient, and also it provided only 1 GB storage for free usage.

   Alternative: Using Google Firebase as a cloud is observed less delay comparatively, which makes the system more efficient and it provides up to 10 GB of storage for free usage.

   Trade off: Even though the delay time was reduced by using Google Firebase, the main tradeoff is it cannot create graphs in the Firebase cloud, and only up to 1 GB of data can be transmitted to the application by the users per day.

2. **Constraint:** Sensors and Microcontroller may get damaged due to sweat and rain, which makes them inadequate to use during summer or rainy days and performing workouts. It also has close contact with the skin, which may cause heat damage to the user due to the heat generated from the components used in the system.

   Alternative: To prevent heat damage and water entry into components it can be covered with a heat resistant cloth-like high silica glass fiber used in industries and a hydrophobic acoustic mesh which allows airflow but prevents water and dust.

   Trade off: It is hard to procure hydrophobic acoustic mesh in India.

3.5 Summary

In this chapter, the implementation of the circuit from breadboard connection to system components, testing the connectivity with IoT platform using Google Firebase Cloud is discussed. Detailed explanations of the proposed design as IoT based wearable device are also discussed. In the next chapter, a cost analysis of various components of this project is given.

CHAPTER 4

COST ANALYSIS

4.1 List of components and their cost

The cost analysis of the various components used for this project is given in Table 4.1.

Table 4.1.1 List of Components and their Cost

COMPONENT	COST
ESP32 (Microcontroller)	Rs. 560
MAX30102 (Pulse Oximeter)	Rs. 450
REES52 (Humidity Sensor)	Rs. 120
Mask	Rs. 50
Li-Po Battery	Rs. 150
Wristband	Rs. 30
Miscellaneous (wires etc.,)	Rs. 20
TOTAL	Rs. 1,380/-

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Mobile Application UI

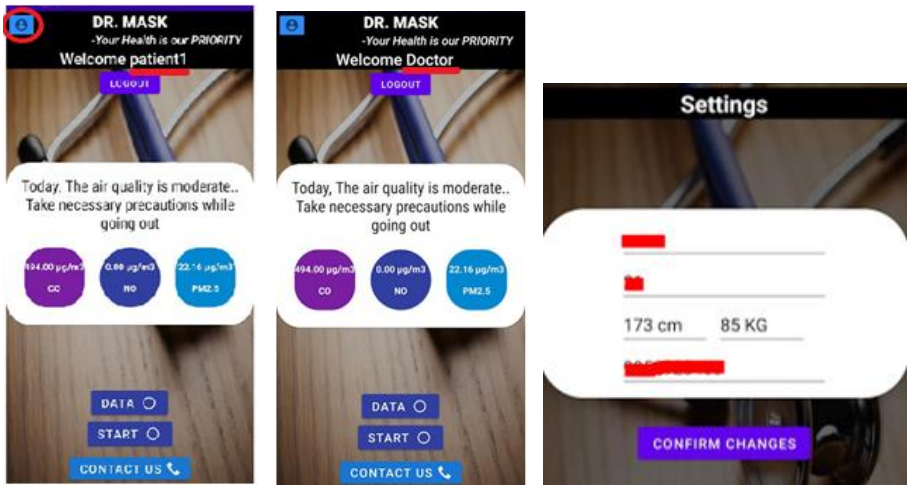
The application is developed to view the data which is transferred from the Google Firebase cloud. It provides the user to understand their condition during any part of the day at any place. The application is made through Android Studios, an official IDE Software for Google's Android Operating System. It is used to accelerate the development and help in building the highest-quality apps for every Android device. It provides a unified environment to build applications for Android phones, Tablets, Watches and Smart TV. The project is divided into units of functionality allowed by the structured code modules that one can independently build, test, and debug. Android studios can be coded in Java or Kotlin language and we used Kotlin language to code for the development of the application. The description of the entire code used is explained in detail in the chapter 7 Appendix, under section 7.2 Software/Application code.

The developed application is named as "Dr. Mask". Whenever the app gets opened user can find the login page as illustrated in figure 5.1.1. The new account needs to be created by the individual user using their email address. A separate username and password will be generated for future reference of accessing this application. Along with the user, his/her attender and doctor needs to create a new login account should monitor the patient history for the long run.

Figure 5.1.1 Login Page of the App

Once the login is made, the introduction page will provide the details about the air quality like Carbon Monoxide (CO), Nitrogen Monoxide (NO), and Particle matter (PM2.5) around the user. It also pops up a message stating about the air quality whether it is better, or poor as

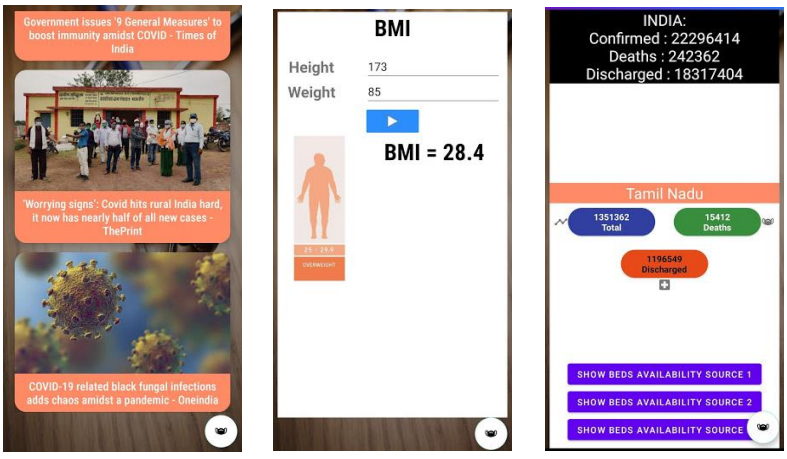
shown in figure 5.1.2 (a) & (b). The login details of the user and the doctor can be monitored from these figures, which are underlined in red. Individual profile editing can be made at any time by clicking the human icon on the top left of the page marked in the red circle as given in figure 5.1.2 (b).



(a) **(b)** **(c)**

Figure 5.1.2 (a) Home Page App for Patient1, (b) Doctor, and (c) User Details Setting Page

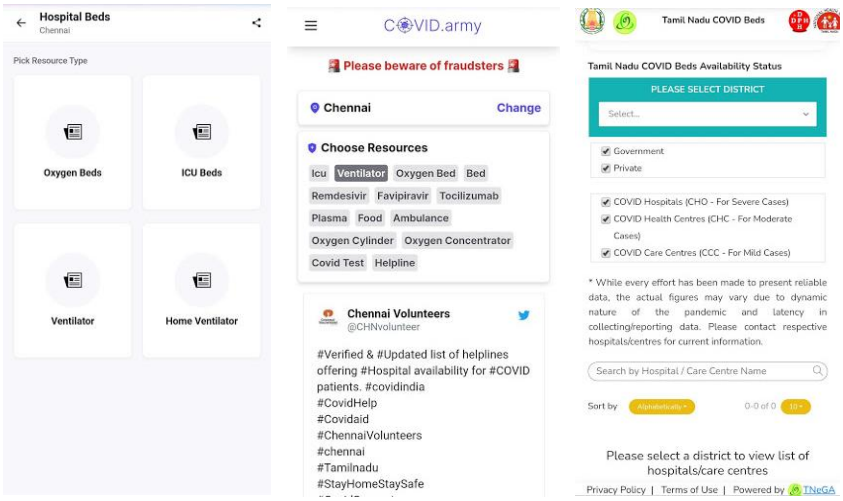
As shown in figure 5.1.2 (a) Home page has a START button, on clicking the same one can find the news page, and by clicking the MASK symbol below on the right corner, and helps one to navigate to the BMI calculator. COVID stats page of a particular state/district can be known as an additional feature in this application as shown in figure 5.1.3 (b) and (c).



(a) **(b)** **(c)**

Figure 5.1.3 (a) News, (b) BMI Calculator and (c) COVID Stats Pages of App

In figure 5.1.3 (c) one can easily get the information about bed availability with oxygen or with ventilator and other needs. Figure 5.1.4 shows all the pages that open when clicked the bed availability button.



(a) **(b)** **(c)**

Figure 5.1.4 (a) Source 1, (b) Source 2 (c) Source 3 respectively

Source 1 as shown in figure 5.1.4 (a) uses the Sprinklr platform to get messages regarding ventilators, ICU, and Oxygen beds. About home ventilators posted on Twitter can be visible here. Source 2 as shown in figure 5.1.4 (b) uses the Twitter platform to get messages posted regarding bed availability based on the resource and area selected. Source 3 in figure 5.1.4 (c) used to select the Tamil Nadu COVID Beds platform, where it gives the exact information about the bed availability in the selected district.

