Graphical user interface, website

Description automatically generated

The department of Medical Engineering

**Project Name: Wristband identification and measurement for patient**

Project Engineering Report

|  |  |
| --- | --- |
| Doron Gusenberg , Yuval Shapira | Students Names: |
| 318183589 , 312486665 | ID Numbers: |
| Dr. Nadav Sheffer | Supervisor's Name: |
|  | Approved By: |
| 09/06/2023 | Submission Date: |

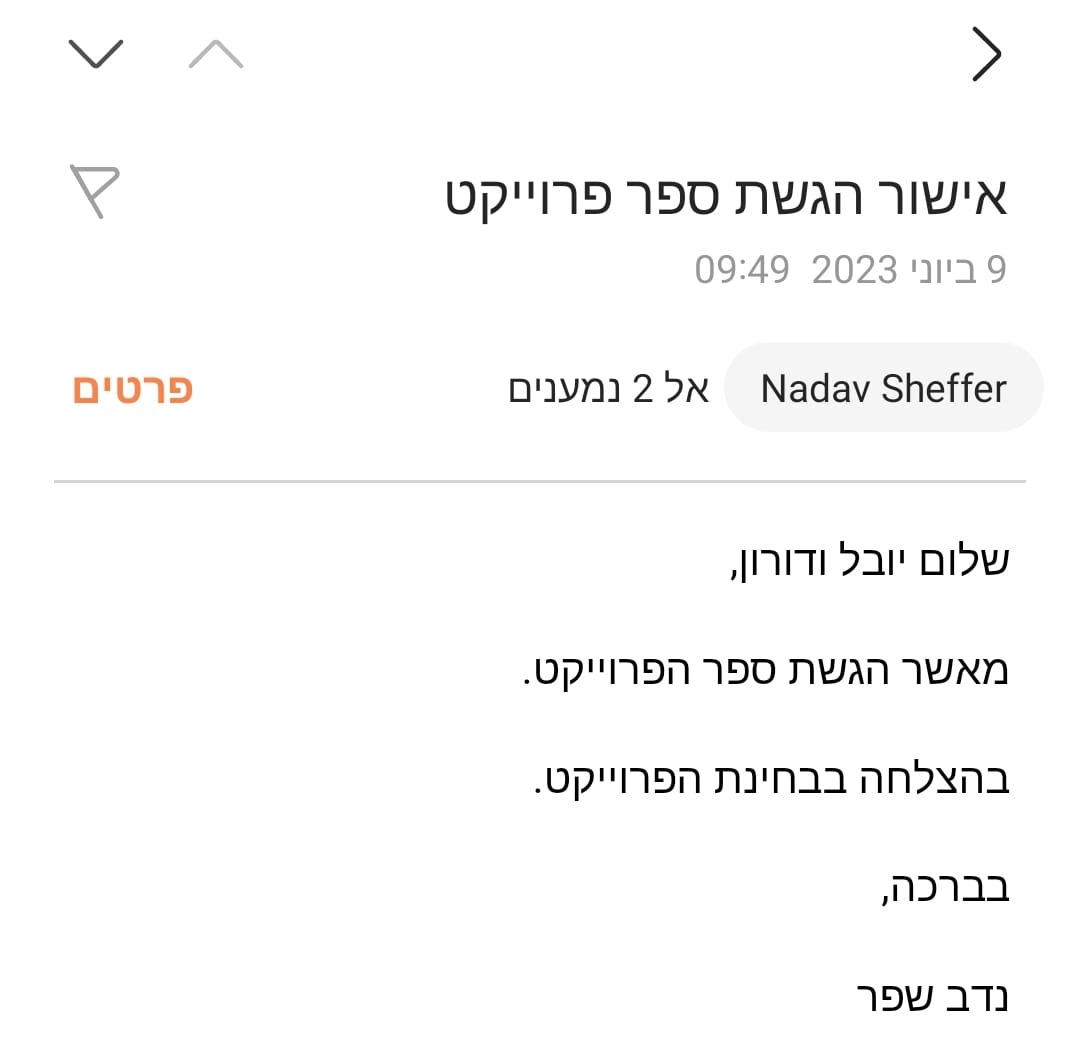


Table of Contents

[**1.** **Executive Summery** 4](#_Toc137200359)

[**2.** **Dictionary of terms** 5](#_Toc137200360)

[**3.** **Introduction** 6](#_Toc137200361)

[**4.** **Literature Review** 7](#_Toc137200362)

[**Heart Rate:** 7](#_Toc137200363)

[**Oxygen Saturation:** 8](#_Toc137200364)

[**Optic sensor (for measuring HR and PO):** 8](#_Toc137200365)

[**Overcoming Patient concerns about privacy:** 8](#_Toc137200366)

[**Systemic alternatives:** 9](#_Toc137200367)

[**RFID:** 9](#_Toc137200368)

[**Bluetooth:** 9](#_Toc137200369)

[**Image processing face recognition:** 9](#_Toc137200370)

[**Market Survey:** 9](#_Toc137200371)

[**Products at the market:** 9](#_Toc137200372)

[**Zebra ZD510-HC Wristband Printing Solution | ZD510-HC:** 10](#_Toc137200373)

[**Apple Watch:** 10](#_Toc137200374)

[**Garmin Fenix 7:** 11](#_Toc137200375)

[**Fitbit Charge 5:** 11](#_Toc137200376)

[**Table of comparing:** 12](#_Toc137200377)

[**5.** **Methods** 13](#_Toc137200378)

[**Block diagram:** 13](#_Toc137200379)

[**Flow chart:** 14](#_Toc137200380)

[**Required Means and Tools:** 15](#_Toc137200381)

[**Study and research:** 15](#_Toc137200382)

[Arduino Uno: 15](#_Toc137200383)

[GPS component- GY-GPS6MV2: 17](#_Toc137200384)

[MAX30102 - Heart Rate and SpO2 sensor: 18](#_Toc137200385)

[LED: 20](#_Toc137200386)

[Free Software 22](#_Toc137200387)

[**6.** **Results**.. 22](#_Toc137200388)

[**Information system** 22](#_Toc137200389)

[**The home page** 22](#_Toc137200390)

[**Patient administration** 23](#_Toc137200391)

[**Patients insert window** 23](#_Toc137200392)

[**Update patient data by ID** 24](#_Toc137200393)

[**Update patient data** 24](#_Toc137200394)

[**Search patient data** 25](#_Toc137200395)

[**Delete patient** 25](#_Toc137200396)

[**Patient database view** 25](#_Toc137200397)

[**Doctor administration** 26](#_Toc137200398)

[**Doctor Insert window** 26](#_Toc137200399)

[**Update doctor data by ID** 27](#_Toc137200400)

[**Delete doctor** 27](#_Toc137200401)

[**Doctor database view** 28](#_Toc137200402)

[**Patient measurement's view** 28](#_Toc137200403)

[**ERD** 28](#_Toc137200404)

[**Measurements** 29](#_Toc137200405)

[**Heart rate** 29](#_Toc137200406)

[**GPS** 30](#_Toc137200407)

[**Saturation** 31](#_Toc137200408)

[**Alert system** 32](#_Toc137200409)

[**7.** **Discussion** 33](#_Toc137200410)

[**8.** **Summary and Conclusions** 34](#_Toc137200411)

[**9.** **References** 34](#_Toc137200412)

[**10.** **Appendices** 36](#_Toc137200413)

[**Gannt chart:** 36](#_Toc137200414)

[**Changes made during the project:** 39](#_Toc137200415)

[**The project poster:** 40](#_Toc137200416)

[**Scientific journal article:** 40](#_Toc137200417)

[**Other information:** 40](#_Toc137200418)

[**Tables, Figures and Code:** 40](#_Toc137200419)

# **Executive Summery**

The system is intended to provide comfortable way to get patient information and measurements. Our smart wristband project identifies patients and tracks physiology using an Arduino board and sensors for heart rate, oxygen saturation and location. Data is sent to a computer via serial communication and displayed on the screen, then stored in a database. The system is controlled using a graphical user interface.

The system is intended to provide more data in more convenient and safer way to help the medical stuff deal with the patients.

# **Dictionary of terms**

Table 2. 1 Dictionary of terms

|  |  |
| --- | --- |
| Term | Explanation |
| Heart rate | Refers to the number of times the heart beats per minute. It is an essential indicator of cardiovascular health. |
| Peripheral oxygen saturation (SpO2) | Also known as blood-oxygen saturation, measures the percentage of oxygen-bound hemoglobin in the blood. It provides information about the efficiency of oxygen delivery to the body's tissues. |
| Hemoglobin | A protein molecule found in red blood cells that carries oxygen from the lungs to the body's tissues and removes carbon dioxide. |
| Photoplethysmography (PPG) | Non-invasive optical technique used to detect changes in blood volume in peripheral blood vessels. It measures the pulsatile changes in light absorption, typically using a light-emitting diode (LED) and a photodetector. |
| Photodetector | Device that converts light energy into an electrical signal, commonly used in applications such as optical communication, imaging systems, and light sensing. |
| Plethysmogram | A graphical representation of changes in blood volume over time, typically obtained using photoplethysmography. It shows the pulsatile nature of blood flow. |
| Photodiode | A semiconductor device that converts light into an electrical current. It is commonly used in photoplethysmography to detect changes in light intensity caused by blood volume variations. |
| Arteriolar vessels | Small arteries that regulate blood flow by constricting or dilating. They help control blood pressure and direct blood flow to different tissues. |
| Pulsatile energy | The energy associated with the pulsatile or rhythmic nature of blood flow caused by the heart's contractions. |
| Capillaries | Tiny blood vessels that connect the smallest arteries (arterioles) to the smallest veins (venules). They facilitate the exchange of nutrients and waste products between the blood and tissues. |
| Radio Frequency Identification (RFID) | Technology that uses radio waves to wirelessly identify and track objects or individuals. RFID tags consist of a small integrated circuit and an antenna. |
| Ultra-High Frequency (UHF) | Range of radio frequencies typically between 300 MHz and 3 GHz. UHF frequencies are commonly used in various communication applications. |
| Industrial, Scientific, and Medical (ISM) bands | ISM bands are frequency ranges designated by regulatory bodies for use in industrial, scientific, and medical applications. These bands are often used for wireless communication, including devices like Bluetooth and Wi-Fi. |
| Personal Area Network (PANs) | Network that connects devices near an individual, typically within a range of a few meters. |
| Antimicrobial coated | Surface or material that has been treated or coated with substances that inhibit the growth of microorganisms, such as bacteria, fungi, or viruses. |
| Elevate | The term coined by Garmin to describe their technology of photoplethysmography. |
| Pure Pulse | The term coined by Fitbit to describe their technology of photoplethysmography. |

# **Introduction**

Every year 98,000 people die in U.S.A only, due to medical malpractice. One of the most significant causes to medical malpractice is patient’s mis identification. The main hazard came from wrong identification, is giving patients Unnecessary treatment, where in the best scenario expressed in waste of Expensive and valued Personnel which could be allocated better. And at the worst case can cause severe damage to the patient. Moreover, the wrong identification causes large Economic burden for the hospitals due to Lawsuits and expensive Unnecessary treatment.

Today at hospitals, simple bracelets attached to the patient who is hospitalized for identification, which contain a barcode, the name and ID number of the patient. The down sides of this This solution are, medical staff must get very close to the patient to deplete vital information, fact that Puts them at risk of infection (e.g., Covid 19) and sometimes even violence on the part of patients and they companion. also, the bracelet has a negative image on the public and a lot of time patient refused to wear them such that they have no immediate means of identification. In addition to those down sides the details are minimal and can contain more data including medical measurements and medical history.

At hospitals Useful medical measurements taking using medical devices. Some of these devices are transmit the information to the hospital's systems, within the patient's medical file. Some of the simpler measurements, such as pulse and stature, measured when the patient is only in the bed and connected to the measuring instruments near the bed. Also, the measurements do not take passively and there is need of medical staff supervision. In addition, the regular and continuous measurements will usually be performed only on a patient who has a medical need for these measurements, and usually in extreme and life-threatening situations, and not on every patient in the hospital.

As a result, there is no regular and continuous record of metrics per patient. also, there is no warning of a life-threatening condition when the patient is not connected to the medical devices or stay in his bed. A distress button is existing in the hospital and located next to the bed and in the toilet or the shower. the downside stands out when the patient is in an emergency, and a life-saving medical response is needed, the alarm does not activate automatically. The patient is depending on his or his companion to detect the danger and push the distress button.

Due to these reasons, there is a need for a simple solution, which will reduce the level of human involvement in the patient identification process and thus increase patient safety, save costs, and improves the treatment process. In addition, measurements will be made regularly, information will be monitored, analyzed, and transferred to the medical institution's information systems. ‎ [1]

One goal of the project is to develop a smart bracelet which will replace the simple plastic bracelet used today mostly in hospitals, as a sophisticated means of identification that will be efficient and convenient to use and will provide the medical staff with vital and reliable information in an accessible way.

Another goal is to take physiological measurements and location, then analyze the signals with satisfy accuracy.

The last goal is to establish functional and secure communication between the bracelet and the information system including synchronization with the medical staff. including alert option when there is a deviation from the norm of measurements values that will be defined.

The objectives are having the Ability to measure and translate signals from sensors to heart rate, SpO2, and location. Make a bracelet that will be comfortable to wear. Having a reliable communication between bracelet and computer. Creating GUI for control the communication. And creating data base.

The indicators that we will achieve is a bracelet that will measure Heart rate in accuracy of 70% and SpO2 in accuracy of 70%. The medical staff could read the information using dedicated digital device from 1 meter away from bracelet. The bracelet will weigh under 500 grams, suit at least 3 different sizes of hand and provide location with 2-meter accuracy inside the building. Software will notify if any deviation from the norm of several values occurs. When click on GUI data Butten the information will appear clearly on the screen. The data from the sensor will be saved accurately in the data base.

# **Literature Review**

In this section we will examine measurements, and popular methods to measure them from the current literature.

## **Heart Rate:**

Heart rate, or pulse, is the number of times a heart beats per minute. Heart rate varies from person to person. Also, heart rate changes according to physical conditions. Therefore, heart rate varies between ages, and as people age, the heart rate decreases [‎‎2, ‎‎‎3].

The best places at human body to find pulse are the wrists, inside of an elbow, at side of the neck and top of the foot. Resting heart rate is the heart pumping the lowest amount of blood for human needs, and normally heart rate is between 50 beats per minute to 100 beats per minute. Other factors affect heart rate like body size, when meaning to obese, heart rate is usually higher. Also, body position. If resting, sitting or standing, heart rate is varying. When feeling exiting emotions like very happy or sad as well as angry or stressed, heart rate increases. Also, drugs influence heart rate.

## **Oxygen Saturation:**

A blood-oxygen saturation reading indicates the percentage of hemoglobin molecules in the arterial blood which are saturated with oxygen. The reading may be referred to as SaO2. Readings vary from 0 to 100%. Normal readings in a healthy adult, however, range from 92% to 100%.

When levels are right, body is able to deliver oxygen to the muscles and help them to function properly. When they are low, blood does not contain as much oxygen. This may result in dizziness, fatigue and a lack of overall energy.

Normally blood oxygen levels are about 90 to 100 percent oxygen saturation. If they are not this high normally, it could be an ailment that's preventing body from getting enough oxygen.

The term SpO2 means the SaO2 measurement determined by pulse oximetry [‎4].

## **Optic sensor (for measuring HR and PO):**

Based on Photoplethysmography obtained by plethysmogram [‎‎5]. A PPG is often acquired by using a pulse oximeter which illuminates the skin and measures changes in light absorption. Every cardiac cycle pumps blood to the periphery. Every pressure pulse cause change in blood volume, which is calculating by illuminating the skin with light and measuring the amount of light either transmitted or reflected to a photodiode. Every cardiac cycle appears as a peak. The general consensus is that the waveform comes from the site of maximum pulsation within the arteriolar vessels where pulsatile energy is converted to smooth flow just before the level of the capillaries. The source of the waveform is Beer's law of light, which describe the elements that contribute to the pulse oximeter waveform [‎‎6].

