

HEALTH CARE BOT

Final Report

Project: Project Integration

Company: Saxion University Of Applied Science

Address: Enschede, The Netherlands

Date: Enschede, 28th June, 2019

Version: 1.0

Drawn up by:	Hong Trinh	438443@student.saxion.nl	Software
	Phuc Le	456061@student.saxion.nl	Software
	Minh Thanh Nguyen	444856@student.saxion.nl	Hardware
	Minh Le	467475@student.saxion.nl	Hardware
	Van Nguyen	444878@student.saxion.nl	Hardware

Supervisors: Ramazan Kirmali

Ümit Güler

Client: Johan Stokkink

Executive Summary

As the main task from the project integration of LED Engineering department in Saxion University of Applied Sciences for 2nd year students of ACS and EIE major, the given topic is about implementing a health-care device with artificial intelligent controlling which can take care and keep track the patient's health status. The final result should be a mobility device which can be able to measure the heart rate of the patient and notify the nurse. The project will be divided into 2 main parts : software where all ACS students gather and work together with the help from Mr. Ramazan Kirmali, meanwhile Mr. Ümit Güler will lead and take care of hardware students from EIE. The final product will be judged by the 2 representors above and our main client, Mr. Johan Stokkin.

Our team, which consists of 5 members, divided into 2 teams. Phuc Le and Hong Trinh are the two in the software team and the hardware team with the present of Minh Thanh Nguyen, Van Nguyen, and Minh Le. The software crew will be responsible for the behaviour of the device which is the functionalities of the wrist band. The other team will take care of designing, soldering and placing components together. Both teams were helping each other to implement and test the device.

The aim is about making the project meet the given requirements: A device - wristband that contain 2 parts - mobility part - that can measure the heart beat of patient by controlling with Amazon Echo Dot Alexa. The device also should be able to notify the nurse about the heart rate status. The outcoming data should be also need to store in local database through cloud and visiable in a window application - stationary part - for the nurse.

The demonstration showed that the device was able to get the live heart beat from patient and store it into local database through cloud. However, the notification for the nurse application on desktop had not been implemented yet due to the lack of time and member in group. However, the device still be able to create notification whenever the heart rate is too high or too low on the mobility device. In general, the result still meet the minimum requirements of the commission party.

Foreword

The project is a cooperation between students who are studying ACS and EIE. For this project, each group was made of 5 to 6 people. The difficulty for the leader increases, since there are more people to supervise. It is also difficult to effectively separate tasks for every member. That was resulting of there are some members being lazy and did less work than others, even quit the project.

The project requires each member being responsibility and working in team profession. However our group is not high level teamwork. The tasks was given but due to some people left the project nearly at the end, the workload increase for the part in the team which had the left person. Moreover, there were troubles and we had to spend lots of time to figure out the problem. In general, the project basically work.

This project also taught us that communication and teamwork are the most critical element to succeed. The connection between hardware and software part is important. The working processing of software part is depended on the processing of hardware part. That means hardware team always ensure their working has high precision for software part can test with their code.

Executive Summary	2
Foreword	3
Abbreviations	6
Introduction	7
Project Problems	7
Preparation and Research	7
ESP32	7
ESP32 featuring	8
ESP32 Pinout and module board versions	9
Implementation	12
Pulse Sensor	12
Pulse sensor featuring	12
Pulse Sensor Pinout and operating	13
OLED SSD1306	13
Echo Dot v2	13
GUI	14
Database (Mysql Database)	14
MQTT broker	14
Hardware Technical overview	16
Electrical	17
Electronics	18
Linear voltage Regulator	18
Digital Alarm	18
PCB for the mobile device	18
ESP mainboard:	18
Heartbeat Sensor	20
Digital Alarm	21
Projectresults:	22
Testing:	24
Acceptance testing:	24
Functional testing:	24
ESP32 testing:	24
OLED testing:	24
Technical testing:	25
[ESP32 + OLED + Pulse Sensor] (stationary part) and Alexa	25

The implementation of stationary part [ESP32 + OLED + Pulse Sensor] and Alexa	25
Libraries	25
Device setup	26
The interruptSetup	28
ISRTr (Interrupt Setup Routine Triggering)	29
Alexa and its features	31
MQTT + Database + GUI (Data part)	32
Code testing:	33
GUI and Database testing:	33
MQTT broker with Nodered	37
Hardware Testing	37
ESP32 mainboard	37
Heartbeat sensor	38
Regulator	40
Digital Alarm	40
Project end result and discussion	41
Hardware Results	44
ESP mainboard	45
Heartbeat Sensor	45
Digital alarm	46
Possible recommendations	47
References	48