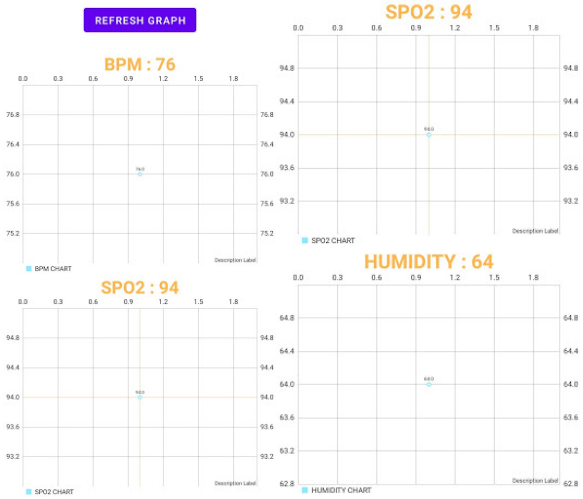
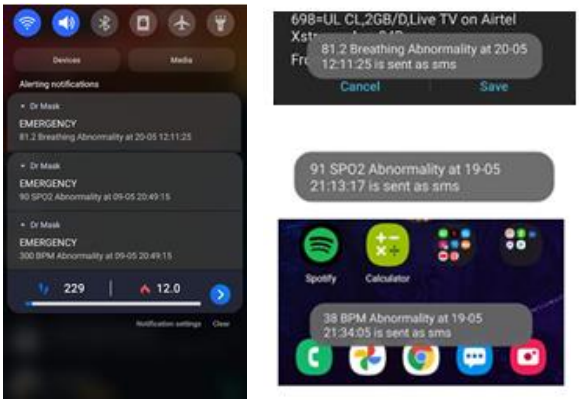


Figure 5.1.5 Graphical Visualization in the Data Page

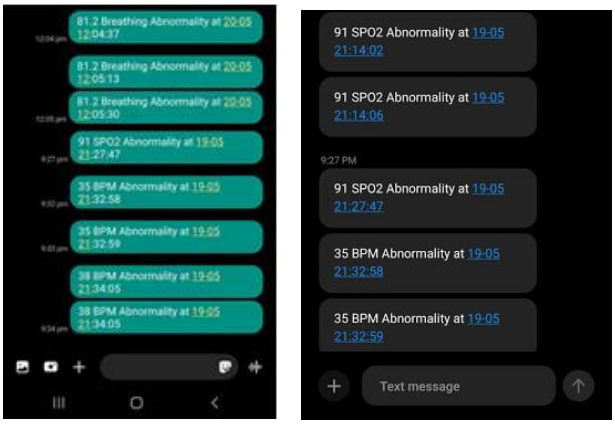
By clicking the DATA icon on the application home page, the user will be entering into the data page as shown in figure 5.1.2 (a). Data sent from ESP32 to Firebase Cloud, then to the Dr. Mask application can be viewed clearly in figure 5.1.5. Data values and graphs obtained can be shown at a time where it might be of 50 data. Depend on the stack size of the data, storage can take place through this application.



(a) **(b)**

Figure 5.1.6 (a) Notification with Sound and (b) Popup Message

When there is an abnormality, a notification appears with a sound for alerting the user as shown in figure 5.1.6 (a). The notification shows the abnormality that occurred in Bpm and SpO_2 values. Also, a message will pop up during this condition which is shown in figure 5.1.6 (b) and an SMS will be sent to the selected mobile number (attender/doctor) as shown in figure 5.1.7. Figure 5.1.7 (a) shows the message sent from the user's mobile and 5.1.7 (b) shows messages received by the mobile number that is set as attender/doctor.



(a) **(b)**

Figure 5.1.7 (a) SMS sent by Patient (b) SMS received by the Guardian Mobile

In figure 5.1.8 shows the page which opens, when contact information is pressed. It consists of email, SMS, contact number, location, and Webpage details.

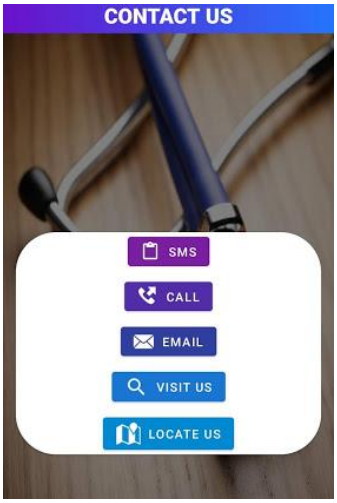
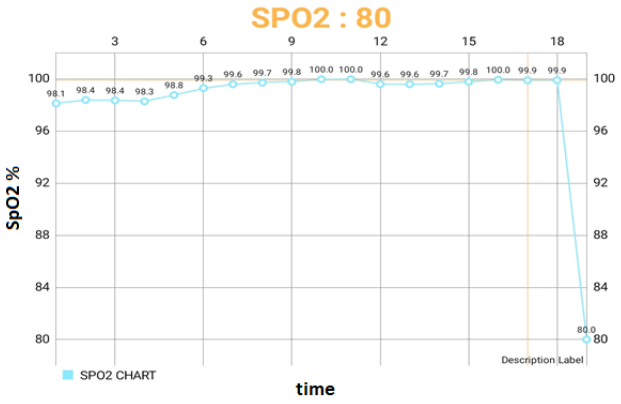


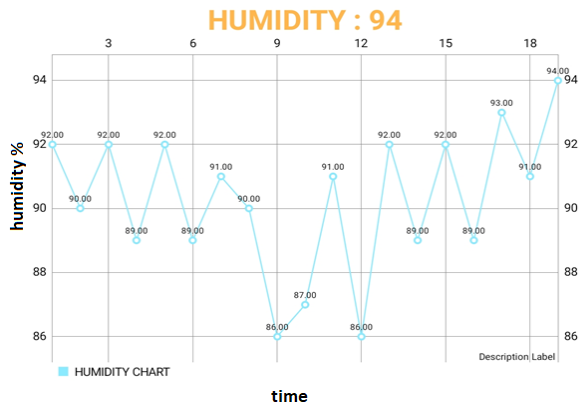
Figure 5.1.8 Contact Information Page

5.2 Graphical Results

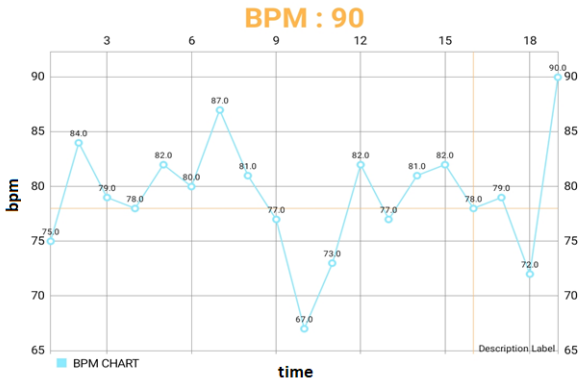
The data sent through the application are represented graphically. For the last 50 data, it helps the user instantly to get an idea about his/her condition in the last few minutes. In figure 5.2.1 (a), (b), and (c) shows the data recorded in the application and shown in the same way, as a user or doctor can view it.



(a)



(b)



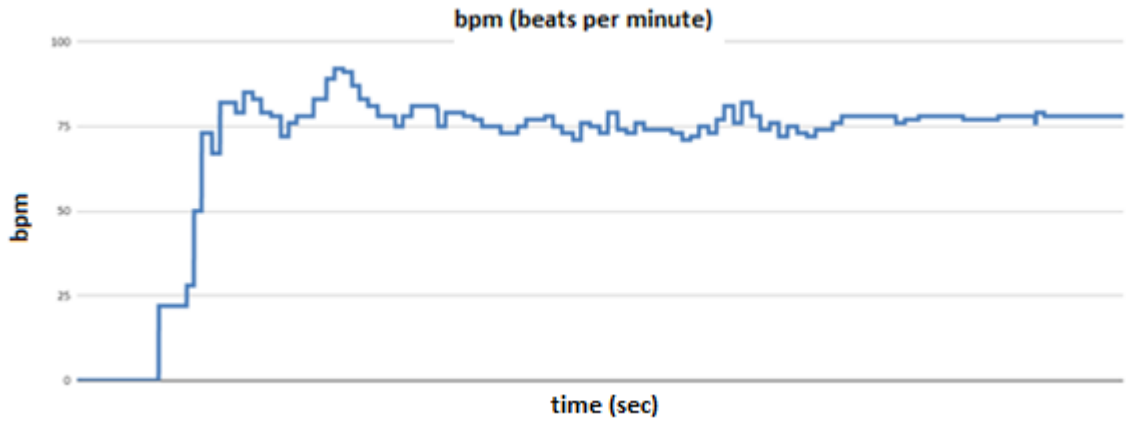
(c)

Figure 5.2.1 (a) Bpm, (b) SpO_2 and (c) Humidity Graphs Observed from App

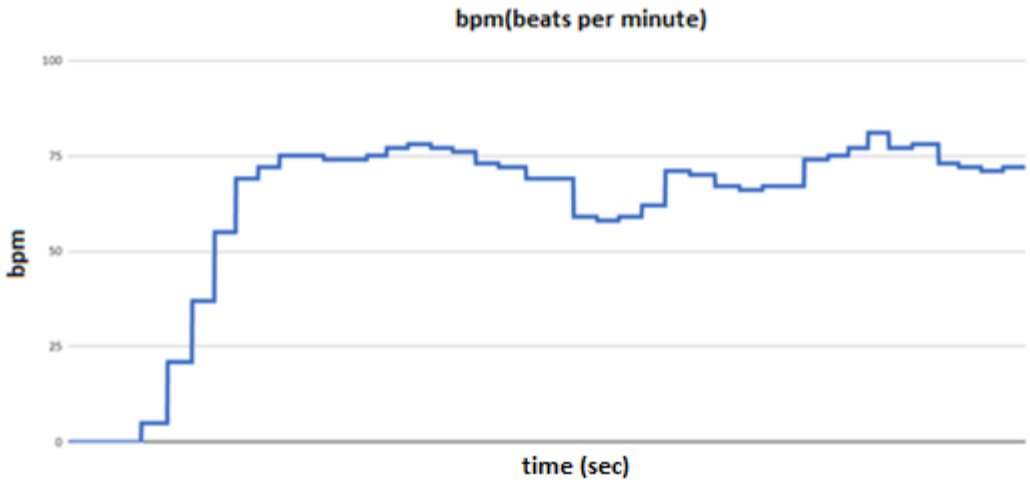
For further testing and finding the threshold values, the system is tested among different people and the data from the Firebase Cloud is collected in excel sheets and plotted the corresponding graphs to visualize the same.