Where:

absorption at a given wavelength

extinction coefficient (absorbency)

concentration

path length

## **Overcoming Patient concerns about privacy:**

For the implementation of smart bracelets to be effective there must be cooperation on the part of the patients to wear them. Smart wristbands have recently been assimilated in hospitals in Canada. Despite the potential inherent in this method, there is a risk of leaking confidential medical information of patients. As a result, many patients may object to wearing the bracelets. The aim of the study is to investigate the causes of patients' concerns and suggest solutions so that hospitals can implement them in the system. In way that will address their concerns in the final product and in his implementation process. According to the study the most prominent concern is their privacy. there are several actions that can encourage patients to use bracelets: Accessing an option for the patient to share minimal or extended information to increase the patient's sense of control over their information. Give the patients immediate sense of great value by wearing the bracelet. For example, significant queue shortening etc. Also, there a need of publication of statistics like the decries of mis identification and cases of medical malpractice. side by side to publication of cases that the bracelet saved patient from danger by alarming the medical staff about his signals, or cases of finding missing or kidnapped kid or finding a lost Demented Adult thanks to bracelet. Finally, to improve the security of data by investing on this issue and publication of those action. ]‎‎‎1]

## **Systemic alternatives:**

### **RFID:**

Radio-frequency identification (RFID) uses electromagnetic fields to automatically identify and track tags attached to objects. An RFID system consists of a tiny radio transponder, a radio receiver and transmitter. When triggered by an electromagnetic interrogation pulse from a nearby RFID reader device, the tag transmits digital data, usually an identifying inventory number, back to the reader. Passive tags are powered by energy from the RFID reader's interrogating [radio waves](https://en.wikipedia.org/wiki/Radio_wave). Active tags are powered by a battery and thus can be read at a greater range from the RFID reader, up to hundreds of meters. [‎‎‎7]

### **Bluetooth:**

Bluetooth is a short-range wireless technology standard that is used for exchanging data between fixed and mobile devices over short distances using UHF radio waves in the ISM bands, from 2.402 GHz to 2.48 GHz, and building personal area networks (PANs). In the most widely used mode, transmission power is limited to 2.5 milliwatts, giving it a very short range of up to 10 meters (33 ft). [‎‎‎8]

### **Image processing face recognition:**

Image processing face recognition is a computerized technique that uses an algorithm to locate and recognize a face in an image, and this technology has several uses. While there are many different facial recognition algorithms available, most programs use edge or eye detection to locate a face. Not only does image processing face recognition find a face, but most algorithms also show the user the exact pixel location of the face within the image, making it easier to find a certain face in a crowded or blurred image. This technique is used in facial recognition security, the algorithm is typically specialized in this case to recognize only certain faces. [‎9]

## **Market Survey:**

### **Products at the market:**

The simple bracelets are the most common solution for identification in hospitals. They are chip, fast to print comfortable to wear. Zebra is very common company where More than 382.8M patients globally were identified by Zebra in 2018.there some different type of bracelets differing by the reading technic the most significant are thermal bracelets, laser bracelets and radio frequency bracelets [‎‎10].

### **Zebra ZD510-HC Wristband Printing Solution | ZD510-HC:**

Printer cost: 766.85$ [‎‎10[

The ZD510-HC Wristband Printing solution combines the reliable ZD510-HC direct thermal printer with easy to load cartridges containing the only antimicrobial coated wristbands on the market. You get dependable wristband printing that increases patient safety and staff productivity.

The printer connection possibilities are:  
USB, USB Host, 10/100 Ethernet, Wireless 802.11ac, Bluetooth

The printer capable of printing different sizes of bracelets as describes in Table 31. bracelet sizes [‎‎10].

The market of smart watches and fitness wrists include hundreds of types, various companies, and prices. This market survey includes the leader companies, with the most common models and the best product, for that moment.

### **Apple Watch:**

Now, Apple has launched eight series of watches. Apple Watch series 8 is the newest, was released at 2022[11].

The Apple Watch 8 technical specifications are shown in Figure 3. 1 Apple watch 8 technical specifications [‎12].

As we can see at the technical specifications, Apple Watch includes heart rate sensor. The heart rate sensor which includes at this model is an optic sensor that use the technology of photoplethysmography [‎13]. As described before, those technology uses the light absorption to measure pulse oximeter, as well as heart rate. Blood reflects red light and absorbs green light. Apple Watch uses green LED lights paired with light‑sensitive photodiodes to detect the amount of blood flowing through the wrist. The sensor uses the changes in artery pressure to measure the green light absorption that changes by the flow of the blood, which depends on the heartbeat. By software instructions, the green light flashing hundreds of times per second. The heart rate is measured by calculation of the number of times heart beat per minute.

Furthermore, Apple Watch has a lot of features. It has water resistant, compatibility with IPhone and smart home product, a several of smartphone attributes and apps. Also, Apple Watch is adjusted to fitness market with a unique application e.g. a comprehensive Workouts App that fit to sort of sports and workouts, Activity App that calculate daily and continues activity, Breathing App that helps controlling breathing and relaxing, etc.

Apple Watch price (when declared): from 399$.

### **Garmin Fenix 7:**

The technical specifications are shown in Table 3. 2 GARMIN FENIX 7 PHYSICAL PROPERTIES and the properties are shown in Table 3. 3 GARMIN FENIX 7 PROPERTIES [‎14].

Garmin is an American multinational [technology](https://en.wikipedia.org/wiki/Technology) company, which is known for its specialization in [GPS](https://en.wikipedia.org/wiki/Global_Positioning_System) technology development. The company creates products for automotive, aviation, marine, outdoor, and sports [‎15‎, ‎16].

The Garmin Fenix 7 is a premium model of smart-fitness watch and it’s one of the powerful multisport watches, for that moment. It has a lot of features especially for sport, fitness, outdoor activity and compatibility with smartphones. It has water resistant, a several smartphone attributes and apps, likewise the Apple Watch. In contrast to Apple Watch, which is aimed to the mainstream, the Fenix 7 is aimed more at athletes, and the features with emphasis on GPS, include setting to almost every sport activity [‎‎‎17].

As we can see at the technical specifications, Garmin Fenix 7 includes heart rate sensor. The heart rate sensor which is include at this model is an optic sensor. Likewise, the Apple Watch, Garmin use the technology of photoplethysmography, which Garmin call it as "Elevate" [‎‎‎18]. This sensor samples heart rate with LED lights that illuminate through the skin to measure changes in light as a result in blood flow through vessels and capillaries. Although the sensor uses the same technology, each company has its own algorithm, and Garmin Fenix 7 support wider types of activities, and as described by the company, more accuracy heart rate which is obtained by digital signal processing.

Battery life of the Fenix 7 is up to 18 days averagely, and when GPS sensor is using occasionally.

Garmin Fenix 7 price (when declared): from 699$.

### **Fitbit Charge 5:**

The technical specifications are shown in Table 3.4 CHARGE 5 PROPERTIES Table 3. 5 CHARGE 5 SENSORS [‎19,‎‎‎20]:

Heart Rate:

Like all heart rate tracking technologies, accuracy is affected by physiology, location of device, and different movements.

Battery and Power:

Is shown in Table 3. 6 CHARGE 5 BATTERY PROPERTIES [‎20].

Memory:

Charge 5 stores your data, including daily stats, sleep information, and exercise history, for 7 days. See your historical data in the Fitbit app.

Materials:

The housing on Charge 5 is made of anodized aluminum and surgical-grade stainless steel. The band that comes with Charge 5 is made of a flexible, durable silicone similar to that used in many sports watches. While all stainless steel and anodized aluminum can contain traces of nickel, which can cause an allergic reaction in someone with nickel sensitivity, the amount of nickel in all Fitbit products meets the European Union's stringent Nickel Directive.

Environmental Requirements:

* Operating temperature: 14° to 113° F
* Maximum operating altitude: 28,000 feet

Syncing & Notifications:

Using Bluetooth LE wireless technology, Fitbit Charge 5 sends smartphone notifications and syncs automatically to computers and 200+ mobile devices

Fitbit is an American company which founded in 2007. It’s known for wireless-enabled wearable technology devices, especially for fitness and health [‎21,‎‎‎22].

Fitbit has five series of products that are relevant to the market survey. Even though the smart watch series is the most elaborate and expensive series, as well as other products which described before are smart watches, this survey about Fitbit focuses on fitness wristband. The Fitbit Charge 5 is the most popular and best-selling product by Fitbit.

The Fitbit Charge 5 is a fitness wristband, which has a lot of features and as described by the technical specifications, include heart rate measuring. The heart rate measuring, likewise, the Apple Watch and Garmin Fenix 7, uses the technology of photoplethysmography, which Fitbit call it as "Pure Pulse" [‎20]. 24 Fitbit claims that they were the first who had that heart rate measuring on wrist technology, based on photoplethysmography, and the company have 19 patents awarded for heart rate innovation. To calculate blood flow, "Pure Pulse" shines a green light onto the skin and uses photodiodes to measure how much light is being absorbed. This measurement is used to determine how many times the heart beats per minute.

Likewise, the Apple Watch and Garmin Fenix 7, the Charge 5 use applications to calculate heart rate, fitness level, activity tracking and is comestible to smartphone.

Battery life is up to 7 days. An advantage is that charging of 2 hours is enough for another 7 days.

Fitbit Charge 5 price (when declared): from 179$.

### **Table of comparing:**

Table 3. 6 comparing smart watches properties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Apple Watch** | **Garmin Fenix 7** | **Charge 5** | **Our Wristband** |
| **Price** | 399$ | 699$ | 179$ | 41.6$ |
| **Wight** | 42g | 79g | 28g | 86g |
| **GPS** | V | V | V | V |
| **ECG** | V | V | V | V |
| **Bluetooth** | V | V | V | X |
| **Battery** | 18 hours | 18days | 7days | X |
| **Waterproof** | 5ATM | 10ATM | 5ATM | X |

It can be seen from the survey that existing products are aimed to fitness and health sector. Therefore, most of the products are fashionable, and aimed for compatibility with smartphones. Our product is cheaper than the products on the market, it is important because it ensures affordability and accessibility. A low-cost solution allows for wider adoption and implementation, particularly in healthcare settings with limited resources or budget constraints. By being cost-effective, more healthcare facilities and individuals can benefit from the device, promoting its widespread use and enabling better patient care across different healthcare environments.

# **Methods**

## **Block diagram:**

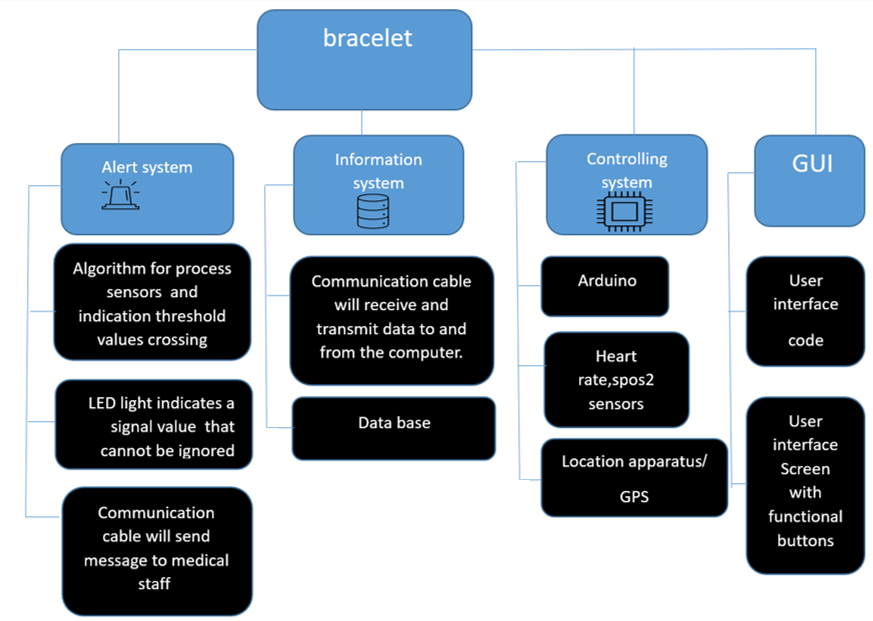


Figure 4.1 bracelet block diagram

## **Flow chart:**

1. The medical staff member request information about the patient through a user interface.
2. The data transferred from to the bracelet to the computer through serial communication cable.
3. The algorithm analyzes the data from the sensors and store it at the data base.
4. If the sensor values are beyond the threshold the led from the bracelet will flesh at fast pace as an alert, danger warning message will appear on the screen.
5. The requested data appears on the computer screen.

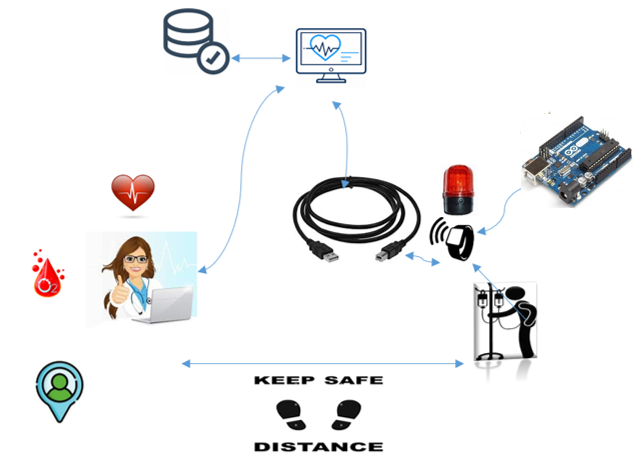


Figure 4.2 flow chart

## **Required Means and Tools:**

### **Study and research:**

* Sensor based on photoplethysmography for heart rate and spo2.
* Signal processing methods.
* Safe communication protocol.
* Transmitter choosing.
* Closing Software gaps.

#### **Tools:**

### Arduino Uno:

Arduino Uno is a microcontroller board based on the ATmega328P ([datasheet](http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf)). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. [‎23]

A picture containing text, electronic engineering, circuit, electronic component

Description automatically generated

Figure 4.1 Arduino UNO components and Pinout

Explanation of components and pinouts which are shown in Figure 4.3:

* ATmega328 - combines Memory (SRAM, EEPROM, and Flash), Analog to Digital Converter, SPI serial ports, I/O lines, registers, timer, external and internal interrupts, and oscillator.
* ICSP pin - allows the user to program using the firmware of the Arduino board.
* Power LED Indicator- The ON status of LED shows the power is activated. When the power is OFF, the LED will not light up.
* Digital I/O pins- The digital pins have the value HIGH or LOW. The pins numbered from D0 to D13 are digital pins.
* TX and RX LED's- The successful flow of data is represented by the lighting of these LED's.
* AREF- The Analog Reference (AREF) pin is used to feed a reference voltage to the Arduino UNO board from the external power supply.
* Reset button- It is used to add a Reset button to the connection.
* USB- It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.
* Crystal Oscillator- The Crystal oscillator has a frequency of 16MHz.
* Voltage Regulator- The voltage regulator converts the input voltage to 5V.
* GND- Ground pins. The ground pin acts as a pin with zero voltage.
* Vin- It is the input voltage.
* Analog Pins- The pins numbered from A0 to A5 are analog pins. The function of Analog pins is to read the analog sensor used in the connection. It can also act as GPIO (General Purpose Input Output) pins. [ ‎24]

### GPS component- GY-GPS6MV2:

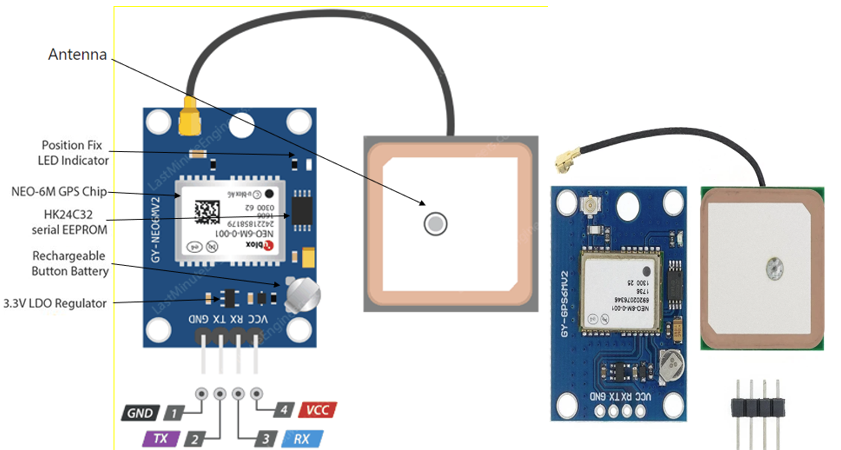


FIGURE 4. 2 GPS component- GY-GPS6MV2

Explanation of components and pinouts which are shown in FIGURE 4. 7:

* NEO-6M GPS Chip- can track up to 22 satellites over 50 channels and achieve tracking sensitivity of i.e. -161 dB, while consuming 45 mA current. performs 5 location updates in a second with 2.5m horizontal position accuracy.
* Antenna-The module comes with -161 dBm sensitivity patch antenna for receiving radio signals from GPS satellites.
* Position Fix LED Indicator- There is a LED on the module that indicates the status of the ‘Position Fix’. It will blink at different rates depending on which state it is in.
* EEPROM- The module is equipped with HK24C32 Two Wire Serial EEPROM which is used to store small amounts of data. It is 4KB in size and is connected via I2C to the NEO-6M chip.
* rechargeable button battery- The battery charges automatically when power is supplied to the module and retains data for two weeks without power.
* 3.3V LDO Regulator- used to convert a higher input voltage to a lower, more stable output voltage.
* GND- is the ground pin and needs to be connected to the GND pin on the Arduino.
* TxD- (Transmitter) pin is used for serial communication.
* RxD -(Receiver) pin is used for serial communication.
* VCC- supplies power to the module. Can be connected directly to the 5V pin on the Arduino [‎‎25]

The connection to the Arduino is presented in FIGURE 4. 3 GPS

A circuit board with wires

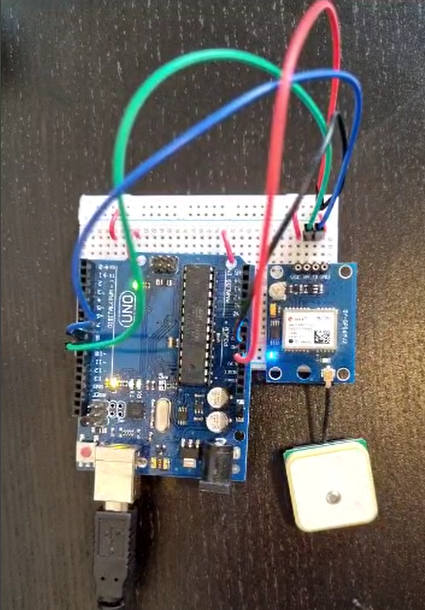
Description automatically generated with low confidence

FIGURE 4. 3 GPS connections to Arduino

### MAX30102 - Heart Rate and SpO2 sensor:

The MAX30102 pulse oximeter and heart rate sensor is an I2C-based low-power plug-and-play biometric sensor. The sensor integrates pulse oximeter and heart rate sensor IC, from Analog Devices. It combines two LEDs, a photodetector, optimized optics, and low-noise analog signal processing to detect pulse oximetry (SpO2) and heart rate (HR) signals. [26, 27]

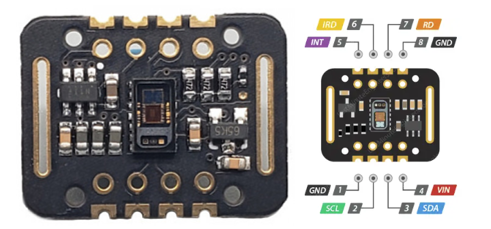


Figure 4. 4 Heart Rate and SpO2 sensor- MAX30102

Explanation of pinouts which are shown in Figure 4. 4:

* VIN- is the power pin. Can be connected to 3.3V or 5V output from your Arduino.
* SCL- is the I2C clock pin, connected to Arduino’s I2C clock line.
* SDA- is the I2C data pin, connected to Arduino’s I2C data line.
* INT- The MAX30102 can be programmed to generate an interrupt for each pulse. This line is open-drain, so it is pulled HIGH by the onboard resistor. When an interrupt occurs the INT pin goes LOW and stays LOW until the interrupt is cleared.
* IRD- The MAX30102 integrates an LED driver to drive LED pulses for SpO2 and HR measurements.
* RD- pin is similar to the IRD pin, but is used to drive the Red LED.
* GND- is the ground.

Explanation of components:

components which are shown in Figure 4. 5:

* These LEDs shine light through the skin and the sensor measures the amount of light that is absorbed or reflected back by the blood vessels.
* RED LED- emits light in the red spectrum that is more readily absorbed by oxygenated blood.
* IR LED -emits light in the infrared spectrum that is more readily absorbed by deoxygenated blood.
* Photodetector -measure the amount of light that is transmitted or reflected back from the red and infrared (IR) LEDs after passing through the skin. The photodetector detects the intensity of the light at each wavelength and converts it into an electrical signal

A picture containing text, circuit, screenshot, electronic engineering

Description automatically generated

Figure 4. 5 RED LED, IR LED and Photodetector

components which are shown in Figure 4. 6:

* Power-two different supply voltages: 1.8V for the IC and 3.3V for the RED and IR LEDs.

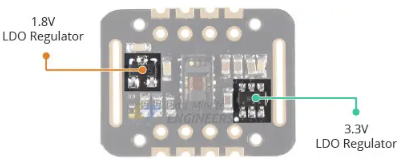


Figure 4. 6 power supply

components which are shown in Figure 4. 7 the opposite side:

* Logic level selection jumper- On the back of the PCB , used to select between 3.3V and 1.8V logic level.

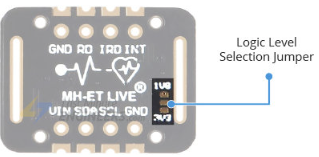


Figure 4. 7 Logic level selection jumper

The connection to the Arduino is presented in Figure 4. 8

A circuit board with wires

Description automatically generated with low confidenceA circuit board with wires

Description automatically generated with low confidence

Figure 4. 8 MAX30102 connections to Arduino

### LED:

LED is a small electronic device that emits light when an electric current passes through it. It is made of a semiconductor material with a p-n junction. LEDs come in different colors and are energy-efficient and durable. They can be controlled to create various lighting effects. [‎28]

The connection to the Arduino is presented in FIGURE 4. 9

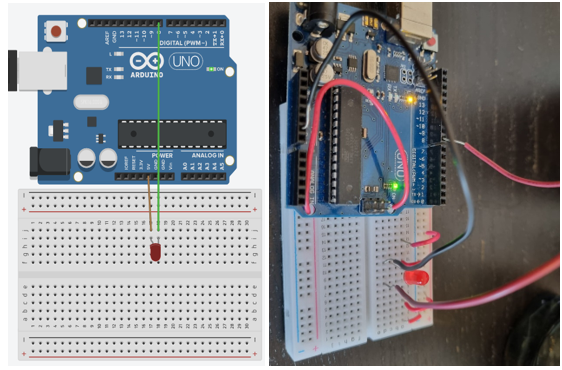


FIGURE 4. 9 LED connection to Arduino

#### Table of components**:**

Table 4.1 project components prices updated

|  |  |  |  |
| --- | --- | --- | --- |
| TOOLS | Cost | Weight | Purpose |
| Arduino Uno | 80 ILS | 25.9g | Control unit. store and send data |
| Breadboard (protoboard) | 8 ILS | 36.4g | solderless circuit board for prototyping circuits. |
| GPS component  GY-GPS6MV2 | 6 ILS | 16.5g | To measure the location of the patient |
| Electronics components (wires, resistors, LED, PCB board etc.) | 60 ILS | 5-6.3g | Tools to build circuit |
| MAX30102 | 9 ILS | 0.9g | Measure heart rate and oxygen level |
| Total | 155 ILS | 86g | - |

The total weight is presented in Figure 4.10

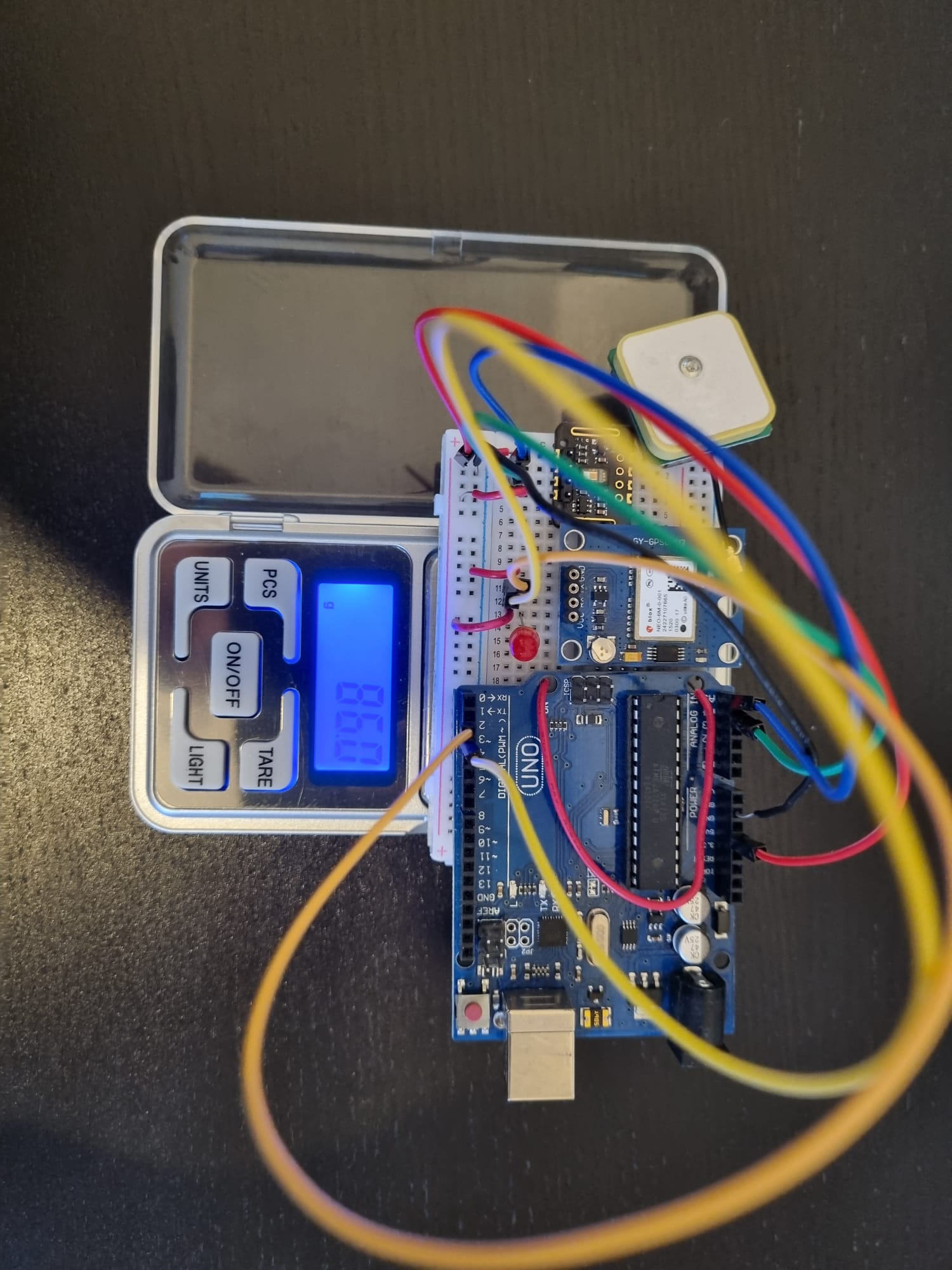


Figure 4. 10 total weight of components

Calculations:

We preformed calculations in our code.

First we have calculated the maximum average heart rate and minimum average SpO2 values from a given set of heart rate and SpO2 data over in 5 samples in a row for getting more accurate values before warning the medical staff .It then generates warnings based on certain conditions, such as a high heart rate or low SpO2, and present them to the user. In this code it can be observed :

max\_heart\_rate\_avg = 0  
min\_spo2\_avg = 0  
for i in range(len(rows)):  
 if i >= 4:  
 heart\_rate\_avg = sum(heart\_rate\_values[i - 4:i + 1]) / 5  
 spo2\_avg = sum(spo2\_values[i - 4:i + 1]) / 5  
 if heart\_rate\_avg > 90 and heart\_rate\_avg > max\_heart\_rate\_avg:  
 max\_heart\_rate\_avg = heart\_rate\_avg  
 if spo2\_avg < 90 and spo2\_avg > min\_spo2\_avg:  
 min\_spo2\_avg = spo2\_avg  
warnings = []  
if max\_heart\_rate\_avg > 0:  
 warnings.append(f"Patient heart rate is high! Maximum average: {max\_heart\_rate\_avg:.2f}")  
if min\_spo2\_avg > 0:  
 warnings.append(f"Patient SpO2 is low! Maximum average: {min\_spo2\_avg:.2f}")  
  
root = tk.Tk()  
root.withdraw()

for the location we are using calculation that use gps latitude and longtitude points for determine the distance over two points on earth using kilometer mertric. We assume the earth radious. The calculation made in this section :

def calculate\_distance(lat1, lon1, lat2, lon2):  
 # approximate radius of Earth in km  
 R = 6371.0  
  
 lat1\_rad = radians(lat1)  
 lon1\_rad = radians(lon1)  
 lat2\_rad = radians(lat2)  
 lon2\_rad = radians(lon2)  
  
 dlon = lon2\_rad - lon1\_rad  
 dlat = lat2\_rad - lat1\_rad  
  
 a = sin(dlat / 2) \*\* 2 + cos(lat1\_rad) \* cos(lat2\_rad) \* sin(dlon / 2) \*\* 2  
 c = 2 \* atan2(sqrt(a), sqrt(1 - a))  
  
 distance = R \* c  
 return distance

#### Free Software:

* Python
* C (we have used that in arduino)
* Arduino software (IDE)
* CVI (finely we didn’t use that)
* SQL for data bases (we have used sqlite3 inside python)
* Tinkercad for visualized circuits

# **Results**

## **Information system**

The code creates 2 databases using sqlite3, patient general information and patient measurements, and tkinter library for user interface. there are functions and classes of each action available in the gui.

The full code for the information system can be viewed at Table 5. 1 system codes

### **The home page**

Presents all the windows option we have in the system as shown in Figure 5. 1

Chart

Description automatically generated with medium confidence

Figure 5. 1 home page

### **Patient administration**

Presents all the Patient information management as shown in Figure 5. 2

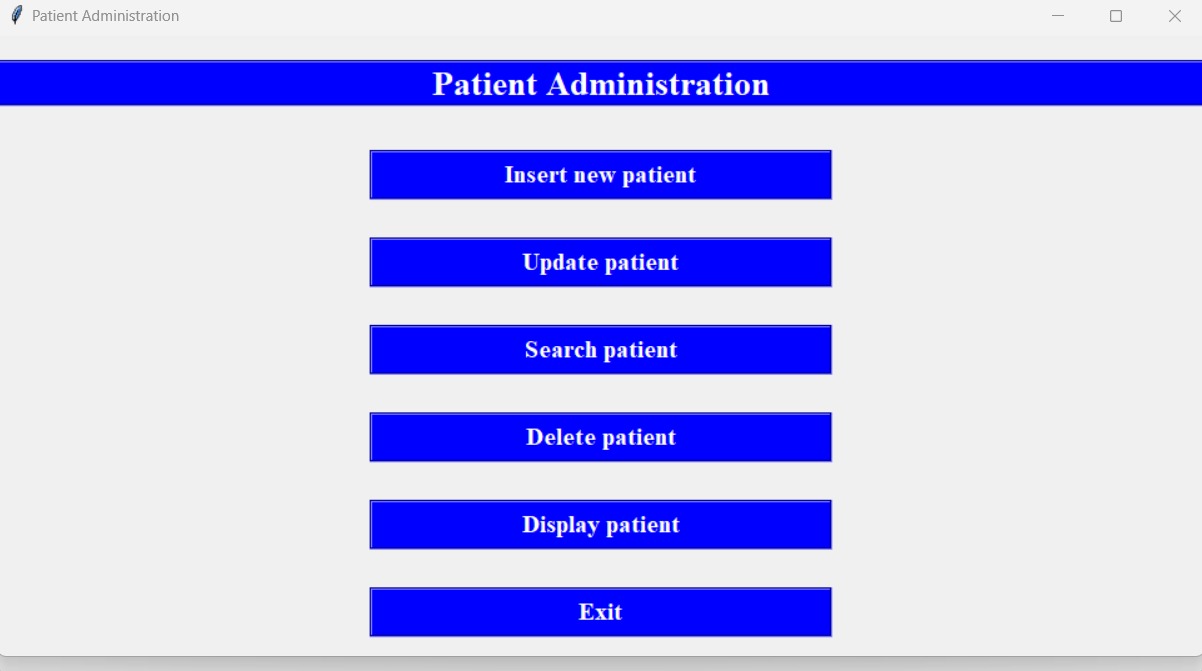


Figure 5.2 Patient administration

### **Patients insert window**

Allows us to add a new patient and include his basick data as shown in Figure 5. 3