Abbreviations

ACS	Applied Computer Science
EIE	Electronic Information Engineering
MCU	MicroController Unit
Alexa	Amazon Echo Dot V2 “Alexa”
GUI	Graphic User Interface
ISRT _r	Interrupt Setup Routines Triggering
MQTT	Message Queuing Telemetry Transport
OLED	Organic light-emitting diode

Introduction

Following behind the project software engineering, the project integration is the one that expands more about realistic. The realistic here is making a mobility measuring heart beat device that must be controlled by voice with a smart home device. In this case, Amazon Echo Alexa was used to implement with the device. By the end of the project, the device should be able to get the heart rate data and display in a small built-in OLED display in the device, together with storing data to the local database through the MQTT broker. Last but not least, all the data in database will be displayed in a user interface - a window application, which is stationary part, for users to keep track of patient's health.

Project Problems

With the given topic is about health care robot, our team came up with building a wristband to keep track of the health status of patients. With that idea, our device should be:

- Measuring/Getting live heart rate and display it on the wrist OLED display
- Real-time measured data also need to storage in a local database
- From that database, a window application is needed to show the live heart rate and also plot a graph of heart rate, together with patient's information.
- If the heart rate is too low or too high, there should be notification for both patient (on the wrist) and nurse (on the window application).

Through the meeting with the client, the client also agreed with our product functionality so that we can move to the realisation phase.

Preparation and Research

The table that show the work division during the project:

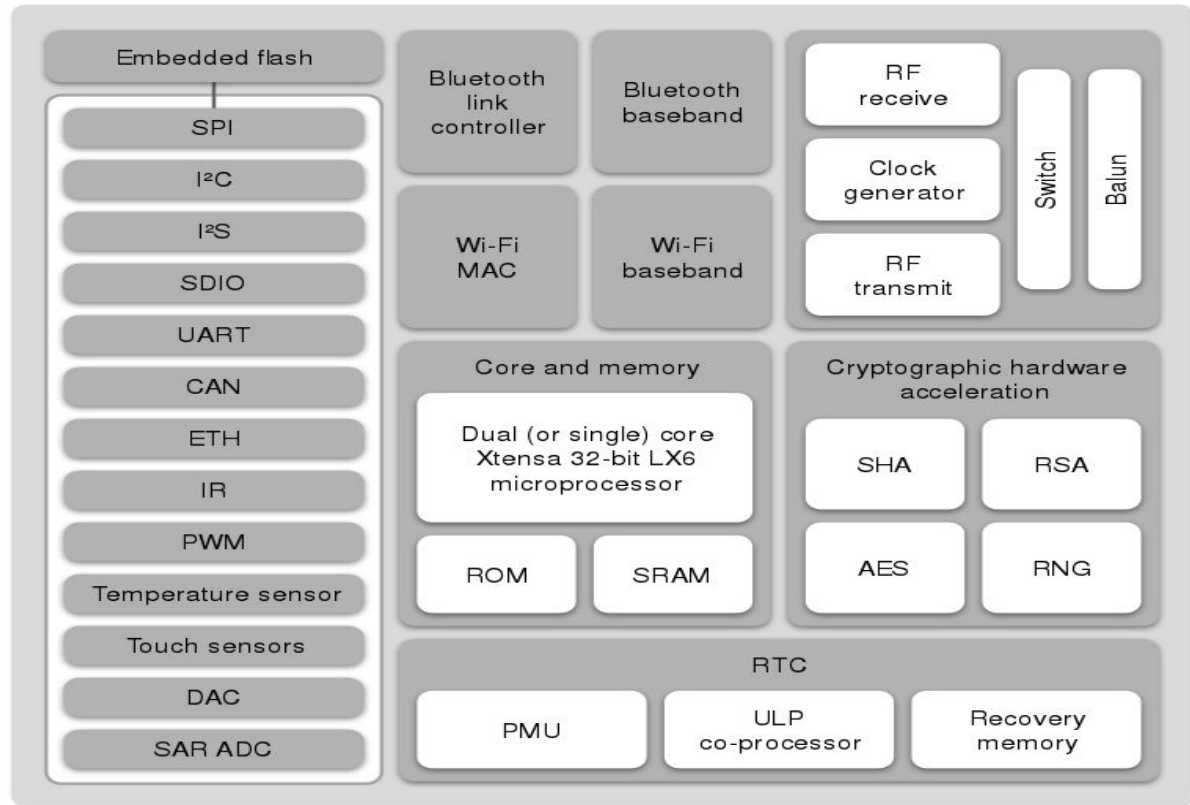
Hong Trinh	GUI, Database, testing
Phuc Le	MQTT, stationary part, Alexa , testing
Minh Le	Heartbeat Sensor
Thanh Nguyen	USB bridge, ESP board
Van Nguyen	Battery regulator, Digital alarm