In figure 5.2.2 the Bpm graph for different people with no abnormality shows that the sensor takes a few seconds which is 30 seconds to find the beats per minute, as it starts with zero, increases slowly, and stabilizes soon. The graphs are plotted between Bpm in the y-axis and time in the x-axis. Here the data is recorded every 0.3 seconds. The stabilized part of the graph in figure 5.2.2 (a) shows that the Bpm varies between 70 - 100. The graph in figure 5.2.2 (b) shows that the values change between 60 - 85, in figure 5.2.2 (c) shows the range is between 60 - 80, and figure 5.2.2 (d) shows the range between 75 - 90. On further analyzing various graphs we found that the normal resting Bpm range is between 60 - 100 while walking or not at rest Bpm should be in the range of 120 - 150. Therefore, the threshold value is chosen between 60 - 150. If the value is not in the range shows clearly the abnormality occurs to the

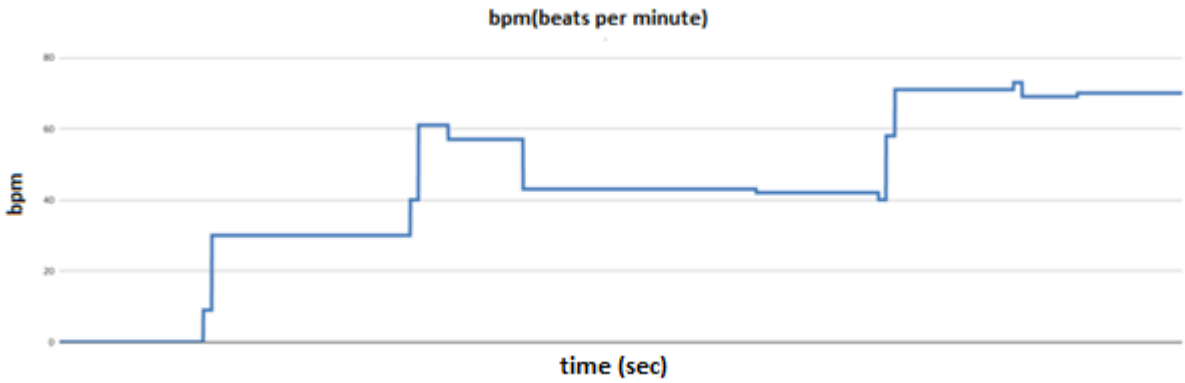
user and in need of an emergency.