Graphical user interface

Description automatically generated with medium confidence

Figure 5. 3 insert window

### **Update patient data by ID**

Allows us to find the patient by his ID and update the information if there has been a change since it was entered as shown in Figure 5. 4

A screenshot of a computer error

Description automatically generated with low confidence

Figure 5. 4 Update patient data by ID

### **Update patient data**

Allows us to update the information if there has been a change since it was entered as shown in Figure 5. 5

Graphical user interface, application

Description automatically generated

Figure 5. 5 update data

### **Search patient data**

Allows us to find a specific patient by entering his ID number to the search box as shown in Figure 5. 6

Graphical user interface, application

Description automatically generated

Figure 5. 6 search data

### **Delete patient**

Allows us to delete a specific patient by entering his ID number to the search box as shown in Figure 5. 7

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5. 7 delete patient

### **Patient database view**

Allows us to see a specific patient data base as shown in Figure 5. 8

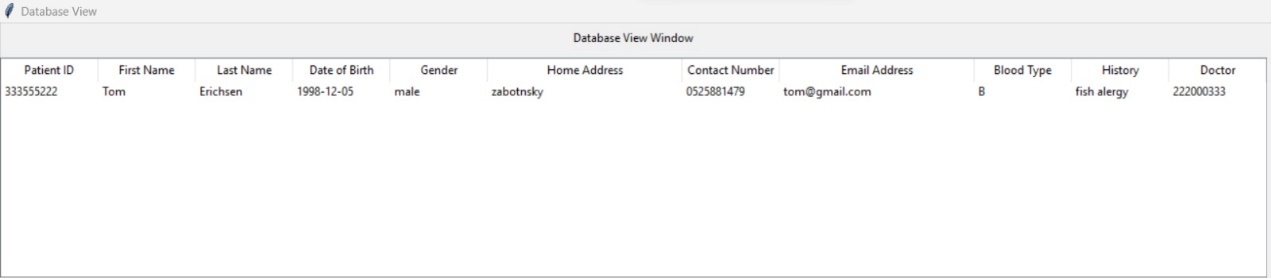


Figure 5. 8 patient database view

### **Doctor administration**

Presents all the doctor information management as shown in Figure 5. 9

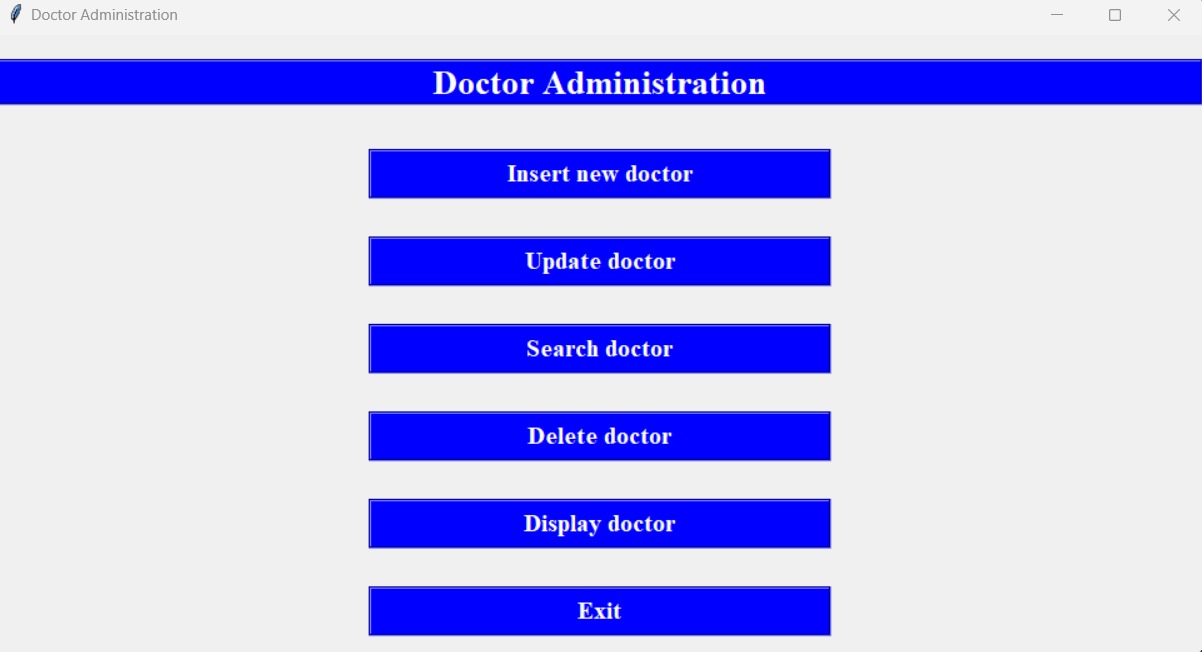


Figure 5. 9 doctor administration

### **Doctor Insert window**

Allows us to add a new doctor and include his basick data as shown in Figure 5. 10

A screenshot of a computer

Description automatically generated

Figure 5. 10 doctor insert window

### **Update doctor data by ID**

Allows us to find the doctor by his ID and update the information if there has been a change since it was entered as shown in Figure 5. 11

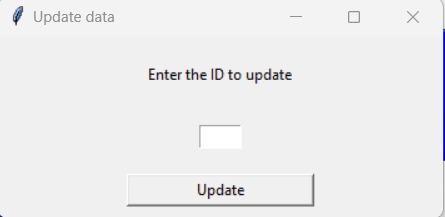


Figure 5. 11 update doctor data by ID

### **Delete doctor**

Allows us to delete a specific doctor by entering his ID number to the search box as shown in Figure 5. 12

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5. 12 delete doctor

### **Doctor database view**

Allows us to see a specific doctor data base as shown in Figure 5. 13

A screenshot of a computer

Description automatically generated

Figure 5. 13 doctor database view

### **Patient measurement's view**

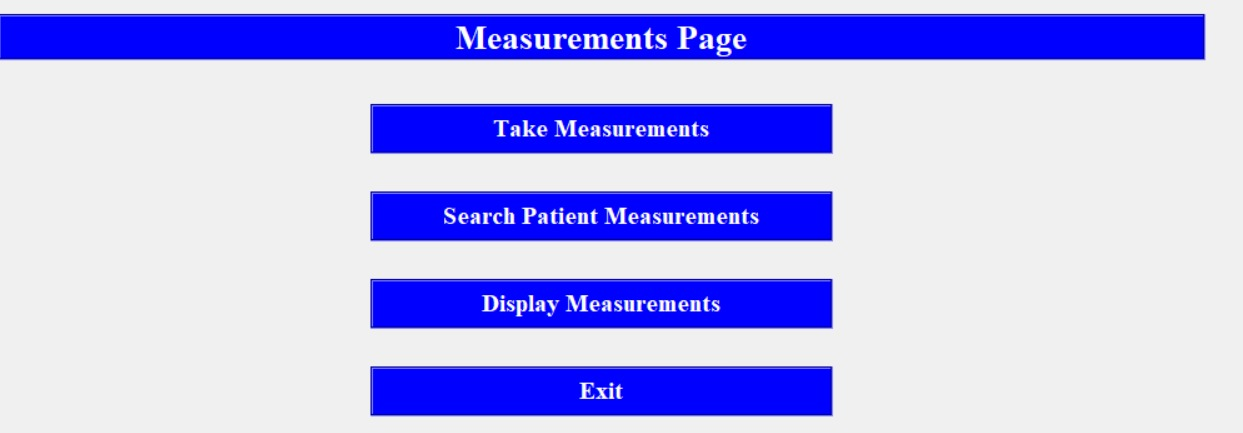
Allows us to see the patient measurements and information management allowing as to present and control the data, take measurements take new measure from Arduino uno and insert into the data base. Search shows us the specific patient previous measurements and displays lets us to display the measurements in graph and on the map. the data in clear way as shown in Figure 5. 14

Figure 5. 14 patient measurements view

### **Patient display view**

the window provides us the choice of which patient to choose by is id and then to decide what to present the patient last locations or the patient spo2 and heart rate graph through time.in figure 5.15

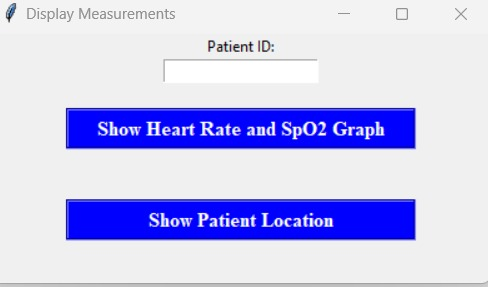


Figure 5.15 display window

### **ERD**

Flow chart of the database structure is shown in Figure 5. 15

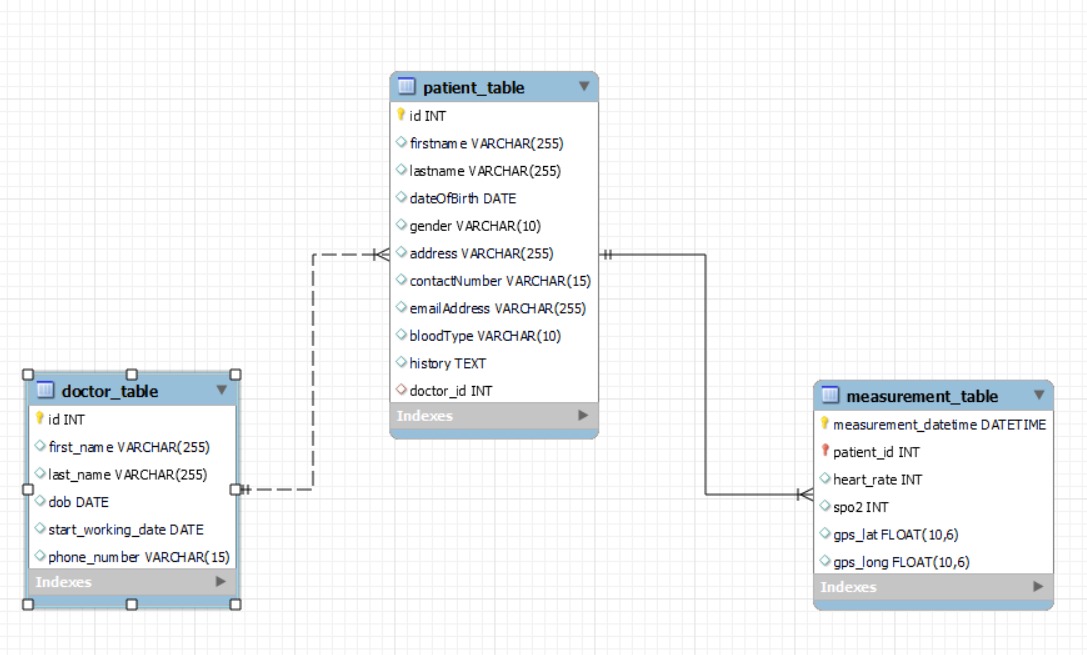


Figure 5. 15 ERD

## **Measurements**

The full code for the measurement tools can be viewed at Table 5.1 system codes

### **Heart rate**

The heart rate sensor functions and gives a good result when it's steady, it is still needed to be connected to the bracelet and checked again.

The measurements are shown in Figure 5.16, Figure 5.17, Figure 5.18, Figure 5.19

A picture containing text

Description automatically generated

Figure 5.16 heart rate

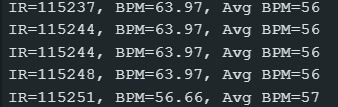


Figure 5.17 heart RATE AVERAGE

Figure 5.18 heart RATE NO finger

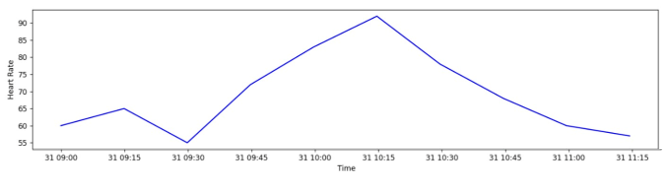


Figure 5.19 heart rate graph

### **GPS**

The GPS sensor gives an incorrect location and will be replaced with a different chip if won’t be improved.

The latitude and longitude measurements are shown in figure 5.20 and the location as found on google maps in figure 5.21, the location and time stamp on the Information system is presented in figure 5.22

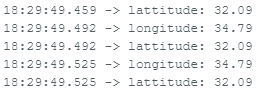


figure 5.20 gps measurments

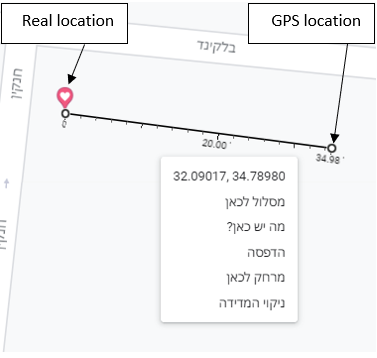


figure 5.21 gps measurements on google maps

A map of a city

Description automatically generated with low confidence

figure 5.22 location on Information system

### **Saturation**

The saturation sensor functions and gives a good result when it's steady, it still needed to be connected to the bracelet and checked again.

The measurements are shown in figure 5.23 and figure 5.24

A picture containing text, font, screenshot

Description automatically generated

figure 5.23 spo2 + heart rate

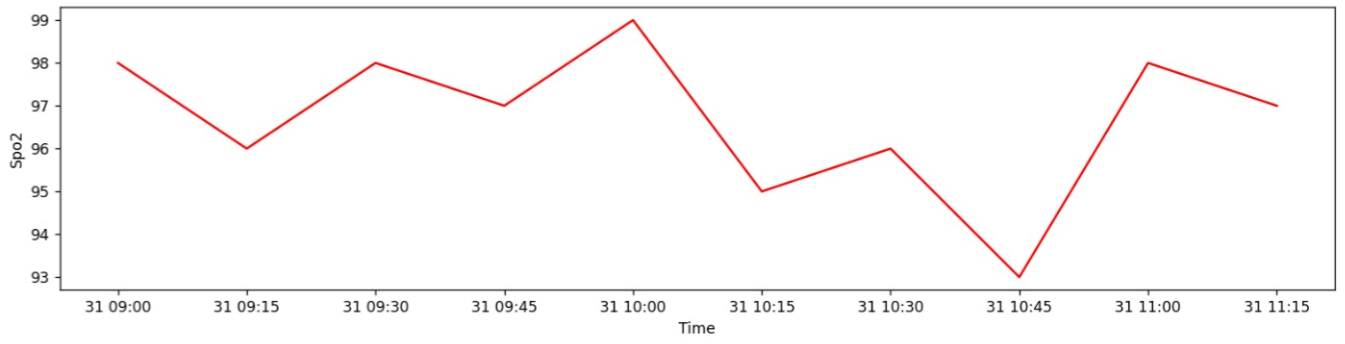


figure 5.24 spo2 graph

### **Alert system**

Communication cable will send message to medical staff when the heart rate is high or the SpO2 is low as presented in figure 5.25