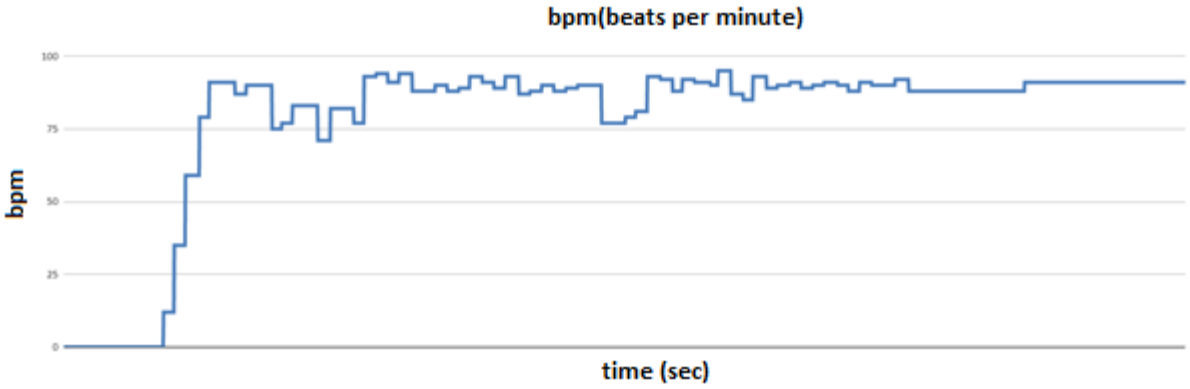
(a) Patient 1



(b) Patient 2



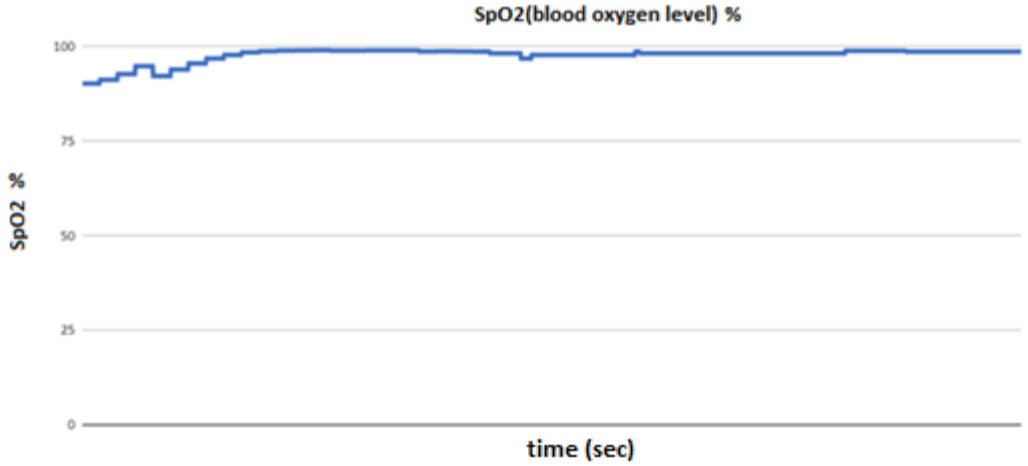
(c) Patient 3



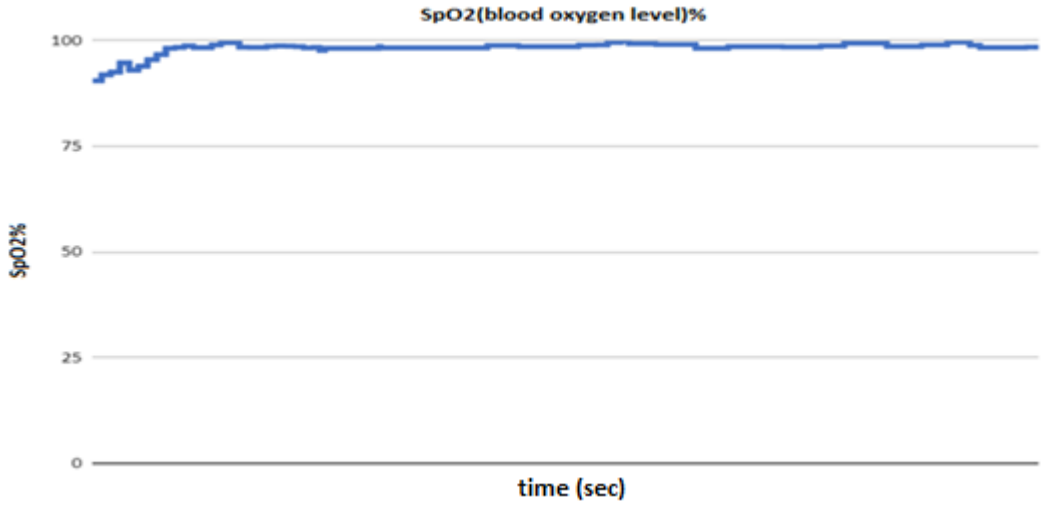
(d) Patient 4

Figure 5.2.2 Bpm Graph for Different People

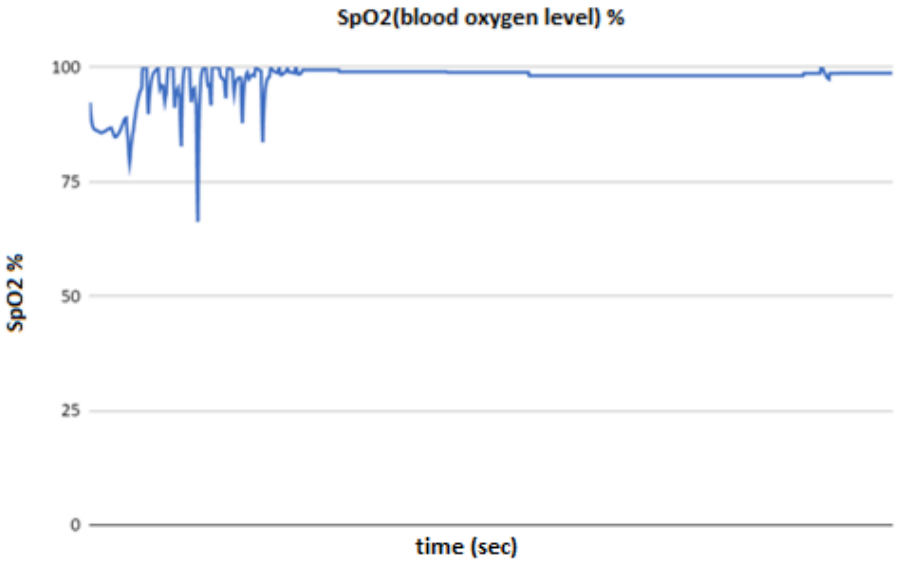
Figure 5.2.3 shows the SpO_2 graph observed from different people with no abnormality. The sensor took some time; generally, it took 30 seconds to stabilize the oxygen level reading. The graphs are plotted between SpO_2% on the y-axis and time on the x-axis. Here the data gets recorded every 0.6 seconds. The stabilized part of the reading in figure 5.2.3 (a) shows that the SpO_2 level is almost above 95%, it varies between 96% - 100%. The graph in figure 5.2.3 (b) clearly shows the values are varying between 97% - 100%. In figure 5.2.3 (c) after some time, the values are stabilized between 97% - 100%. Similarly in figure 5.2.3 (d) after stabilizing the values shows the variations occurred 98% -100%. Normal people SpO_2 values should be between 92% - 98% and below 92% is considered hypoxia [15]. For people with lung problems will have SpO_2 between 88% - 92% and lower than 88% is considered to be hypoxia. It shows clearly that in this condition enough oxygen is not reaching the body tissues, which makes the drop in oxygen level which goes to the brain cells. The patient may faint or feel dizzy. From further studying various graphs we found that the normal range is above 92% or 93%. If the SpO_2 percentage goes below 92% the user may faint, where they need emergency attention to resolve the issue.



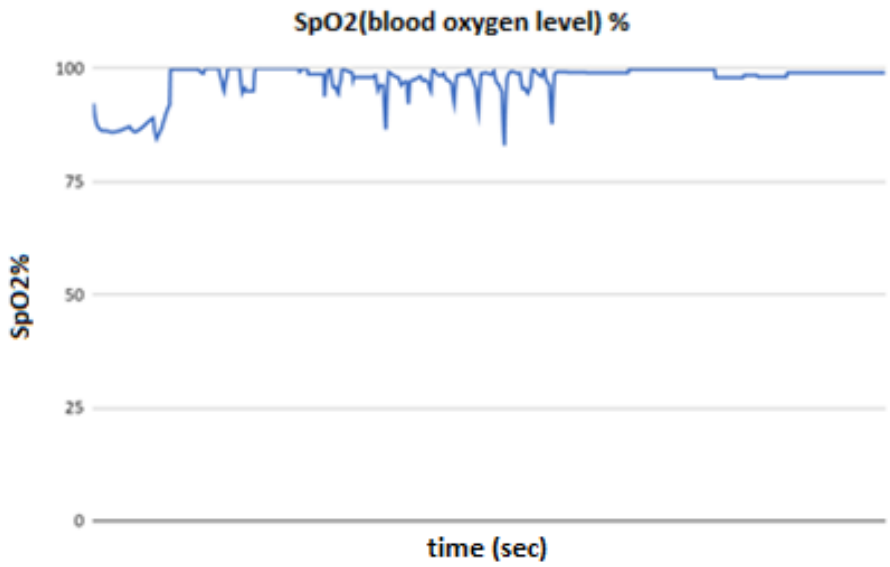
(a) Patient 1



(b) Patient 2



(c) Patient 3

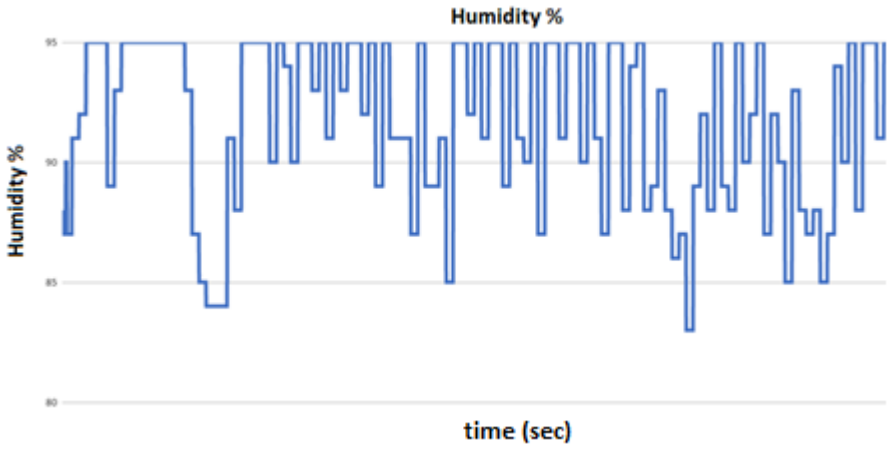


(d) Patient 4

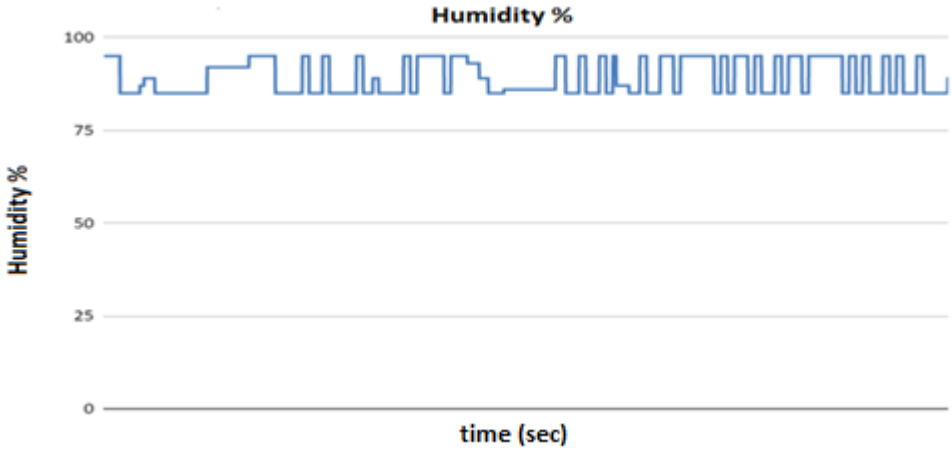
Figure 5.2.3 SpO_2 Graph for Different People

Figure 5.2.4 shows the graphs of humidity values for different people with no abnormalities. The sensor takes some time to stabilize the readings, generally, it took 30 seconds. The graphs are plotted between humidity % in the y-axis and time in the x-axis. Here the data is recorded every 0.3 seconds. The stabilized part of the reading in figure 5.2.4 (a) shows that the values range between 80% - 95% as the air let out has higher humidity than when taking air IN. It is observed through peaks and troughs in the graph. The average value is 91.045%. For the graph in figure 5.2.4 (b) the values range between 85% - 95% and the average value is 89.342%. In figure 5.2.4 (c) the values vary between 82% - 95% and the average value is 88.6769%. Similarly, for figure 5.2.4 (d) the values vary between 85% - 95% after ignoring the first few data as 91.395%. When an asthma attack occurs the time of low humidity is more than the time of high humidity because during the attack the time taken to breathe IN would be more as compared with breathes OUT. Therefore, there will be a decrease in the average breathing rate during an attack as compared with a normal case. When a person is experiencing an asthma attack the time of inhaling is longer than exhale. During inhaling the reading by humidity sensor will be the surrounding humidity (let it be X) and during exhale it is of humidity in the breath (let it be Y). The data acquired the number of X's will be more than Y's which keeps changing for each breath, and the X's value won't change usually. Therefore, the average will be closer to the X value during an attack. Further studying various

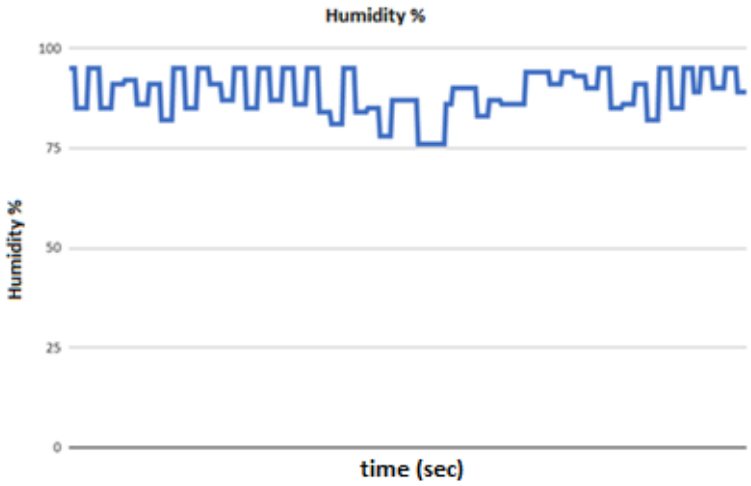
graphs, we find that the normal average is supposed to be a value in a percentage more than X+((Y-X))/4, if the average humidity for the recent data values is less than this, then it should be considered as an abnormality and the user experiencing an asthma attack, immediate attention should be given to prevent them from death.