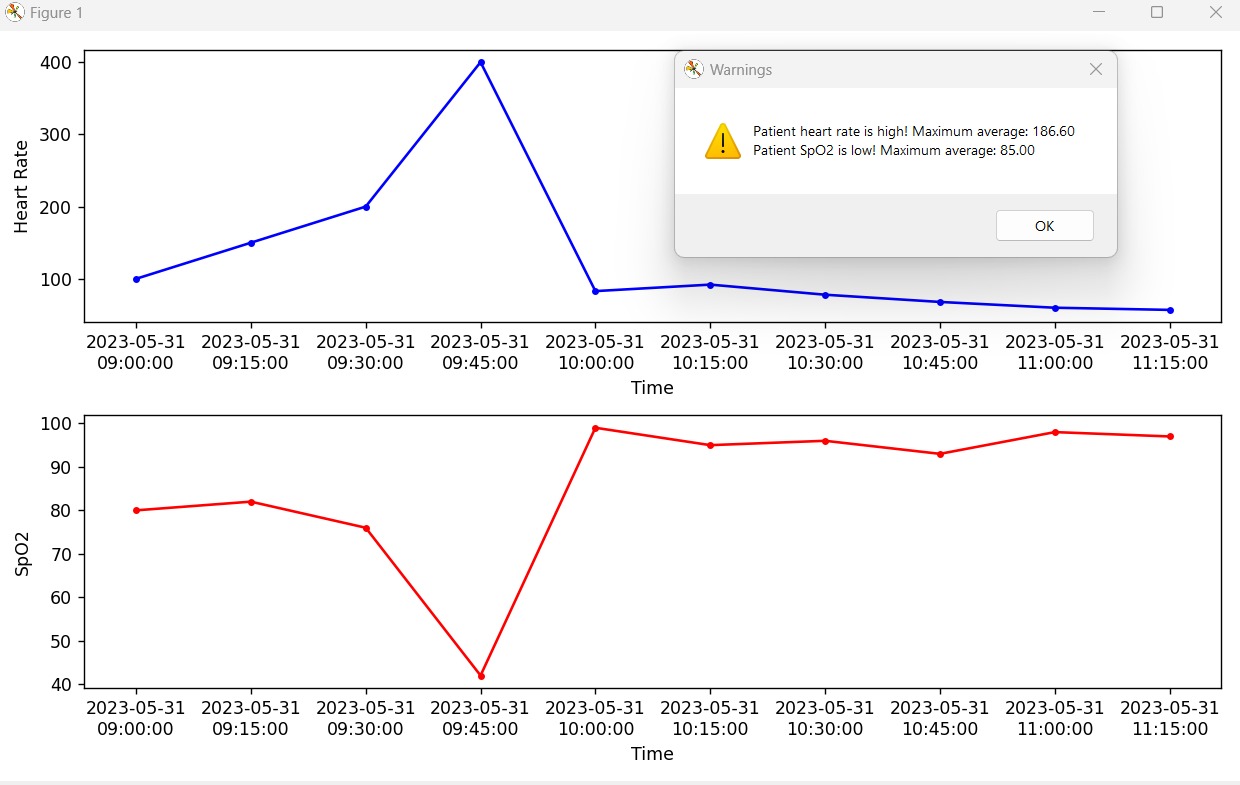


figure 5.25 heart rate or Spo2 warning



figure 5.26 Patient too far away warning

While the GPS, HR or SpO2 sends the warning with the communication cable to the stuff the red LED flickers on the braclet, 1 flick per second, as shown in figure 5.27

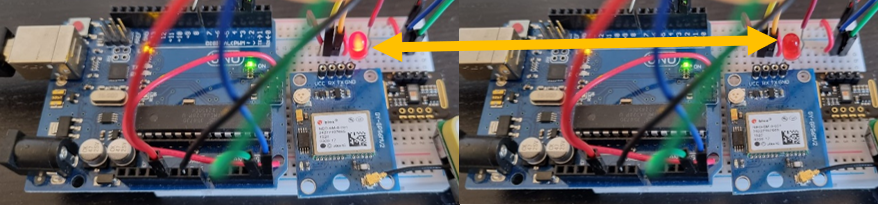


figure 5.27 LED flickering

# **Discussion**

At the results of the project, we have been able to get the patient measurement including location, heart rate and oxygen saturation. The accuracy is well in the spo2 and oxygen values and the location us a bit less accurate. The results are not stable enough all the time. For better results in the future, better quality sensors that will be able to perform in a more stable way with better accuracy. We establish a bracelet that have different size for different people.

The failures points of the project were the failure of the hardware to work well. Arduino micro we have been using it for the brain of the bracelet and it stop working with two different Arduinos, we delt with it using Arduino uno, more reliable and understood better duo to more information and connection to it in the internet the tradeoff is the fact that the Arduino uno is bigger and heavier. The heart rate and spo2 models didn’t work well and we were not able to get stable read, so we have purchased more types and choose the best ones.

# **Summary and Conclusions**

The conclusion from the project is that the bracelet could be very effective for using in hospitals, by make the everyday routine more comfortable for the hospital staff by using more digitalis way to identify patients and get essential measurements from them.

We had planned to use a small Arduino, but it turns out to be not stable and didn’t working in a continuous way. The work was not organized in the best way since the project doesn’t have hospital as partner to guide us and support as to create realistic work that compatible to day by day needs of the patient and the medical staff .as well as a team, we could organized ourself better. For the future we will organize better the missions as a team and our timeline of missions will be organized better.

For future work we suggest using better sensors, more accurate and faster to produce more stable, reliable results for a longer period. As well we are suggesting using a better Arduino board that will be smaller and more reliable for a better comfortable user experience. The last suggestion is to use more than one bracelet and check the system under pressure.

# **References**

* 1. Ksenia Sergueeva and Norman Shaw Ryerson University (2016), "Wearable Technology in Hospitals Overcoming Patient Concerns About Privacy", ResearchGate GmbH, <https://www.researchgate.net/publication/304344244_Wearable_Technology_in_Hospitals_Overcoming_Patient_Concerns_About_Privacy> . [retrieved at 01/06/2023](https://www.researchgate.net/publication/304344244_Wearable_Technology_in_Hospitals_Overcoming_Patient_Concerns_About_Privacy.%20retrieved%20at%2001/06/2022).
  2. ©2023 American Heart Association (2021), target heart raters chart, American Heart Association, <https://www.heart.org/en/healthy-living/fitness/fitness-basics/target-heart-rates> . retrieved on 01/06/2023.
  3. ©2023 American Heart Association (2023), all about heart rate (pulse), American Heart Association, <https://www.heart.org/en/health-topics/high-blood-pressure/the-facts-about-high-blood-pressure/all-about-heart-rate-pulse> . retrieved on 01/06/2023.
  4. Tremper, Kevin K. "Pulse oximetry." *Chest Journal* 95.4 (1989): 713-715.‏
  5. Wikipedia® (2023), Plethysmograph, Wikipedia official website, <https://en.wikipedia.org/wiki/Plethysmograph>. retrieved on 01/06/2023.
  6. Shelley, Kirk H. "Photoplethysmography: beyond the calculation of arterial oxygen saturation and heart rate." *Anesthesia & Analgesia* 105.6 (2007): S31-S36.‏
  7. Wikipedia® (2023), radio-frequency identification, Wikipedia official website, <https://en.wikipedia.org/wiki/Radio-frequency_identification> . retrieved on 01/06/2023.
  8. Wikipedia® (2023), Bluetooth, Wikipedia official website, <https://en.wikipedia.org/wiki/Bluetooth> . retrieved on 01/06/2023.
  9. Alex Newth (2023), " What Is Image Processing Face Recognition?", easytechjunkie, <https://www.easytechjunkie.com/what-is-image-processing-face-recognition.htm> . retrieved on 01/06/2023.
  10. © 2011 - 2023 microless.com (2023), Zebra ZD510-HC Wristband Printing Solution, mircoless website, <https://global.microless.com/product/zebra-zd510-hc-wristband-printing-solution-zd510-hc/?currency=usd>. retrieved on 01/06/2023.
  11. Copyright © 2023 Apple Inc (2023), Apple watch 8 information, Apple official website, <https://www.apple.com/apple-watch-series-8/> . retrieved on 01/06/2023.
  12. Copyright © 2023 Apple Inc (2023), Apple watch 8 Technical Specifications, Apple official website, <https://support.apple.com/kb/SP878?locale=en_US> . retrieved on 01/06/2023.
  13. Copyright © 2023 Apple Inc (2022), Apple watch Support, Apple official website, <https://support.apple.com/en-us/HT204666>. retrieved on 01/06/2023.
  14. Copyright © 1996-2023 Garmin Ltd. (2023), GARMIN Fenix 7 specs, GARMIN official website <https://www.garmin.com/en-US/p/735611#specs>. retrieved on 01/06/2023.
  15. Copyright © 1996-2023 Garmin Ltd. (2023), about GARMIN, GARMIN official website <http://www.garmin.com/en-US/company/about>. retrieved on 01/06/2023.
  16. Wikipedia® (2023), Garmin, Wikipedia official website, <https://en.wikipedia.org/wiki/Garmin>. retrieved on 01/06/2023.
  17. Copyright © 1996-2023 Garmin Ltd. or its subsidiaries (2023), GARMIN Fenix 7 overview, GARMIN official website <https://www.garmin.com/en-US/p/735611>. retrieved on 01/06/2023.
  18. Copyright © 1996-2023 Garmin Ltd. or its subsidiaries (2023), GARMIN Technology, GARMIN official website <https://www.garmin.com.sg/minisite/garmin-technology/wearable-science/heart-rate/>. retrieved on 01/06/2023.
  19. © 2023 Smartwatch Charts (2023), Fitbit Charge 5 Full Specifications, smartwatch graph official website, <https://smartwatchchart.com/smartwatches/fitbit-charge-5-specifications-features-and-price/> . retrieved on 01/06/2023.
  20. ©2023 Fitbit LLC. (2023), Fitbit charge 5 manual, Fibit official website, <https://help.fitbit.com/manuals/manual_charge_5_en_US.pdf>. retrieved on 01/06/2023.
  21. Wikipedia® (2023), Fibit, Wikipedia official website, <https://en.wikipedia.org/wiki/Fitbit>. retrieved on 01/06/2023.
  22. ©2021 Fitbit LLC. (2021), about Fibit, Fibit official website, <https://canarywww.fitbit.com/ph/about> . retrieved on 01/06/2023.
  23. Unknown (Unknown), Arduino UNO, JavaTpoint official website, <https://www.javatpoint.com/arduino-uno> . retrieved on 01/06/2023.

* 1. [© 2021 Arduino S.r.l.](https://www.arduino.cc/en/Main/CopyrightNotice" \t "_blank) (2021), Arduino UNO, Arduino official website, <https://store-usa.arduino.cc/products/arduino-uno-rev3?selectedStore=us> . retrieved on 01/06/2023.
  2. Copyright © 2022 LastMinuteEngineers.com (2022), Interface ublox NEO-6M GPS Module with Arduino, Last minute engineers website, <https://lastminuteengineers.com/neo6m-gps-arduino-tutorial/>. retrieved on 01/06/2023.
  3. [© 2023 Arduino](https://www.arduino.cc/en/Main/CopyrightNotice) (2023), Measure Heart Rate and SpO2 with MAX30102, Arduino official website, <https://projecthub.arduino.cc/SurtrTech/eb4f74c6-bb73-4148-9aaf-9cf1363e5c6d> retrieved on 01/06/2023.
  4. Copyright © 2023 LastMinuteEngineers.com (2023), Interfacing MAX30102 Sensor with Arduino. Last minute engineers website, <https://lastminuteengineers.com/max30102-pulse-oximeter-heart-rate-sensor-arduino-tutorial/> retrieved on 01/06/2023.

* 1. [Scott Campbell](https://www.circuitbasics.com/author/circuitbasicsgmail-com/" \o "Posts by Scott Campbell) (2015), GETTING STARTED WITH THE ARDUINO – CONTROLLING LEDS (PART 1), circuit basics website, <https://www.circuitbasics.com/arduino-basics-controlling-led/> retrieved on 01/06/2023.

# **Appendices**

## **Gannt chart:**

A screenshot of a calendar

Description automatically generated with medium confidence

A screenshot of a calendar

Description automatically generated with medium confidence

FIGURE 10. 1 Gannt chart

## **Changes made during the project:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| original | the certifier | date | change | Cause |
| External client project | Gil Yardeni | 09.05.2022 | Personal project | The external client has updated that the project is no longer relevant |
| Arduino Micro | Dr. Nadav Sheffer | 19.04.2023 | Arduino uno | More reliable working better |
| SQLite | Dr. Nadav Sheffer | 19.04.2023 | MySQL | The software was learned at courses and the developer feels more comfortable with it. MySQL have more function such as ERD and better arrangement |
| Heart rate Hw-827  /  SparkFun sensor | Dr. Nadav Sheffer | 19.04.2023 | Max 30102 sensor | Measure heart rate and oxygen saturation at the same time and working well |
| Memory card | Dr. Nadav Sheffer | 07.05.2023 | Not using a memory card | technical problems, and schedule |
| Battery | Dr. Nadav Sheffer | 07.05.2023 | Not using a battery | technical problems, and schedule |

## **The project poster:**

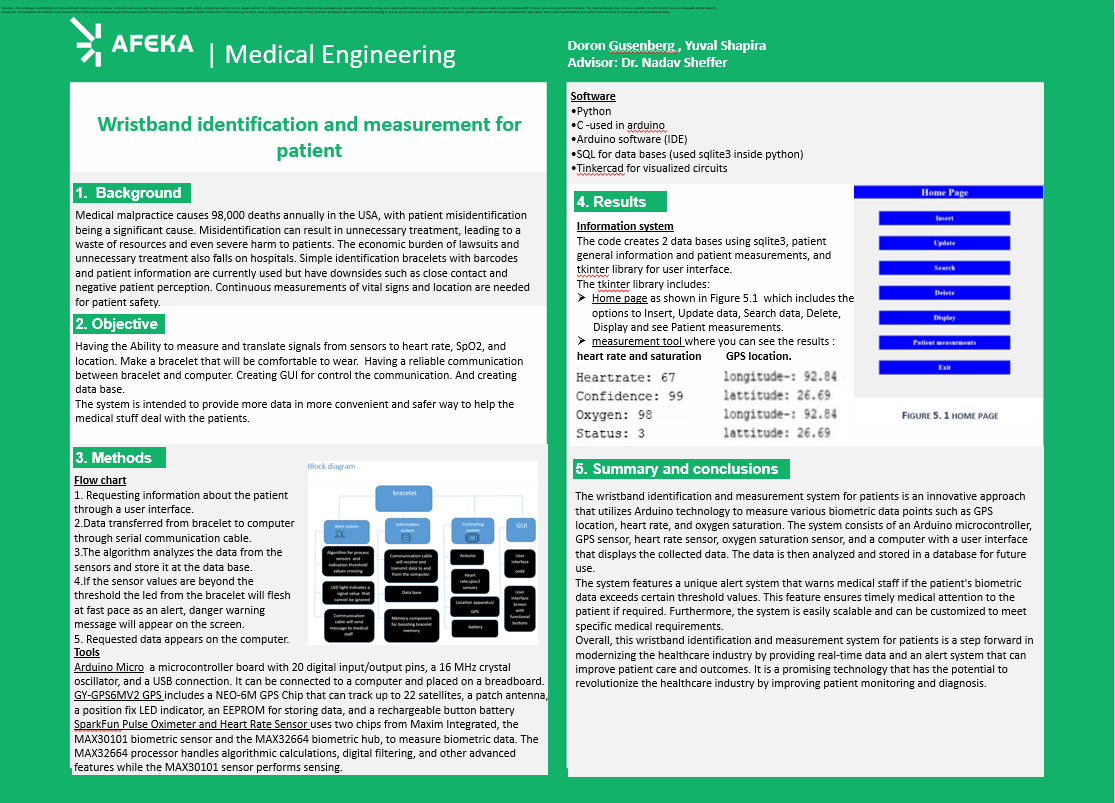


FIGURE 10. 2 project poster

## **Scientific journal article:**

Ksenia Sergueeva and Norman Shaw Ryerson University (2016), "Wearable Technology in Hospitals Overcoming Patient Concerns About Privacy", ResearchGate GmbH, <https://www.researchgate.net/publication/304344244_Wearable_Technology_in_Hospitals_Overcoming_Patient_Concerns_About_Privacy> . [retrieved at 01/06/2023](https://www.researchgate.net/publication/304344244_Wearable_Technology_in_Hospitals_Overcoming_Patient_Concerns_About_Privacy.%20retrieved%20at%2001/06/2022).

## **Other information:**

### **Tables, Figures and Code:**

Table 31. bracelet sizes

Table

Description automatically generated

Table

Description automatically generated

Figure 3. 1 Apple watch 8 technical specifications

Table 3. 2 GARMIN FENIX 7 PHYSICAL PROPERTIES

|  |  |
| --- | --- |
| General | |
| LENS MATERIAL | Corning® Gorilla® Glass |
| BEZEL MATERIAL | stainless steel |
| CASE MATERIAL | fiber-reinforced polymer with metal rear cover |
| QUICKFIT™ WATCH BAND COMPATIBLE | included (22 mm) |
| STRAP MATERIAL | Silicone |
| PHYSICAL SIZE | 47 x 47 x 14.5 mm  Fits wrists with the following circumference: Silicone band: 125-208 mm Leather Band: 132-210 mm Fabric Band: 132-210 mm Metal Band: 132-215 mm |
| TOUCHSCREEN | V |
| COLOR DISPLAY | V |
| DISPLAY SIZE | 1.3” (33.02 mm) diameter |
| DISPLAY RESOLUTION | 260 x 260 pixels |
| DISPLAY TYPE | sunlight-visible, transflective memory-in-pixel (MIP) |
| WEIGHT | 79 g (case only: 56 g) |
| BATTERY LIFE | Smartwatch: Up to 18 days Battery Saver Watch Mode: Up to 57 days GPS Only: Up to 57 hours All Satellite Systems: Up to 40 hours All Satellite Systems + Music: Up to 10 hours Max Battery GPS: Up to 136 hours Expedition GPS: Up to 40 days |
| [WATER RATING](http://www.garmin.com/waterrating/) | 10 ATM |
| MEMORY/HISTORY | 16 GB |

Table 3. 3 GARMIN FENIX 7 PROPERTIES

|  |  |  |
| --- | --- | --- |
| Sensors | | |
| GPS | | V |
| GLONASS | | V |
| GALILEO | | V |
| GARMIN ELEVATE™ WRIST HEART RATE MONITOR | | V |
| BAROMETRIC ALTIMETER | | V |
| COMPASS | | V |
| GYROSCOPE | | V |
| ACCELEROMETER | | V |
| THERMOMETER | | V |
| PULSE OX BLOOD OXYGEN SATURATION MONITOR | | V |
| Smartphone compatibility | iPhone®, Android™, | |

Table 3.4 CHARGE 5 PROPERTIES

|  |  |
| --- | --- |
| Display size | 1.04 inches |
| Display type | 450nits, AMOLED |
| Resolution | 0.86" x 0.58"px |
| Pixel density | 326 ppi |
| Customizable watchface | Yes |
| Input type | Touchscreen |
| Strap size | 140 – 180mm (small), 180 -220mm (big) |
| Screen protection | [Corning Gorilla Glass](https://smartwatchgraph.com/screen-protection/corning-gorilla-glass/) |
| Dimension | 36.7 x 22.7 x 11.2 mm |
| Weight | 28g |
| Waterproof rating | Yes, 5ATM |
| Bluetooth | Bluetooth v4.0 |
| Wi-Fi | No |
| NFC | Yes |

Table 3. 5 CHARGE 5 SENSORS

|  |  |
| --- | --- |
| Sensors: |  |
| GPS | Yes |
| Accelerometer | Yes |
| HRM | Yes |
| Gyroscope | No |
| Pulse Ox. | Yes |
| Altimeter | No |
| Barometer | No |
| Compass | No |
| ECG | Yes |
| Thermometer | Yes |
| Ambient light | Yes |

Table 3. 6 CHARGE 5 BATTERY PROPERTIES

|  |  |
| --- | --- |
| Battery type | non-removable Lithium-polymer |
| Battery capacity | N/A |
| Battery life | Regular: 7 days+, Standby:10 days+ |
| Charging type | Charging dock |
| Charging duration | 2hrs |