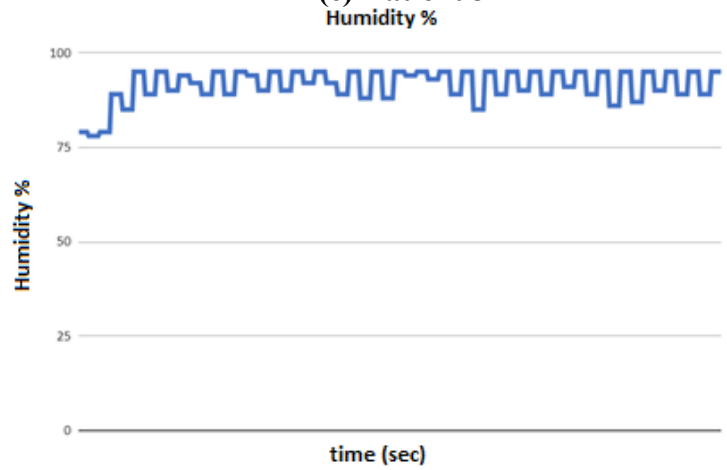
(a) Patient 1



(b) Patient 2



(c) Patient 3



(d) Patient 4

Figure 5.2.4 Humidity Graph for Different People

Table 5.2.1 shows all the threshold values that are tested and finalized after analyzing data obtained from different people.

Table 5.2.1 Threshold Values for each Parameter used in the Proposed System

TYPE	THRESHOLD VALUE/RANGE
Bpm	60 - 150
SpO_2	>92%
Average humidity	>X+((Y-X)/4) %

Table 5.2.2 compiles the results obtained from the Patients (namely 1, 2, 3 & 4) for each parameter from which we can conclude that all the data obtained are normal as the

average value is more than the threshold as shown in table 5.2.1. Bpm values are more than 60 for all patients, the SpO_2 values are more than 92%, and the humidity values are more than the threshold value, as calculated through the formula. The obtained data clearly shows that all patients are in normal condition.

Table 5.2.2 Results of each Patient with respect to each Parameter

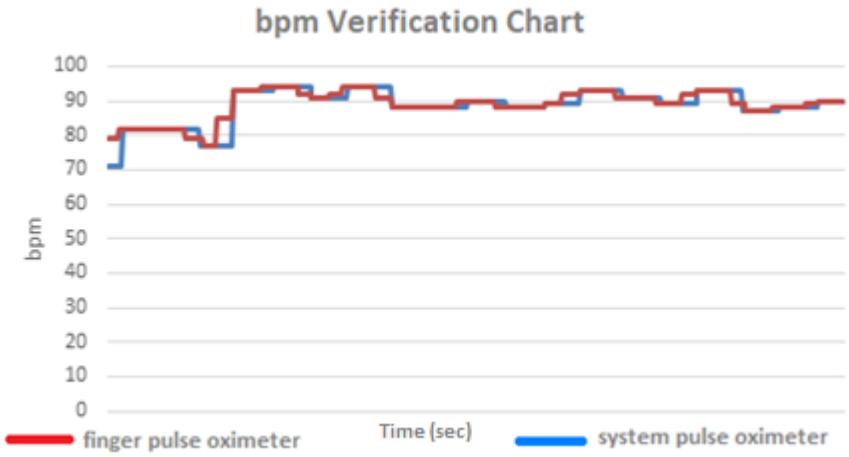
Data Type ⬇ / Patient ➡	Patient 1	Patient 2	Patient 3	Patient 4
Bpm (60-150)	Around 85	Around 72.5	Around 70	Around 82.5
SpO_2 (>92%)	>= 96%	>= 97%	>= 97%	>= 98%
Humidity (>X+((Y-X)/4) %)	91.045% (>83.75%)	89.342% (>87.5%)	88.6769% (>85.25%)	91.395% (>87.5%)

5.3 Verification of Results

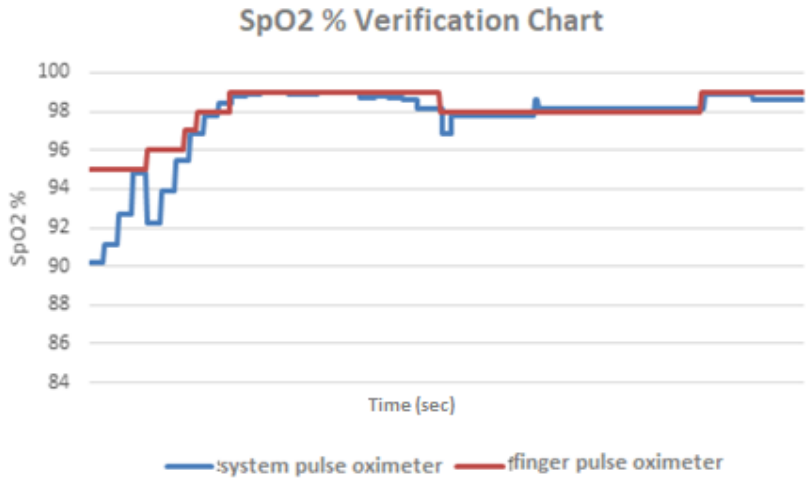
Figure 5.3.1 Finger Pulse Oximeter with Proposed System

Using the proposed model, we find the Bpm, humidity, and SpO_2 levels from various people. Here we compared how the data obtained from the sensors used in our system are so accurate to the existing real values. We used a finger pulse oximeter for the verification of Bpm and SpO_2 data observed from various people. Figure 5.3.1 shows simultaneously both

the reading values observed through finger pulse oximeter and system pulse oximeter used by MAX30102. The graphs are plotted with the help of excel sheets, where the data observed from the system and noted through the finger pulse oximeter are filled in the excel sheets. Repeated data values are noted and it is visualized that the graph can be plotted for any changes in system data values with repeating changes that occurred in the finger pulse oximeter values. The graphs in figure 5.3.2 show Bpm and SpO_2 data collected from the system and finger pulse oximeter. It is easily observed that both Bpm and SpO_2 values obtained from both the sensors are not varied drastically. The difference is very small so it can be negligible. Therefore, the data from the proposed system is confirmed that is accurate with the reference of the values obtained through a finger pulse oximeter.



(a)



(b)

Figure 5.3.2 System vs Finger Pulse Oximeter Graph for (a) Bpm and (b) SpO_2

5.4 Summary

In this chapter the application User Interface, Results obtained from the proposed model are explained properly. The obtained results are verified through a finger pulse oximeter as a reference and it is explained clearly here. The accuracy of our system is also verified and justified. In the next chapter, the conclusion of our work and planned future works using this system are also discussed.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

This chapter concludes the thesis on how to use this proposed system in Health Monitoring System for asthma and hypoxia patients. Additional functionalities for future implantation are also suggested here clearly.

6.1 Conclusion

A new health monitoring system is developed to help people in emergency conditions. Two different patients are considered into account, i.e., asthma and hypoxia. The proposed system includes a wearable hardware setup along with the developed application, and it is connected through the Google Firebase cloud. The application provides the user complete details once he/she created an account. It also stored the corresponding doctor information after the doctor logged in through his/her account. The attender for the user should also create a separate account to support the user when there is a need. Notification about the user's health condition will be sent through SMS or email to the user's attender and doctor when he/she meets the emergency condition. Alarm/notification can alert the user during an emergency and guide them to take immediate help from the people surrounding them at any different conditions.

The system is user-friendly as described in the thesis, it can be used by anyone at any time during a rainy or sunny day or even during their regular workout period. In this way, the user can be monitored continuously throughout the day. This model monitors the Bpm, breathing rate, and blood oxygen level of the user at various conditions. The whole proposed system is made on the mask and wristband, which is very normally used by many people nowadays. So it won't cause any inconvenience to the people and also due to its size reduction and less weight, it becomes easier to the user to use it in their daily life. This system is developed to aid patients with conditions that can't be cured like asthma, hypoxia and helped to avoid accidental deaths and immediate cures during emergency conditions.

6.2 Future work

There is enormous scope in improvising this project both at the software and hardware level.

- One main improvement is by adding Wi-Fi to the humidity sensor can make the system wireless, i.e., it reduces the inconvenience to the user which becomes the major improvement in hardware part.

- The improvement on the software part is by connecting the application to the current mobile location, where the system can send the exact location of the user to the doctor or attender or the nearest hospital if the user faced an emergency condition.

These improvements can make the system more user-friendly and give quick responses during emergency conditions.