Table 5. 1 system codes

|  |  |
| --- | --- |
| Function | Code |
| Information system  Written in Tkinter python . and in MySQL data base is storage. The system include front end using buttons and query as the model of the system doing actions. And connecting the buttons to the mysql data base | import tkinter  import tkinter.messagebox  import tkinter.ttk  from tkinter import messagebox  from tkcalendar import Calendar, DateEntry  import folium  import pandas as pd  import webbrowser  from math import radians, sin, cos, sqrt, atan2  import tkinter as tk  import matplotlib.dates as mdates  import matplotlib.pyplot as plt  import mysql.connector  class Database:  def \_init\_(self):  self.dbConnection = mysql.connector.connect(  host="localhost",  user="root",  password="ure363?!!?747",  )  self.dbCursor = self.dbConnection.cursor()  self.dbCursor.execute("CREATE DATABASE IF NOT EXISTS final\_project22")  self.dbCursor.execute("USE final\_project22")  self.dbCursor.execute("CREATE TABLE IF NOT EXISTS doctor\_table (id INT AUTO\_INCREMENT PRIMARY KEY, "  "first\_name VARCHAR(255), last\_name VARCHAR(255), dob DATE, start\_working\_date DATE, "  "phone\_number VARCHAR(15))")  self.dbCursor.execute("CREATE TABLE IF NOT EXISTS patient\_table (patient\_id INT AUTO\_INCREMENT PRIMARY KEY, "  "firstname VARCHAR(255), lastname VARCHAR(255), dateOfBirth DATE, gender VARCHAR(10), "  "address VARCHAR(255), contactNumber VARCHAR(15), emailAddress VARCHAR(255), "  "bloodType VARCHAR(10), history TEXT, doctor\_id INT, "  "FOREIGN KEY (doctor\_id) REFERENCES doctor\_table(id))")  self.dbCursor.execute(  "CREATE TABLE IF NOT EXISTS measurement\_table (patient\_id INT,measurement\_date\_time DATETIME, "  "heart\_rate INT, spo2 INT, gps\_lat FLOAT(10,6), gps\_long FLOAT(10,6), "  "PRIMARY KEY (measurement\_date\_time , patient\_id), FOREIGN KEY (patient\_id) REFERENCES patient\_table(id))")  def \_del\_(self):  self.dbCursor.close()  self.dbConnection.close()  def Insert(self, id,firstname, lastname, dateOfBirth, gender, address, contactNumber, emailAddress, bloodType, history,  doctor\_id):  sql = "INSERT INTO patient\_table (id, firstname, lastname, dateOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor\_id) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"  val = (id,  firstname, lastname, dateOfBirth, gender, address, contactNumber, emailAddress, bloodType, history,  doctor\_id  )  self.dbCursor.execute(sql, val)  self.dbConnection.commit()  def Insert\_doctor(self, firstname, lastname, dateOfBirth, start\_working\_date, contactNumber):  sql = "INSERT INTO doctor\_table (firstname, lastname, dateOfBirth, start\_working\_date, contactNumbe) VALUES (%s, %s, %s, %s, %s)"  val = (  firstname, lastname, dateOfBirth, start\_working\_date, contactNumber)  self.dbCursor.execute(sql, val)  self.dbConnection.commit()  def Update(self, id, firstname, lastname, dateOfBirth, gender, address, contactNumber, emailAddress, bloodType,  history, doctor\_id):  sql = "UPDATE patient\_table SET firstname = %s, lastname = %s, dateOfBirth = %s, gender = %s, address = %s, contactNumber = %s, emailAddress = %s, bloodType = %s, history = %s, doctor\_id = %s WHERE id = %s"  val = (  firstname, lastname, dateOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor\_id,  id)  self.dbCursor.execute(sql, val)  self.dbConnection.commit()  def Search(self, id):  sql = "SELECT \* FROM patient\_table WHERE id = %s"  val = (id,)  self.dbCursor.execute(sql, val)  searchResults = self.dbCursor.fetchall()  return searchResults  def Search\_doctor(self, id):  sql = "SELECT \* FROM doctor\_table WHERE id = %s"  val = (id,)  self.dbCursor.execute(sql, val)  searchResults = self.dbCursor.fetchall()  return searchResults  def Delete(self, id):  sql = "DELETE FROM patient\_table WHERE id = %s"  val = (id,)  self.dbCursor.execute(sql, val)  tkinter.messagebox.showinfo("Deleted data", "Successfully Deleted the Patient data in the database")  self.dbConnection.commit()  def Delete\_doctor(self, id):  sql = "DELETE FROM doctor\_table WHERE id = %s"  val = (id,)  self.dbCursor.execute(sql, val)  tkinter.messagebox.showinfo("Deleted data", "Successfully Deleted the Patient data in the database")  self.dbConnection.commit()  def Display(self):  self.dbCursor.execute("SELECT \* FROM patient\_table")  records = self.dbCursor.fetchall()  return records  def Display\_doctor(self):  self.dbCursor.execute("SELECT \* FROM doctor\_table")  records = self.dbCursor.fetchall()  return records  class Values:  def Validate(self, firstname, lastname, contactNumber, emailAddress):  if not (firstname.isalpha()):  return "firstname"  elif not (lastname.isalpha()):  return "lastname"  elif not (contactNumber.isdigit() and (len(contactNumber) == 10)):  return "contactNumber"  elif not (emailAddress.count("@") == 1 and emailAddress.count(".") > 0):  return "emailAddress"  else:  return "SUCCESS"  class DoctorValues:  def Validate(self, firstname, lastname, contactNumber):  if not (firstname.isalpha()):  return "firstname"  elif not (lastname.isalpha()):  return "lastname"  elif not (contactNumber.isdigit() and (len(contactNumber) == 10)):  return "contactNumber"  else:  return "SUCCESS"  class InsertWindow:  def \_init\_(self):  self.window = tkinter.Tk()  self.window.wm\_title("Insert Patient Data ")  bg\_color = "Blue"  fg\_color = "white"  self.id = tkinter.StringVar()  self.firstname = tkinter.StringVar()  self.lastname = tkinter.StringVar()  self.address = tkinter.StringVar()  self.contactNumber = tkinter.StringVar()  self.emailAddress = tkinter.StringVar()  self.history = tkinter.StringVar()  self.doctor = tkinter.StringVar()  self.genderType = ["Male", "Female", "Transgender", "Other"]  self.bloodListType = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]  # Labels  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, text="Patient Id", font=("times new roman", 10, "bold"),  width=25).grid(pady=5, column=1, row=1)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, text="Patient First Name",  font=("times new roman", 10, "bold"), width=25).grid(pady=5, column=1, row=2)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Last Name", width=25).grid(pady=5, column=1, row=3)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"), text="Date of Birth",  width=25).grid(pady=5, column=1, row=4)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Gender", width=25).grid(pady=5, column=1, row=5)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Address", width=25).grid(pady=5, column=1, row=6)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Contact Number", width=25).grid(pady=5, column=1, row=7)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Email Address", width=25).grid(pady=5, column=1, row=8)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Blood Type", width=25).grid(pady=5, column=1, row=9)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="History of Patient", width=25).grid(pady=5, column=1, row=10)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Name of Doctor", width=25).grid(pady=5, column=1, row=11)  self.idEntry = tkinter.Entry(self.window, width=25, textvariable=self.id)  self.firstnameEntry = tkinter.Entry(self.window, width=25, textvariable=self.firstname)  self.lastnameEntry = tkinter.Entry(self.window, width=25, textvariable=self.lastname)  self.addressEntry = tkinter.Entry(self.window, width=25, textvariable=self.address)  self.contactNumberEntry = tkinter.Entry(self.window, width=25, textvariable=self.contactNumber)  self.emailAddressEntry = tkinter.Entry(self.window, width=25, textvariable=self.emailAddress)  self.historyEntry = tkinter.Entry(self.window, width=25, textvariable=self.history)  self.doctorEntry = tkinter.Entry(self.window, width=25, textvariable=self.doctor)  self.idEntry.grid(pady=5, column=3, row=1)  self.firstnameEntry.grid(pady=5, column=3, row=2)  self.lastnameEntry.grid(pady=5, column=3, row=3)  self.addressEntry.grid(pady=5, column=3, row=6)  self.contactNumberEntry.grid(pady=5, column=3, row=7)  self.emailAddressEntry.grid(pady=5, column=3, row=8)  self.historyEntry.grid(pady=5, column=3, row=10)  self.doctorEntry.grid(pady=5, column=3, row=11)  self.dobEntry = DateEntry(self.window, date\_pattern="yyyy-mm-dd")  # Combobox widgets  self.genderBox = tkinter.ttk.Combobox(self.window, values=self.genderType, width=25)  self.bloodListBox = tkinter.ttk.Combobox(self.window, values=self.bloodListType, width=25)  self.dobEntry.grid(pady=5, column=3, row=4)  self.genderBox.grid(pady=5, column=3, row=5)  self.bloodListBox.grid(pady=5, column=3, row=9)  # Button widgets  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Insert", command=self.Insert).grid(pady=15, padx=5, column=1,  row=14)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Reset", command=self.Reset).grid(pady=15, padx=5, column=2, row=14)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Close", command=self.window.destroy).grid(pady=15, padx=5, column=3,  row=14)  self.window.mainloop()  def Insert(self):  self.values = Values()  self.database = Database()  self.values = Values()  self.database = Database()  self.test = self.values.Validate(self.firstnameEntry.get(), self.lastnameEntry.get(),  self.contactNumberEntry.get(), self.emailAddressEntry.get())  # Rest of the code...  if (self.test == "SUCCESS"):  self.database.Insert(self.idEntry.get(), self.firstnameEntry.get(), self.lastnameEntry.get(),  self.dobEntry.get(),  self.genderBox.get(), self.addressEntry.get(),  self.contactNumberEntry.get(), self.emailAddressEntry.get(), self.bloodListBox.get(),  self.historyEntry.get(), self.doctorEntry.get())  tkinter.messagebox.showinfo("Inserted data", "Successfully inserted the above data in the database")  else:  self.valueErrorMessage = "Invalid input in field " + self.test  tkinter.messagebox.showerror("Value Error", self.valueErrorMessage)  def Reset(self):  self.idEntry.delete(0, tkinter.END)  self.firstnameEntry.delete(0, tkinter.END)  self.lastnameEntry.delete(0, tkinter.END)  self.genderBox.set("")  self.addressEntry.delete(0, tkinter.END)  self.contactNumberEntry.delete(0, tkinter.END)  self.emailAddressEntry.delete(0, tkinter.END)  self.bloodListBox.set("")  self.historyEntry.delete(0, tkinter.END)  self.doctorEntry.delete(0, tkinter.END)  class InsertDoctorWindow:  def \_init\_(self):  self.doctorvalue = DoctorValues()  self.window = tkinter.Tk()  self.window.wm\_title("Insert Doctor Data ")  bg\_color = "Blue"  fg\_color = "white"  # id, first\_name, last\_name, dob, start\_working\_date, phone\_number  self.id = tkinter.StringVar()  self.firstname = tkinter.StringVar()  self.lastname = tkinter.StringVar()  self.contactNumber = tkinter.StringVar()  # Labels  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, text="Doctor Id", font=("times new roman", 10, "bold"),  width=25).grid(pady=5, column=1, row=1)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, text="Doctor First Name",  font=("times new roman", 10, "bold"), width=25).grid(pady=5, column=1, row=2)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Doctor Last Name", width=25).grid(pady=5, column=1, row=3)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"), text="Date of Birth",  width=25).grid(pady=5, column=1, row=4)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="start work date", width=25).grid(pady=5, column=1, row=5)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Contact Number", width=25).grid(pady=5, column=1, row=9)  self.idEntry = tkinter.Entry(self.window, width=25, textvariable=self.id)  self.firstnameEntry = tkinter.Entry(self.window, width=25, textvariable=self.firstname)  self.lastnameEntry = tkinter.Entry(self.window, width=25, textvariable=self.lastname)  self.dobEntry = DateEntry(self.window, date\_pattern="yyyy-mm-dd")  self.StartWorkDateEntry = DateEntry(self.window, date\_pattern="yyyy-mm-dd")  self.contactNumberEntry = tkinter.Entry(self.window, width=25, textvariable=self.contactNumber)  self.idEntry.grid(pady=5, column=3, row=1)  self.firstnameEntry.grid(pady=5, column=3, row=2)  self.lastnameEntry.grid(pady=5, column=3, row=3)  self.dobEntry.grid(pady=5, column=3, row=4)  self.StartWorkDateEntry.grid(pady=5, column=3, row=5)  self.contactNumberEntry.grid(pady=5, column=3, row=6)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Insert", command=self.Insert).grid(pady=15, padx=5, column=1,  row=14)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Reset", command=self.Reset).grid(pady=15, padx=5, column=2, row=14)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Close", command=self.window.destroy).grid(pady=15, padx=5, column=3,  row=14)  self.window.mainloop()  def Insert(self):  # self.values = Values()  self.database = Database()  self.test = self.doctorvalue.Validate(self.firstnameEntry.get(), self.lastnameEntry.get(),  self.contactNumberEntry.get())  if (self.test == "SUCCESS"):  self.database.Insert\_doctor(self.idEntry.get(), self.firstnameEntry.get(), self.lastnameEntry.get(),  self.dobEntry.get(),  self.contactNumberEntry.get())  tkinter.messagebox.showinfo("Inserted data", "Successfully inserted the above data in the database")  else:  self.valueErrorMessage = "Invalid input in field " + self.test  tkinter.messagebox.showerror("Value Error", self.valueErrorMessage)  def Reset(self):  self.idEntry.delete(0, tkinter.END)  self.firstnameEntry.delete(0, tkinter.END)  self.lastnameEntry.delete(0, tkinter.END)  self.contactNumberEntry.delete(0, tkinter.END)  class UpdateWindow:  def \_init\_(self, id):  self.window = tkinter.Tk()  self.window.wm\_title("Update data")  bg\_color = "Blue"  fg\_color = "white"  # Initializing all the variables  self.id = (id)  self.firstname = tkinter.StringVar()  self.lastname = tkinter.StringVar()  self.address = tkinter.StringVar()  self.contactNumber = tkinter.StringVar()  self.emailAddress = tkinter.StringVar()  self.history = tkinter.StringVar()  self.doctor\_id = tkinter.StringVar()  self.genderType = ["Male", "Female", "Transgender", "Other"]  self.bloodListType = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]  # Labels  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, text="Patient Id", font=("times new roman", 10, "bold"),  width=25).grid(pady=5, column=1, row=1)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, text="Patient First Name",  font=("times new roman", 10, "bold"), width=25).grid(pady=5, column=1, row=2)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Last Name", width=25).grid(pady=5, column=1, row=3)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"), text="Date of Birth",  width=25).grid(pady=5, column=1, row=4)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Gender", width=25).grid(pady=5, column=1, row=5)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Address", width=25).grid(pady=5, column=1, row=6)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Contact Number", width=25).grid(pady=5, column=1, row=7)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Email Address", width=25).grid(pady=5, column=1, row=8)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Patient Blood Type", width=25).grid(pady=5, column=1, row=9)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="History of Patient", width=25).grid(pady=5, column=1, row=10)  tkinter.Label(self.window, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Doctor id", width=25).grid(pady=5, column=1, row=11)  # Set previous values  self.database = Database()  self.searchResults = self.database.Search(id)  if len(self.searchResults) > 0:  tkinter.Label(self.window, text=self.searchResults[0][1], width=25).grid(pady=5, column=2, row=2)  tkinter.Label(self.window, text=self.searchResults[0][2], width=25).grid(pady=5, column=2, row=3)  tkinter.Label(self.window, text=self.searchResults[0][3], width=25).grid(pady=5, column=2, row=4)  tkinter.Label(self.window, text=self.searchResults[0][4], width=25).grid(pady=5, column=2, row=5)  tkinter.Label(self.window, text=self.searchResults[0][5], width=25).grid(pady=5, column=2, row=6)  tkinter.Label(self.window, text=self.searchResults[0][6], width=25).grid(pady=5, column=2, row=7)  tkinter.Label(self.window, text=self.searchResults[0][7], width=25).grid(pady=5, column=2, row=8)  tkinter.Label(self.window, text=self.searchResults[0][8], width=25).grid(pady=5, column=2, row=9)  tkinter.Label(self.window, text=self.searchResults[0][9], width=25).grid(pady=5, column=2, row=10)  tkinter.Label(self.window, text=self.searchResults[0][10], width=25).grid(pady=5, column=2, row=11)  #tkinter.Label(self.window, text=self.searchResults[0][11], width=25).grid(pady=5, column=2, row=12)  else:  tkinter.messagebox.showinfo("Search Error", "No records found with the given ID")  self.idEntry = tkinter.Entry(self.window, width=25, textvariable=self.id)  self.firstnameEntry = tkinter.Entry(self.window, width=25, textvariable=self.firstname)  self.lastnameEntry = tkinter.Entry(self.window, width=25, textvariable=self.lastname)  self.addressEntry = tkinter.Entry(self.window, width=25, textvariable=self.address)  self.contactNumberEntry = tkinter.Entry(self.window, width=25, textvariable=self.contactNumber)  self.emailAddressEntry = tkinter.Entry(self.window, width=25, textvariable=self.emailAddress)  self.historyEntry = tkinter.Entry(self.window, width=25, textvariable=self.history)  self.doctorEntry = tkinter.Entry(self.window, width=25, textvariable=self.doctor\_id)  self.idEntry.grid(pady=5, column=3, row=1)  self.firstnameEntry.grid(pady=5, column=3, row=2)  self.lastnameEntry.grid(pady=5, column=3, row=3)  self.addressEntry.grid(pady=5, column=3, row=8)  self.contactNumberEntry.grid(pady=5, column=3, row=9)  self.emailAddressEntry.grid(pady=5, column=3, row=10)  self.historyEntry.grid(pady=5, column=3, row=12)  self.doctorEntry.grid(pady=5, column=3, row=13)  self.dobEntry = DateEntry(self.window, date\_pattern="yyyy-mm-dd")  # Combobox  self.genderBox = tkinter.ttk.Combobox(self.window, values=self.genderType, width=20)  self.bloodListBox = tkinter.ttk.Combobox(self.window, values=self.bloodListType, width=20)  self.dobEntry.grid(pady=5, column=3, row=4)  self.genderBox.grid(pady=5, column=3, row=7)  self.bloodListBox.grid(pady=5, column=3, row=11)  # Button  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Update", command=self.Update).grid(pady=15, padx=5, column=1,  row=14)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Reset", command=self.Reset).grid(pady=15, padx=5, column=2, row=14)  tkinter.Button(self.window, width=10, fg=fg\_color, bg=bg\_color, font=("times new roman", 10, "bold"),  text="Close", command=self.window.destroy).grid(pady=15, padx=5, column=3,  row=14)  self.window.mainloop()  def Update(self):  self.database = Database()  self.database.Update(self.firstnameEntry.get(), self.lastnameEntry.get(), self.dobEntry.get(),  self.genderBox.get(), self.addressEntry.get(),  self.contactNumberEntry.get(),  self.emailAddressEntry.get(), self.bloodListBox.get(), self.historyEntry.get(),  self.doctorEntry.get(), self.id)  tkinter.messagebox.showinfo("Updated data", "Successfully updated the above data in the database")  def Reset(self):  self.idEntry.delete(0, tkinter.END)  self.firstnameEntry.delete(0, tkinter.END)  self.lastnameEntry.delete(0, tkinter.END)  self.genderBox.set("")  self.addressEntry.delete(0, tkinter.END)  self.contactNumberEntry.delete(0, tkinter.END)  self.emailAddressEntry.delete(0, tkinter.END)  self.bloodListBox.set("")  self.historyEntry.delete(0, tkinter.END)  self.doctorEntry.delete(0, tkinter.END)  class DatabaseView:  def \_init\_(self, data):  self.databaseViewWindow = tkinter.Tk()  self.databaseViewWindow.wm\_title("Database View")  # Label widgets  tkinter.Label(self.databaseViewWindow, text="Database View Window", width=25).grid(pady=5, column=1, row=1)  self.databaseView = tkinter.ttk.Treeview(self.databaseViewWindow)  self.databaseView.grid(pady=5, column=1, row=2)  self.databaseView["show"] = "headings"  self.databaseView["columns"] = (  "id", "firstname", "lastname", "dateOfBirth", "gender", "address",  "contactNumber", "emailAddress", "bloodType", "history",  "doctor\_id")  # id, firstname, lastname, dateOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor\_id  # Treeview column headings  self.databaseView.heading("id", text="Patient ID")  self.databaseView.heading("firstname", text="First Name")  self.databaseView.heading("lastname", text="Last Name")  self.databaseView.heading("dateOfBirth", text="Date of Birth")  self.databaseView.heading("gender", text="Gender")  self.databaseView.heading("address", text="Home Address")  self.databaseView.heading("contactNumber", text="Contact Number")  self.databaseView.heading("emailAddress", text="Email Address")  self.databaseView.heading("bloodType", text="Blood Type")  self.databaseView.heading("history", text="History")  self.databaseView.heading("doctor\_id", text="Doctor")  # Treeview columns  self.databaseView.column("id", width=100)  self.databaseView.column("firstname", width=100)  self.databaseView.column("lastname", width=100)  self.databaseView.column("dateOfBirth", width=100)  self.databaseView.column("gender", width=100)  self.databaseView.column("address", width=200)  self.databaseView.column("contactNumber", width=100)  self.databaseView.column("emailAddress", width=200)  self.databaseView.column("bloodType", width=100)  self.databaseView.column("history", width=100)  self.databaseView.column("doctor\_id", width=100)  self.db = mysql.connector.connect(  host="localhost",  user="root",  password="ure363?!!?747",  database="final\_project22"  )  self.cursor = self.db.cursor()  # Insert data into MySQL database table  self.cursor.execute("SELECT \* FROM patient\_table")  data = self.cursor.fetchall()  for record in data:  self.databaseView.insert('', 'end', values=record)  self.databaseViewWindow.mainloop()  class DoctorDatabaseView:  def \_init\_(self, data):  self.doctordatabaseViewWindow = tkinter.Tk()  self.doctordatabaseViewWindow.wm\_title("Database View")  # Label widgets  tkinter.Label(self.doctordatabaseViewWindow, text="Doctors View Window", width=25).grid(pady=5, column=1, row=1)  self.doctordatabaseView = tkinter.ttk.Treeview(self.doctordatabaseViewWindow)  self.doctordatabaseView.grid(pady=5, column=1, row=2)  self.doctordatabaseView["show"] = "headings"  self.doctordatabaseView["columns"] = (  "id", "firstname", "lastname", "dob", "start\_working\_date", "phone\_number")  # Treeview column headings  self.doctordatabaseView.heading("id", text="Doctor ID")  self.doctordatabaseView.heading("firstname", text="First Name")  self.doctordatabaseView.heading("lastname", text="Last Name")  self.doctordatabaseView.heading("dob", text="Date of Birth")  self.doctordatabaseView.heading("start\_working\_date", text="Start work date")  self.doctordatabaseView.heading("phone\_number", text="Phone Number")  # id,first\_name, last\_name ,dob,start\_working\_date,phone\_number  # Treeview columns  self.doctordatabaseView.column("id", width=100)  self.doctordatabaseView.column("firstname", width=100)  self.doctordatabaseView.column("lastname", width=100)  self.doctordatabaseView.column("dob", width=100)  self.doctordatabaseView.column("start\_working\_date", width=100)  self.doctordatabaseView.column("phone\_number", width=100)  self.db = mysql.connector.connect(  host="localhost",  user="root",  password="ure363?!!?747",  database="final\_project22"  )  self.cursor = self.db.cursor()  # Insert data into MySQL database table  self.cursor.execute("SELECT \* FROM doctor\_table")  data = self.cursor.fetchall()  for record in data:  self.doctordatabaseView.insert('', 'end', values=record)  self.doctordatabaseViewWindow.mainloop()  class patientAdminstrationWindow:  def \_init\_(self):  self.patientWindow = tkinter.Tk()  self.patientWindow.wm\_title("Patient Administration")  bg\_color = "blue"  fg\_color = "white"  lbl\_color = 'green'  tkinter.Label(self.patientWindow, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Patient Administration",  font=("times new roman", 20, "bold"), width=60).grid(pady=20, column=1, row=1)  tkinter.Button(self.patientWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Insert new patient", font=("times new roman", 15, "bold"), command=self.Insert).grid(  pady=15,  column=1,  row=3)  tkinter.Button(self.patientWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Update patient", font=("times new roman", 15, "bold"), command=self.Update).grid(pady=15,  column=1,  row=4)  tkinter.Button(self.patientWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Search patient", font=("times new roman", 15, "bold"), command=self.Search).grid(pady=15,  column=1,  row=5)  tkinter.Button(self.patientWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Delete patient", font=("times new roman", 15, "bold"), command=self.Delete).grid(pady=15,  column=1,  row=6)  tkinter.Button(self.patientWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Display patient", font=("times new roman", 15, "bold"), command=self.Display).grid(pady=15,  column=1,  row=7)  tkinter.Button(self.patientWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Exit",  font=("times new roman", 15, "bold"), command=self.patientWindow.destroy).grid(pady=15,  column=1,  row=8)  self.patientWindow.mainloop()  def Insert(self):  self.insertWindow = InsertWindow()  def Update(self):  self.updateIDWindow = tkinter.Tk()  self.updateIDWindow.wm\_title("Update data")  # Initializing all the variables  self.id = tkinter.StringVar()  # Label  tkinter.Label(self.updateIDWindow, text="Enter the ID to update", width=50).grid(pady=20, row=1)  # Entry widgets  self.idEntry = tkinter.Entry(self.updateIDWindow, width=5, textvariable=self.id)  self.idEntry.grid(pady=10, row=2)  # Button widgets  tkinter.Button(self.updateIDWindow, width=20, text="Update", command=self.updateID).grid(pady=10, row=3)  self.updateIDWindow.mainloop()  def updateID(self):  self.updateWindow = UpdateWindow(self.idEntry.get())  self.updateIDWindow.destroy()  def Search(self):  self.searchWindow = SearchDeleteWindow("Search")  def Delete(self):  self.deleteWindow = SearchDeleteWindow("Delete")  def Display(self):  self.database = Database()  self.data = self.database.Display()  self.displayWindow = DatabaseView(self.data)  class DoctorAdminstrationWindow:  def \_init\_(self):  self.doctorWindow = tkinter.Tk()  self.doctorWindow.wm\_title("Doctor Administration")  bg\_color = "blue"  fg\_color = "white"  lbl\_color = 'green'  tkinter.Label(self.doctorWindow, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Doctor Administration",  font=("times new roman", 20, "bold"), width=60).grid(pady=20, column=1, row=1)  tkinter.Button(self.doctorWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Insert new doctor", font=("times new roman", 15, "bold"), command=self.Insert\_doctor).grid(  pady=15,  column=1,  row=3)  tkinter.Button(self.doctorWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Update doctor", font=("times new roman", 15, "bold"), command=self.Update).grid(pady=15,  column=1,  row=4)  tkinter.Button(self.doctorWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Search doctor", font=("times new roman", 15, "bold"), command=self.Search\_doctor).grid(  pady=15,  column=1,  row=5)  tkinter.Button(self.doctorWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Delete doctor", font=("times new roman", 15, "bold"), command=self.Delete\_doctor).grid(  pady=15,  column=1,  row=6)  tkinter.Button(self.doctorWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Display doctor", font=("times new roman", 15, "bold"), command=self.Display\_doctor).grid(  pady=15,  column=1,  row=7)  tkinter.Button(self.doctorWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Exit",  font=("times new roman", 15, "bold"), command=self.doctorWindow.destroy).grid(pady=15,  column=1,  row=8)  self.doctorWindow.mainloop()  def Insert\_doctor(self):  self.insertWindow = InsertDoctorWindow()  def Update(self):  self.updateIDWindow = tkinter.Tk()  self.updateIDWindow.wm\_title("Update data")  # Initializing all the variables  self.id = tkinter.StringVar()  # Label  tkinter.Label(self.updateIDWindow, text="Enter the ID to update", width=50).grid(pady=20, row=1)  # Entry widgets  self.idEntry = tkinter.Entry(self.updateIDWindow, width=5, textvariable=self.id)  self.idEntry.grid(pady=10, row=2)  # Button widgets  tkinter.Button(self.updateIDWindow, width=20, text="Update", command=self.updateID).grid(pady=10, row=3)  self.updateIDWindow.mainloop()  def updateID(self):  self.updateWindow = UpdateWindow(self.idEntry.get())  self.updateIDWindow.destroy()  def Search\_doctor(self):  self.searchWindow = SearchDeletedoctorWindow("Search")  def Delete\_doctor(self):  self.deleteWindow = SearchDeletedoctorWindow("Delete")  def Display\_doctor(self):  self.database = Database()  self.data = self.database.Display\_doctor()  self.displayWindow = DoctorDatabaseView(self.data)  class measuresWindow(Database):  def \_init\_(self):  super().\_init\_() # Call the parent class constructor  self.measuresWindow = tkinter.Tk()  self.measuresWindow.wm\_title("Patient Measurements")  bg\_color = "blue"  fg\_color = "white"  lbl\_color = 'green'  tkinter.Label(self.measuresWindow, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Measurements Page",  font=("times new roman", 20, "bold"), width=60).grid(pady=20, column=1, row=1)  tkinter.Button(self.measuresWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Take Measurements", font=("times new roman", 15, "bold"),  command=self.take\_measurements).grid(  pady=15, column=1, row=3)  tkinter.Button(self.measuresWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Search Patient Measurements", font=("times new roman", 15, "bold"),  command=self.search\_measurements).grid(pady=15,  column=1,  row=4)  tkinter.Button(self.measuresWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Display Measurements", font=("times new roman", 15, "bold"),  command=self.display\_measurements).grid(pady=15,  column=1,  row=5)  tkinter.Button(self.measuresWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Exit", font=("times new roman", 15, "bold"), command=self.measuresWindow.destroy).grid(  pady=15,  column=1,  row=6)  self.measuresWindow.mainloop()  def take\_measurements(self):  # Prompt for patient ID  # ... implementation of taking measurements ...  pass  def search\_measurements(self):  # Add functionality for searching patient measurements here  pass  def display\_measurements(self):  display\_window = tkinter.Toplevel(self.measuresWindow)  display\_window.title("Display Measurements")  display\_window.geometry("400x200")  def show\_spo2\_heartrate():  patient\_id = patient\_id\_entry.get()  query = f"SELECT \* FROM measurement\_table WHERE patient\_id = {patient\_id}"  self.dbCursor.execute(query)  rows = self.dbCursor.fetchall()  # Convert rows to a DataFrame  rows\_df = pd.DataFrame(rows,  columns=['patient\_id', 'measurement\_date\_time', 'heart\_rate', 'spo2', 'gps\_lat',  'gps\_long'])  # Access columns using column names  heart\_rate\_values = rows\_df['heart\_rate']  spo2\_values = rows\_df['spo2']  timestamps = pd.to\_datetime(rows\_df['measurement\_date\_time'], format='%Y-%m-%d %H:%M:%S')  fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6))  ax1.plot(timestamps, heart\_rate\_values, 'b.-')  ax1.set\_xlabel('Time')  ax1.set\_ylabel('Heart Rate')  ax2.plot(timestamps, spo2\_values, 'r.-')  ax2.set\_xlabel('Time')  ax2.set\_ylabel('SpO2')  # Format the date and time  date\_fmt = '%Y-%m-%d\n'  time\_fmt = '%H:%M:%S'  ax1.xaxis.set\_major\_locator(mdates.AutoDateLocator())  ax1.xaxis.set\_major\_formatter(mdates.DateFormatter(date\_fmt + time\_fmt))  ax2.xaxis.set\_major\_locator(mdates.AutoDateLocator())  ax2.xaxis.set\_major\_formatter(mdates.DateFormatter(date\_fmt + time\_fmt))  plt.tight\_layout()  plt.show(block=False)  max\_heart\_rate\_avg = 0  max\_spo2\_avg = 0  for i in range(len(rows)):  if i >= 4:  heart\_rate\_avg = sum(heart\_rate\_values[i - 4:i + 1]) / 5  spo2\_avg = sum(spo2\_values[i - 4:i + 1]) / 5  if heart\_rate\_avg > 90 and heart\_rate\_avg > max\_heart\_rate\_avg:  max\_heart\_rate\_avg = heart\_rate\_avg  if spo2\_avg < 90 and spo2\_avg > max\_spo2\_avg:  max\_spo2\_avg = spo2\_avg  warnings = []  if max\_heart\_rate\_avg > 0:  warnings.append(f"Patient heart rate is high! Maximum average: {max\_heart\_rate\_avg:.2f}")  if max\_spo2\_avg > 0:  warnings.append(f"Patient SpO2 is low! Maximum average: {max\_spo2\_avg:.2f}")  root = tk.Tk()  root.withdraw()  def show\_warnings():  if warnings:  messagebox.showwarning("Warnings", "\n".join(warnings))  root.after(200, show\_warnings)  root.mainloop()  def show\_warnings():  if warnings:  messagebox.showwarning("Warnings", "\n".join(warnings))  root.after(200, show\_warnings)  root.mainloop()  def generate\_patient\_map():  map\_center = [32.08063, 34.78857]  m = folium.Map(location=map\_center, zoom\_start=12)  patient\_id = patient\_id\_entry.get()  query = f"SELECT \* FROM measurement\_table WHERE patient\_id = {patient\_id}"  self.dbCursor.execute(query)  rows = self.dbCursor.fetchall()  rows\_df = pd.DataFrame(rows,  columns=['patient\_id', 'measurement\_date\_time', 'heart\_rate', 'spo2', 'gps\_lat',  'gps\_long'])  def add\_patient\_markers(rows\_df):  grouped\_df = rows\_df.groupby(['gps\_lat', 'gps\_long'])  for (lat, lon), group in grouped\_df:  for index, row in group.iterrows():  timestamp = row['measurement\_date\_time']  popup\_text = f"Timestamp: {timestamp}"  folium.Marker(  location=[lat, lon],  icon=folium.DivIcon(  icon\_size=(30, 30),  icon\_anchor=(15, 15),  html='<div><img src="patient\_icon.png" style="width:30px;height:30px;"></div>'  ),  popup=popup\_text  ).add\_to(m)  def calculate\_distance(lat1, lon1, lat2, lon2):  # approximate radius of Earth in km  R = 6371.0  lat1\_rad = radians(lat1)  lon1\_rad = radians(lon1)  lat2\_rad = radians(lat2)  lon2\_rad = radians(lon2)  dlon = lon2\_rad - lon1\_rad  dlat = lat2\_rad - lat1\_rad  a = sin(dlat / 2) \* 2 + cos(lat1\_rad) \* cos(lat2\_rad) \* sin(dlon / 2) \* 2  c = 2 \* atan2(sqrt(a), sqrt(1 - a))  distance = R \* c  return distance  def check\_patient\_activity(rows):  max\_distance = 0  for i in range(1, len(rows)):  lat1, lon1 = rows.loc[i - 1, 'gps\_lat'], rows.loc[i - 1, 'gps\_long']  lat2, lon2 = rows.loc[i, 'gps\_lat'], rows.loc[i, 'gps\_long']  distance = calculate\_distance(lat1, lon1, lat2, lon2)  if distance > max\_distance:  max\_distance = distance  if max\_distance > 10:  warning\_text = f"Patient is running away! Maximum distance from Ichilov: {round(max\_distance)} km"  messagebox.showwarning("Warning", warning\_text)  warning\_label = tk.Label(window, text=warning\_text)  warning\_label.pack()  if len(rows\_df) > 0:  # Add patient markers to the map  add\_patient\_markers(rows\_df)  # Create a path connecting the measurements  patient\_measurements = rows\_df[['gps\_lat', 'gps\_long']].values.tolist()  folium.PolyLine(locations=patient\_measurements, color='red', weight=3).add\_to(m)  # Save the map to an HTML file  map\_html = 'patient\_map.html'  m.save(map\_html)  # Create a Tkinter window  window = tk.Tk()  window.title("Patient Map and Warnings")  # Open the HTML file in the default web browser  webbrowser.open(map\_html)  # Check patient activity and show warnings  check\_patient\_activity(rows\_df)  else:  messagebox.showwarning("Warning", "No measurements found for the specified patient.")  window.mainloop()  def show\_patient\_location():  # Add functionality to show patient location on the map here  generate\_patient\_map()  patient\_id\_label = tkinter.Label(display\_window, text="Patient ID:")  patient\_id\_label.pack()  patient\_id\_entry = tkinter.Entry(display\_window)  patient\_id\_entry.pack()  # Create buttons for showing heart rate and spo2 graph, and patient location  spo2\_heartrate\_button = tkinter.Button(display\_window, width=30, relief=tkinter.GROOVE, fg='white', bg='blue',  text="Show Heart Rate and SpO2 Graph",  font=("times new roman", 12, "bold"),  command=lambda: show\_spo2\_heartrate())  spo2\_heartrate\_button.pack(pady=20)  patient\_location\_button = tkinter.Button(display\_window, width=30, relief=tkinter.GROOVE, fg='white', bg='blue',  text="Show Patient Location", font=("times new roman", 12, "bold"),  command=show\_patient\_location)  patient\_location\_button.pack(pady=20)  class SearchDeleteWindow:  def \_init\_(self, task):  window = tkinter.Tk()  window.wm\_title(task + " data")  # Initializing all the variables  self.id = tkinter.StringVar()  self.firstname = tkinter.StringVar()  self.lastname = tkinter.StringVar()  self.heading = "Please enter Patient ID to " + task  # Labels  tkinter.Label(window, text=self.heading, width=50).grid(pady=20, row=1)  tkinter.Label(window, text="Patient ID", width=10).grid(pady=5, row=2)  # Entry widgets  self.idEntry = tkinter.Entry(window, width=5, textvariable=self.id)  self.idEntry.grid(pady=5, row=3)  # Button widgets  if (task == "Search"):  tkinter.Button(window, width=20, text=task, command=self.Search).grid(pady=15, padx=5, column=1, row=14)  elif (task == "Delete"):  tkinter.Button(window, width=20, text=task, command=self.Delete).grid(pady=15, padx=5, column=1, row=14)  def Search(self):  self.database = Database()  self.data = self.database.Search(self.idEntry.get())  self.databaseView = DatabaseView(self.data)  def Delete(self):  self.database = Database()  self.database.Delete(self.idEntry.get())  class SearchDeletedoctorWindow:  def \_init\_(self, task):  window = tkinter.Tk()  window.wm\_title(task + " data")  # Initializing all the variables  self.id = tkinter.StringVar()  self.firstname = tkinter.StringVar()  self.lastname = tkinter.StringVar()  self.heading = "Please enter Doctor ID to " + task  # Labels  tkinter.Label(window, text=self.heading, width=50).grid(pady=20, row=1)  tkinter.Label(window, text="Doctor ID", width=10).grid(pady=5, row=2)  # Entry widgets  self.idEntry = tkinter.Entry(window, width=5, textvariable=self.id)  self.idEntry.grid(pady=5, row=3)  # Button widgets  if (task == "Search"):  tkinter.Button(window, width=20, text=task, command=self.Search).grid(pady=15, padx=5, column=1, row=14)  elif (task == "Delete"):  tkinter.Button(window, width=20, text=task, command=self.Delete).grid(pady=15, padx=5, column=1, row=14)  def Search(self):  self.database = Database()  self.data = self.database.Search\_doctor(self.idEntry.get())  self.databaseView = DatabaseView(self.data)  def Delete(self):  self.database = Database()  self.database.Delete\_doctor(self.idEntry.get())  class HomePage:  def \_init\_(self):  self.homePageWindow = tkinter.Tk()  self.homePageWindow.wm\_title("bracelet Information system Home Page")  bg\_color = "blue"  fg\_color = "white"  lbl\_color = 'green'  tkinter.Label(self.homePageWindow, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Home Page",  font=("times new roman", 20, "bold"), width=60, height=6).grid(pady=20, column=1, row=1)  # tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Insert",  # font=("times new roman", 15, "bold"), command=self.Insert).grid(pady=15, column=1, row=3)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="patient administration",  font=("times new roman", 15, "bold"), command=self.patient\_admin).grid(pady=15, column=1, row=3)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="measurements",  font=("times new roman", 15, "bold"), command=self.measure).grid(pady=15, column=1, row=4)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="doctor administration",  font=("times new roman", 15, "bold"), command=self.doctor\_admin).grid(pady=15, column=1, row=5)  """tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Update",  font=("times new roman", 15, "bold"), command=self.Update).grid(pady=15, column=1, row=7)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Search",  font=("times new roman", 15, "bold"), command=self.Search).grid(pady=15, column=1, row=8)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Delete",  font=("times new roman", 15, "bold"), command=self.Delete).grid(pady=15, column=1, row=9)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Display",  font=("times new roman", 15, "bold"), command=self.Display).grid(pady=15, column=1,  row=10)  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color,  text="Patient measurments",  font=("times new roman", 15, "bold"), command=self.measure).grid(pady=15, column=1, row=11)"""  tkinter.Button(self.homePageWindow, width=30, relief=tkinter.GROOVE, fg=fg\_color, bg=bg\_color, text="Exit",  font=("times new roman", 15, "bold"), command=self.homePageWindow.destroy).grid(pady=15,  column=1,  row=6)  self.homePageWindow.mainloop()  def Insert(self):  self.insertWindow = InsertWindow()  def measure(self):  self.measuresWindow = measuresWindow()  def patient\_admin(self):  self.patientAdminstrationWindow = patientAdminstrationWindow()  def doctor\_admin(self):  self.DoctorAdminstrationWindow = DoctorAdminstrationWindow()  def Update(self):  self.updateIDWindow = tkinter.Tk()  self.updateIDWindow.wm\_title("Update data")  # Initializing all the variables  self.id = tkinter.StringVar()  # Label  tkinter.Label(self.updateIDWindow, text="Enter the ID to update", width=50).grid(pady=20, row=1)  # Entry widgets  self.idEntry = tkinter.Entry(self.updateIDWindow, width=5, textvariable=self.id)  self.idEntry.grid(pady=10, row=2)  # Button widgets  tkinter.Button(self.updateIDWindow, width=20, text="Update", command=self.updateID).grid(pady=10, row=3)  self.updateIDWindow.mainloop()  def updateID(self):  self.updateWindow = UpdateWindow(self.idEntry.get())  self.updateIDWindow.destroy()  def Search(self):  self.searchWindow = SearchDeleteWindow("Search")  def Delete(self):  self.deleteWindow = SearchDeleteWindow("Delete")  def Display(self):  self.database = Database()  self.data = self.database.Display()  self.displayWindow = DatabaseView(self.data)  homePage = HomePage() |
| Heart rate | #define USE\_ARDUINO\_INTERRUPTS true // Set-up low-level interrupts for most acurate BPM math.  #include <PulseSensorPlayground.h>  // Variables  const char PulseWire = A11; // PulseSensor PURPLE WIRE connected to ANALOG PIN A11  const int LED13 = 13;  int Threshold = 550; // Determine which Signal to "count as a beat" and which to ignore.  // Use the "Gettting Started Project" to fine-tune Threshold Value beyond default setting.  // Otherwise leave the default "550" value.    PulseSensorPlayground pulseSensor;  void setup() {  Serial.begin(9600); // For Serial COMUNNCATION  // Configure the PulseSensor object, by assigning our variables to it.  pulseSensor.analogInput(PulseWire);  pulseSensor.blinkOnPulse(LED13);  pulseSensor.setThreshold(Threshold);  // Double-check the "pulseSensor" object was created and "began" seeing a signal.  if (pulseSensor.begin()) {  Serial.println("We created a pulseSensor Object !");  }  }  void loop() {  int myBPM = pulseSensor.getBeatsPerMinute(); // Calls function on our pulseSensor object that returns BPM as an "int".  // "myBPM" hold this BPM value now.  if (pulseSensor.sawStartOfBeat()) { // Constantly test to see if "a beat happened".  Serial.println("♥ A HeartBeat Happened ! "); // If test is "true", print a message "a heartbeat happened".  Serial.print("BPM: "); // Print phrase "BPM: "  Serial.println(myBPM); // Print the value inside of myBPM.  }  delay(20);  } |
| GPS | #include <SoftwareSerial.h>  #include <TinyGPS++.h>  SoftwareSerial gpsSerial(8,9);  TinyGPSPlus gps;  float lattitude,longitude;  void setup() {   gpsSerial.begin(9600);   Serial.begin(9600);    }  void loop()  {    while (gpsSerial.available())    {      int data = gpsSerial.read();      if (gps.encode(data))      {        lattitude = (gps.location.lat());        longitude = (gps.location.lng());        Serial.print ("lattitude: ");        Serial.println (lattitude);        Serial.print ("longitude: ");        Serial.println (longitude);      }    }  } |
| LED alarm | // Pin connected to the LED  const int ledPin = 8 ;  // Blink interval in milliseconds  const unsigned long blinkInterval = 500;  // Variables to track the current state and time  bool ledState = LOW;  unsigned long previousMillis = 0;  void setup() {    // Set the LED pin as an output    pinMode(ledPin, OUTPUT);  }  void loop() {    // Get the current time    unsigned long currentMillis = millis();    // Check if it's time to change the LED state    if (currentMillis - previousMillis >= blinkInterval) {      // Save the current time      previousMillis = currentMillis;      // Toggle the LED state      ledState = !ledState;      // Set the LED pin to the new state      digitalWrite(ledPin, ledState);    }  } |