CHAPTER 7
APPENDIX

CODES:

7.1 Hardware Code

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include "DHT.h"
#include <FirebaseESP32.h>
#include <FirebaseESP32HTTPClient.h>
#include <FirebaseJson.h>
#include "WiFi.h"
#define FB_HOST "https://fyp2021-2c051-default-rtdb.firebaseio.com/"
#define FB_AUTH "VHTU1CXqWWQdPcExx40Lk0t5eAwYw5nRGjybS1Hx"
#define WSSID "Lelouch"
#define WPASS "boom1234"
#define DHTTYPE DHT11
#define DHTPIN 18
DHT dht(DHTPIN, DHTTYPE);
MAX30105 particleSensor;
double aved = 0; double avir = 0;
double sirrms = 0;
double srrms = 0;
double h;
int i = 0;
int N_r = 100;
double ESpO2 = 95.0;
double FSpO2 = 0.7;
double f_rate = 0.95;
#define TIMETOBOOT 3000
#define SCALE 88.0
#define SAMPLING 5
#define FINGER_ON 30000
#define MINIMUM_SPO2 80.0
const byte R_SIZE = 4;
byte rates[R_SIZE];
byte rate_Spot = 0;
long last_Beat = 0;
float bpm;
int beat_Avg;
int j=0;
#define USEFIFO
FirebaseData firebaseData;
FirebaseJson json;
```

```
void setup()
{
  Serial.begin(115200);
  Serial.println("Initializing...");
  dht.begin();
  while (!particleSensor.begin(Wire, I2C_SPEED_FAST))
  {
    Serial.println("MAX30102 was not found. ");
  }
  byte ledBrightness = 0x7F;
  byte sampleAverage = 4;
  byte ledMode = 2;
  int sampleRate = 200;
  int pulseWidth = 411;
  int adcRange = 16384;
  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth,
adcRange);
  particleSensor.enableDIETEMPRDY();
  WiFi.begin(WSSID, WPASS);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();
  Firebase.begin(FB_HOST, FB_AUTH);
  Firebase.reconnectWiFi(true);
  Firebase.setReadTimeout(firebaseData, 1000 * 60);
  Firebase.setwriteSizeLimit(firebaseData, "tiny");
  Serial.println("------------------------------------");
  Serial.println("Connected...");
 xTaskCreate(SendToFirebase, "Send To Firebase",20000, NULL, 1, NULL);
}
void loop()
{
  j++;
  uint32_t ir_dat, red_dat , green_dat;
  double f_red, f_ir;
  double SpO2_dat = 0;
  long last_Msg = 0;
  long present = millis();
  if (present - lastMsg > 3000) {
    last_Msg = present;
  #ifdef USEFIFO
    particleSensor.check();
```

```
  while (particleSensor.available()) {
  #ifdef MAX30105
   red_dat = particleSensor.getFIFORed();
    ir_dat = particleSensor.getFIFOIR();
  #else
    red_dat = particleSensor.getFIFOIR();
    ir_dat = particleSensor.getFIFORed();
  #endif
    long ir_Value = particleSensor.getIR();
if (checkForBeat(ir_Value) == true)
{
long delta = millis() – last_Beat;
last_Beat = millis();
bpm= 60 / (delta / 1000.0);
if (bpm< 255 && bpm > 20)
{
rates[rate_Spot++] = (byte)bpm;
rate_Spot %= R_SIZE;
beat_Avg = 0;
for (byte x = 0 ; x < RATE_SIZE ; x++)
beat_Avg += rates[x];
beat_Avg /= RATE_SIZE;
}
}
Serial.print(", Avg BPM=");
Serial.print(beat_Avg);

if (irValue < 10000)
Serial.print(" No finger?");
    i++;
    f_red = (double)red_dat;
    f_ir = (double)ir_dat;
    aved = aved * f_rate + (double)red_dat * (1.0 – f_rate);
    avir = avir * f_rate + (double)ir_dat * (1.0 – f_rate);
    srrms += (f_red - aved) * (f_red - aved);
    sirrms += (f_ir - avir) * (f_ir - avir);
    if ((i % SAMPLING) == 0) {
      if ( millis() > TIMETOBOOT) {
        if (ir < FINGER_ON) ESpO2 = MINIMUM_SPO2;
          Serial.print(" Oxygen % = ");
        if( ESpO2 >= 100)
       {
        ESpO2=100;
        Serial.println("100");
       }
        else
       {
       Serial.println(ESpO2);
```

```
      }
     }
    }
     if ((i % N_r) == 0) {
      double R = (sqrt(srrms) / aved) / (sqrt(sirrms) / avir);
      // Serial.println(R);
      SpO2_dat = -23.3 * (R - 0.4) + 100;
//http://ww1.microchip.com/downloads/jp/AppNotes/00001525B_JP.pdf
      ESpO2 = FSpO2 * ESpO2 + (1.0 - FSpO2) * SpO2_dat;
      srrms = 0.0; sirrms = 0.0; i = 0;
      break;
     }
     particleSensor.nextSample();
   }
   h = dht.readHumidity();
   Serial.print(F("Humidity: "));
   Serial.print(h);
   Serial.print(F("\n"));
#endif
   }
}
void SendToFirebase( void * parameter )
{
for(;;)
{
Serial.println("\nSent to firebase\n");
json.set("/Value", double(beatAvg));
    Firebase.updateNode(firebaseData,"/BPM",json);
    json.set("/Value", ESpO2);
    Firebase.updateNode(firebaseData,"/SPO2",json);
    json.set("/Value", h);
    Firebase.updateNode(firebaseData,"/Humidity",json);
}
vTaskDelete( NULL );
}
```

7.2 Application Code

7.2.1 Manifest Part

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.capgemini.drmask">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.DrMask">
        <activity android:name=".FireBasePlotActivity"></activity>
        <activity android:name=".CovidBedsActivity" />
        <activity android:name=".AccountActivity" />
        <activity android:name=".authetication.LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".authetication.SignUpActivity" />
        <activity android:name=".ActivityStart" />
        <activity android:name=".ActivityContact" />
        <activity android:name=".MainActivity">
            <!--
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            -->
        </activity>
    </application>
</manifest>
```

7.2.2 Google Firebase Integration Keysets

```
{
  "project_info": {
    "project_number": "705831588980",
    "firebase_url": "https://fyp2021-2c051-default-rtdb.firebaseio.com",
    "project_id": "fyp2021-2c051",
    "storage_bucket": "fyp2021-2c051.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:705831588980:android:c29dd6cf29c30e44a0ed21",
        "android_client_info": {
          "package_name": "com.capgemini.drmask"
        }
      },
      "oauth_client": [
        {
          "client_id": "705831588980-
058ftn13erd80mqvbn5qnkeuls6q30ci.apps.googleusercontent.com",
          "client_type": 3
        }
```

```
    ],
    "api_key": [
      {
        "current_key": "AIzaSyC4CAg7Igffrniii3rb1IhxAbbrLfVgdOM"
      }
    ],
    "services": {
      "appinvite_service": {
        "other_platform_oauth_client": [
          {
            "client_id": "705831588980-
058ftn13erd80mqvbn5qnkeuls6q30ci.apps.googleusercontent.com",
            "client_type": 3
          }
        ]
      }
    }
  }
 ],
 "configuration_version": "1"
}
```

7.2.3 User Object

```
package com.capgemini.drmask.model
data class User(
    val id:String? = null,
    val name:String? = null,
    val emergencyPhone:String? = "",
    val height:String? = "",
    val weight:String? = "",
    val age:String? = ""
)
```

7.2.4 Main Page Activities: Air Pollution Statistics Display

```
package com.capgemini.drmask
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Toast
import com.capgemini.drmask.authetication.LoginActivity
import com.capgemini.drmask.retrofitdatabase.PollutionDbInterface
import com.capgemini.drmask.retrofitdatabase.PollutionDetails
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.*
```

```
import kotlinx.android.synthetic.main.activity_main.*
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
class MainActivity : AppCompatActivity() {
    private lateinit var auth: FirebaseAuth
    lateinit var ref:DatabaseReference
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        auth = FirebaseAuth.getInstance()
        ref = FirebaseDatabase.getInstance().getReference("users").child(auth.currentUser.uid)
        //POLLUTION INDEX RETROFIT
        val request =
PollutionDbInterface.getInstance().getPollutionDetails("https://api.openweathermap.org/data
/2.5/air_pollution?lat=13.1&lon=80.2707&appid=9ff634aced5342a274e1882a50f3960f")
        request.enqueue(object : Callback<PollutionDetails>{
            override fun onResponse(call: Call<PollutionDetails>, response:
Response<PollutionDetails>) {
                if(response.isSuccessful){
                    Log.d("Pollution","Success ${response.body()}")
                    val pollutionList = response.body()!!.list[0]
                    val aqi = pollutionList.main.aqi
                    val co =  String.format("%.2f", pollutionList.components.co)
                    val no = String.format("%.2f", pollutionList.components.no)
                    val pm = String.format("%.2f", pollutionList.components.pm2_5)
                    //   Where 1 = Good, 2 = Fair, 3 = Moderate, 4 = Poor, 5 = Very Poor.
                    var msg=""
                    when(aqi){
                        1->{
                            msg = "Today, The air quality is good.. Have a good day"
                        }
                        2->{
                            msg = "Today, The air quality is fair.. Have a good day"
                        }
                        3->{
                            msg = "Today, The air quality is moderate.. Take necessary precautions
while going out"
                        }
                        4->{
                            msg = "Today, The air quality is Poor.. We recommend not to head outside
without a mask or air filter"
                        }
                        5->{
                            msg = "Today, The air quality is Very Poor.. Stay indoors as it is harmful
levels"
                        }
                    }
                    mainAqiT.text = msg
```

```
                "$co µg/m3\n\nCO".also { mainCoT.text = it }
                "$no µg/m3\n\nNO".also { mainNoT.text = it }
                "$pm µg/m3\n\nPM2.5".also { mainPmT.text = it }
            }
        }
        override fun onFailure(call: Call<PollutionDetails>, t: Throwable) {
            Log.d("Pollution","${t.message}")
        }
    })
    ref.addValueEventListener(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            if(snapshot.exists())
            {
                val name = snapshot.child("name").value
                "Welcome $name".also { mainNameT.text = it }
            }
        }
        override fun onCancelled(error: DatabaseError) {
        }
    })
}
fun onButtonClicked(view: View) {
    when(view.id)
    {
        R.id.startB->{
            Toast.makeText(this,"Starting...",Toast.LENGTH_LONG).show()
            startActivity(Intent(this,ActivityStart::class.java))
        }
        R.id.contactB->{
            Toast.makeText(this,"Contact us",Toast.LENGTH_LONG).show()
            startActivity(Intent(this,ActivityContact::class.java))
        }
        R.id.mainLogoutB->{
            val currentUser = auth.currentUser
            if(currentUser!=null)
            {
                Toast.makeText(this, "Logging Out", Toast.LENGTH_SHORT).show()
                auth.signOut()
            }
            startActivity(Intent(this,LoginActivity::class.java))
            finish()
        }
        R.id.mainAccountB->{
            startActivity(Intent(this,AccountActivity::class.java))
        }
        R.id.dataB->{
            startActivity(Intent(this,FireBasePlotActivity::class.java))
        }
    }
```

```
    }
}
```

7.2.5 Login Activity

```
package com.capgemini.drmask.authetication
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import com.capgemini.drmask.MainActivity
import com.capgemini.drmask.R
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import kotlinx.android.synthetic.main.activity_login.*
class LoginActivity : AppCompatActivity() {
    private lateinit var auth: FirebaseAuth
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        auth = FirebaseAuth.getInstance()
        loginB.setOnClickListener {
            doLogin()
        }
        newUserT.setOnClickListener {
            startActivity(Intent(this, SignUpActivity::class.java))
            finish()
        }
    }
    private fun doLogin() {
        if(loginEmailE.text.isEmpty()){
            loginEmailE.error ="Please enter email"
            loginEmailE.requestFocus()
            return
        }
        if(loginPasswordE.text.isEmpty()){
            loginPasswordE.error ="Please enter password"
            loginPasswordE.requestFocus()
            return
        }
        //----LOGIN USER---
        auth.signInWithEmailAndPassword(loginEmailE.text.toString(),
loginPasswordE.text.toString())
                .addOnCompleteListener(this) { task ->
                    if (task.isSuccessful) {
                        val user = auth.currentUser
                        Toast.makeText(this, "Authentication Successful",
Toast.LENGTH_SHORT).show()
                        updateUI(user)
```

```
                } else { //wrong details
                    Toast.makeText(this, "${task.exception?.message}",
Toast.LENGTH_SHORT).show()
                }
            }
    }
    public override fun onStart() {
        super.onStart()
        val currentUser = auth.currentUser
        if(currentUser!=null)
        {
            Toast.makeText(this, "Automatically Signing you in",
Toast.LENGTH_SHORT).show()
            updateUI(currentUser)
        }
    }
    private fun updateUI(currentUser : FirebaseUser?){
        startActivity(Intent(this,MainActivity::class.java))
        finish()
    }
}
```

7.2.6 New Account Creation Activity

```
package com.capgemini.drmask.authetication

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import com.capgemini.drmask.AccountActivity
import com.capgemini.drmask.MainActivity
import com.capgemini.drmask.R
import com.capgemini.drmask.model.User
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import com.google.firebase.auth.UserProfileChangeRequest
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.FirebaseDatabase
import kotlinx.android.synthetic.main.activity_sign_up.*
class SignUpActivity : AppCompatActivity() {
    private lateinit var auth: FirebaseAuth
    private lateinit var ref :DatabaseReference
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sign_up)
        auth = FirebaseAuth.getInstance()
        ref = FirebaseDatabase.getInstance().getReference("users")
```

```
        signUpBtn.setOnClickListener {
            signUpUser()
        }
        existingUserT.setOnClickListener {
            startActivity(Intent(this, LoginActivity::class.java))
            finish()
        }
    }
    private fun signUpUser() {
        if(signupEmailE.text.isEmpty()){
            signupEmailE.error ="Please enter email"
            signupEmailE.requestFocus()
            return
        }
        if(signupPasswordE.text.isEmpty()){
            signupPasswordE.error ="Please enter password"
            signupPasswordE.requestFocus()
            return
        }
        if(nameE.text.isEmpty()){
            nameE.error ="Please enter name"
            nameE.requestFocus()
            return
        }
        //----CREATE USER---
        auth.createUserWithEmailAndPassword(signupEmailE.text.toString(),
signupPasswordE.text.toString())
            .addOnCompleteListener{ task ->
                if (task.isSuccessful) {
                    val user = auth.currentUser
                    saveUser()
                    updateUI(user)
                } else {
                    Toast.makeText(this, "${task.exception?.message}",
                        Toast.LENGTH_SHORT).show()
                }
            }
    }
    private fun saveUser() {
        val userid = auth.currentUser.uid
        val user = User(id = userid,name = nameE.text.toString())
        ref.child(userid).setValue(user).addOnCompleteListener {
          Log.d("SignUp","UserAdded")
        }
    }
    private fun updateUI(user: FirebaseUser?) {
        Toast.makeText(this, "Added", Toast.LENGTH_SHORT).show()
        val intent = Intent(this, AccountActivity::class.java)
        intent.putExtra("new",true)
```

```
        startActivity(intent)
        finish()
    }
}
```

7.2.7 Plot Code

The reference [16] was used to plot graph in the application using the data received from the firebase cloud, this is the code used in application from [16] as reference.

```
package com.capgemini.drmask
import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Intent
import android.os.Build
import android.os.Bundle
import android.telephony.SmsManager
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.github.mikephil.charting.charts.LineChart
import com.github.mikephil.charting.data.Entry
import com.github.mikephil.charting.data.LineData
import com.github.mikephil.charting.data.LineDataSet
import com.github.mikephil.charting.interfaces.datasets.ILineDataSet
import com.google.firebase.database.*
import kotlinx.android.synthetic.main.activity_fire_base_plot.*
import kotlinx.android.synthetic.main.activity_main.*
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
import java.util.*
import kotlin.collections.ArrayList
class FireBasePlotActivity : AppCompatActivity() {
    lateinit var humidityDataValues : MutableList<String>
    lateinit var bpmDataValues : MutableList<String>
    lateinit var spo2DataValues : MutableList<String>
    var SAMPLE_SIZE = 50
    lateinit var bpmChart : LineChart
    lateinit var spo2Chart : LineChart
    lateinit var humChart : LineChart
    var maxHum = 0.0
    var minHum = 100.0
    var normalHum =90.0
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_fire_base_plot)
        val ref1 = FirebaseDatabase.getInstance().getReference("BPM").child("Value")
        val ref2 = FirebaseDatabase.getInstance().getReference("Humidity").child("Value")
```

```
        val ref3 = FirebaseDatabase.getInstance().getReference("SPO2").child("Value")
        humidityDataValues = mutableListOf()
        bpmDataValues = mutableListOf()
        spo2DataValues = mutableListOf()
        val bpmGraphValues = ArrayList<Entry>()
        val spo2GraphValues = ArrayList<Entry>()
        val humidityGraphValues = ArrayList<Entry>()
        //GRAPHS
        refreshB.setOnClickListener {
            //bpm
            bpmGraphValues.clear()
            for(x in 1..bpmDataValues.size)
                bpmGraphValues.add(Entry((x).toFloat(),(bpmDataValues[x-1]).toFloat()))
            bpmChart = findViewById(R.id.bpmGraph)
            val lineDataSetBpm = LineDataSet(bpmGraphValues,"BPM CHART")
            val dataSetsBpm = ArrayList<ILineDataSet>()
            dataSetsBpm.add(lineDataSetBpm)
            val dataBpm = LineData(dataSetsBpm)
            bpmChart.data=dataBpm
            bpmChart.invalidate()

            //spo2
            spo2GraphValues.clear()
            for(x in 1..spo2DataValues.size)
                spo2GraphValues.add(Entry((x).toFloat(),(spo2DataValues[x-1]).toFloat()))
            spo2Chart = findViewById(R.id.spo2Graph)
            val lineDataSetSpo2 = LineDataSet(spo2GraphValues,"SPO2 CHART")
            val dataSetsSpo2 = ArrayList<ILineDataSet>()
            dataSetsSpo2.add(lineDataSetSpo2)
            val dataSpo2 = LineData(dataSetsSpo2)
            spo2Chart.data=dataSpo2
            spo2Chart.invalidate()
            //spo2
            humidityGraphValues.clear()
            for(x in 1..humidityDataValues.size)
                humidityGraphValues.add(Entry((x).toFloat(),(humidityDataValues[x-
1]).toFloat()))
            humChart = findViewById(R.id.humidityGraph)
            val lineDataSetHum = LineDataSet(humidityGraphValues,"HUMIDITY CHART")
            val dataSetsHum = ArrayList<ILineDataSet>()
            dataSetsHum.add(lineDataSetHum)
            val dataHum = LineData(dataSetsHum)
            humChart.data=dataHum
            humChart.invalidate()
        }
        ref1.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                if(snapshot.exists())
                {
```

```
            val data  = snapshot.value.toString()
            //CONSTRAINT
            if(data.toDouble()<40 ||data.toDouble()>150) {
               sendNotification("$data BPM Abnormality",1)
            }
            bpmDataValues.add(data)
            bpmT.text ="BPM : $data"
            if(bpmDataValues.size>=SAMPLE_SIZE)
               bpmDataValues.removeFirst()
         }
      }
      override fun onCancelled(error: DatabaseError) {
      }
   })
   ref2.addValueEventListener(object : ValueEventListener {
      override fun onDataChange(snapshot: DataSnapshot) {
         if(snapshot.exists())
         {
            val data  = snapshot.value.toString()
            humidityDataValues.add(data)
            humidityT.text ="HUMIDITY : $data"
            if(humidityDataValues.size>=SAMPLE_SIZE) {
               humidityDataValues.removeFirst()
               var sumHum=0.0
               for (x in humidityDataValues)
                  sumHum+=x.toDouble()
               sumHum/=SAMPLE_SIZE
               if(sumHum<normalHum){
                  sendNotification("$sumHum Breathing Abnormality",2)
               }
            }
            if(humidityDataValues.size==10) {
               for (x in humidityDataValues) {
                  if (x.toDouble() > maxHum) maxHum = x.toDouble()
                  if (x.toDouble() < minHum) minHum = x.toDouble()
               }
               normalHum = minHum + (maxHum-minHum)/4
            }
         }
      }
      override fun onCancelled(error: DatabaseError) {
      }
   })
   ref3.addValueEventListener(object : ValueEventListener {
      override fun onDataChange(snapshot: DataSnapshot) {
         if(snapshot.exists())
         {
            val data  = snapshot.value.toString()
            //CONSTRAINT
```

```
                if(data.toDouble()<=92){
                    sendNotification("$data SPO2 Abnormality",3)
                }
                spo2DataValues.add(data)
                spo2T.text ="SPO2 : $data"
                if(spo2DataValues.size>=SAMPLE_SIZE)
                    spo2DataValues.removeFirst()
            }
        }
        override fun onCancelled(error: DatabaseError) {
        }
    })
}
private fun sendNotification(msg: String, id: Int) {
    //Getting intent and PendingIntent instance
    //Getting intent and PendingIntent instance
    val intent = Intent(applicationContext, MainActivity::class.java)
    val pi = PendingIntent.getActivity(applicationContext, 0, intent, 0)

    var notifyMessage =msg
    val nManager =getSystemService(NOTIFICATION_SERVICE) as
NotificationManager
        val builder: Notification.Builder = if(Build.VERSION.SDK_INT >=
Build.VERSION_CODES.O) {//checks version
            val channel = NotificationChannel("test", "Reminder Done",
NotificationManager.IMPORTANCE_HIGH)
            nManager.createNotificationChannel(channel)

            val date = LocalDateTime.now()
            val time = DateTimeFormatter
                .ofPattern("dd-MM HH:mm:ss").format(date)
            notifyMessage ="$msg at $time"
            Notification.Builder(this, "test")
        }
        else  Notification.Builder(this)
    //Get the SmsManager instance and call the sendTextMessage method to send message
    //Get the SmsManager instance and call the sendTextMessage method to send message
        builder.setSmallIcon(R.drawable.ic_launcher_foreground)
        builder.setContentTitle("EMERGENCY")
        builder.setContentText(notifyMessage)
        builder.setAutoCancel(true)
        val myNotify = builder.build()
        nManager.notify(id, myNotify)
    Toast.makeText(this@FireBasePlotActivity,"$notifyMessage is sent as
sms",Toast.LENGTH_SHORT).show()
    val sms: SmsManager = SmsManager.getDefault()
    sms.sendTextMessage("7010153098", null, notifyMessage, pi, null)
}
}
```

REFERENCES

[1] Avrajith Ghosh, Arnab Raha and Amitava Mukherjee, "Energy-Efficient IoT-Health Monitoring System using Approximate computing", *Internet of Things*, vol. 9, pp. 100166 1-17, Jan. 2020.

[2] Ahmed Abdelgwad, Ahmed Khattab and Kumar Yelamarthi, "IoT- Health Based Monitoring system for Active and Assisted living", *International Conference on Smart Objects and Technologies for Social Good,* vol. 195, pp. 11-20, Jul. 2017.

[3] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci and Silvana Andreescu, "Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges", *IEEE International Conference on Services Computing*, pp. 258-292, Jun. 2015.

[4] Md. Milon Islam, Ashikur Rahaman and Md. Rashedul Islam, "Development of Smart Healthcare Monitoring System in IoT Environment", *SN Computer Science*, vol. 1, pp.185 1-11, May. 2020.

[5] D.Shiva Rama Krishnan, Subhash Chand Gupta and Tanupriya Choudhury, "An IoT based Patient Health Monitoring System", *International Conference on Advances in Computing and Communications Engineering*, pp. 01-07, Jun. 2018.

[6] S Lakshmanachari, C. Srihari, A.Sudhakar and Paparao Nalajala, "Design and Implementation of Cloud based Patient Health Care Monitoring Systems using IoT", *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pp. 3713-3717, 2017.

[7] Afef Mdhaffar, Tarak Chaari, Kaouthar Larbi, Mohamed Jmaiel and Bernd Freisleben, "IoT-based Health Monitoring via LoRaWAN", *IEEE EUROCON 2017 -17th International Conference on Smart Technologies*, pp. 519-524*,* Jul. 2017.

[8] M. Sathya, S. Madhan and K. Jayanthi, "Internet of things (IoT) based health monitoring system and challenges", *International Journal of Engineering and Technology*, vol. 7(1), pp. 175-178, Feb. 2018.

[9] Haipeng Liu, John Allen, Dingchang Zheng and Fei Chen, "Recent development of respiratory rate measurement technologies", *Physiol Meas.*, vol. 40(7), pp. 1-27, Aug. 2019.

[10] AndreaAliverti, "Wearable technology: role in respiratory health and disease", *Breathe (Sheff)*, vol. 13(2), pp. e27-e36, Jun. 2017.

[11] Syed Tauhid Ullah Shah, Fiazan Badshah, Faheem Dad, Nouman Amin and Mian Ahmad, "Cloud-Assisted IoT-Based Smart Respiratory Monitoring System for Asthma Patients", *Applications of Intelligent Technologies in Healthcare*, pp. 77-86, Nov. 2018.

[12] S. Mohanraj and K.Sakthisudhan, "An Internet of Things based Smart Wearable System for Asthma Patients", *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 7, pp. 604- 607, Mar. 2019.

[13] Leonardo Juan Ramirez Lopez, Gabriel Puerta Aponte and Arturo Rodriguez Garcia, "Internet of Things Applied in Healthcare Based on Open Hardware with Low-Energy Consumption", *Healthc Inform Res.*, vol. 25(3), pp. 230-235, Jul. 2019.

[14] Dhanurdhar Murali, Deepthi R Rao, Swathi R Rao and Prof. Ananda M, "Pulse Oximetry and IOT based Cardiac Monitoring Integrated Alert System", *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2237-2243, 2018.

[15] https://med.libretexts.org/Bookshelves/Nursing/Book%3A_Clinical_Procedures_for_Safer_Patient_Care_(Doyle_and_McCutcheon)/05%3A_Oxygen_Therapy/5.04%3A_Signs_and_Symptoms_of_Hypoxia

[16] https://github.com/PhilJay/MPAndroidChart

CHAPTER 8

BIODATA

Name : Raya Swethaa
Mobile Number : 9952928460
Email : raya.swethaa2017@vitstudent.ac.in
Permanent Address : 1144/3, Anna Nagar West End Colony
Chennai – 600050, Tamil Nadu,
India.

Name : Aditya Harichandar
Mobile Number : 9444964530
Email : adityaharichandar.a2017@vitstudent.ac.in
Permanent Address : E6D, CDS Regal Palm Gardens,
10/298, Seetharam Nagar,
Velachery Main Road,
Velachery, Chennai - 600042

DEMONSTRATION VIDEO

This is the YouTube link for the Video Demonstration of the Health Monitoring Device developed using IoT for asthma and hypoxia patients -

https://www.youtube.com/watch?v=R7g17s8m7d0