ESP32

Esp32 is a new Node-MCU with the team. It is a series of low cost, low power system on a microcontrollers chip that integrated Wifi & dual-mode Bluetooth. This chip is created and developed by Espressif Systems and is manufactured by TSMC using their 40 nm process. ESP32 is a success of the elder brother ESP8266.

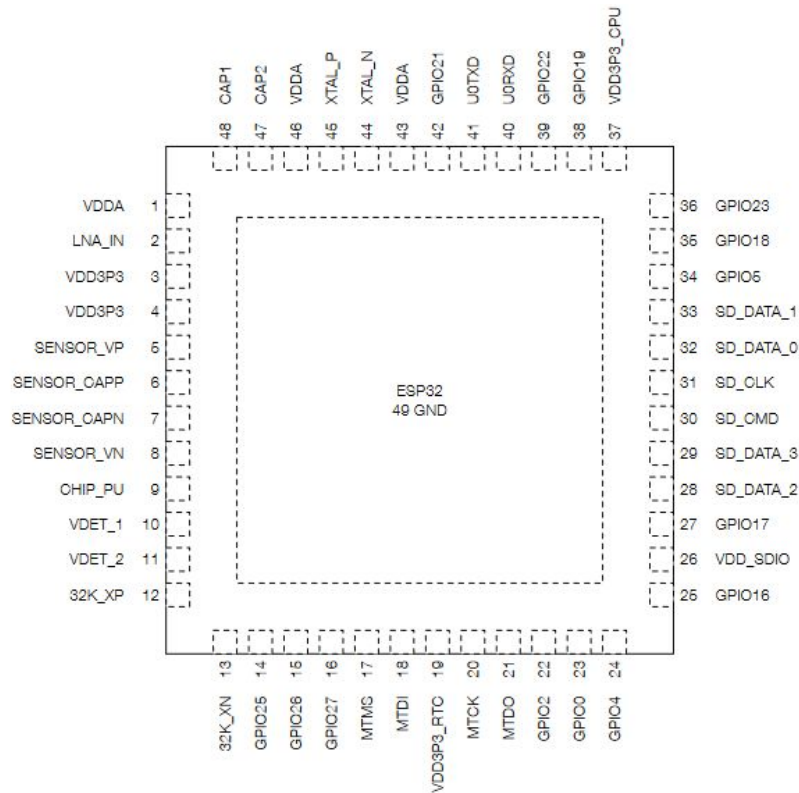
ESP32 featuring

- CPU: Xtensa Dual-Core 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS
- Memory: 520 KiB SRAM
- Wireless connectivity:
 - + Wi-Fi: 802.11 b/g/n/e/i
 - + Bluetooth: v4.2 BR/EDR and BLE
- Peripheral interfaces:
 - + 12-bit SAR ADC up to 18 channels
 - + 2 × 8-bit DACs
 - + 10 × touch sensors
 - + Temperature sensor
 - + 4 × SPI
 - + 2 × I²S
 - + 2 × I²C
 - + 3 × UART
 - + SD/SDIO/MMC host
 - + Slave (SDIO/SPI)
 - + Ethernet MAC interface with dedicated DMA and IEEE 1588 support
 - + CAN bus 2.0
 - + IR (TX/RX)
 - + Motor PWM
 - + LED PWM up to 16 channels
 - + Hall effect sensor
 - + Ultra low power analog preamplifier
- Security:
 - + IEEE 802.11 standard security features all supported, including WPA, WPA/WPA2 and WAPI
 - + Secure boot
 - + Flash encryption
 - + 1024-bit OTP, up to 768-bit for customers
 - + Cryptographic hardware acceleration: AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)



ESP32 Pinout and module board versions

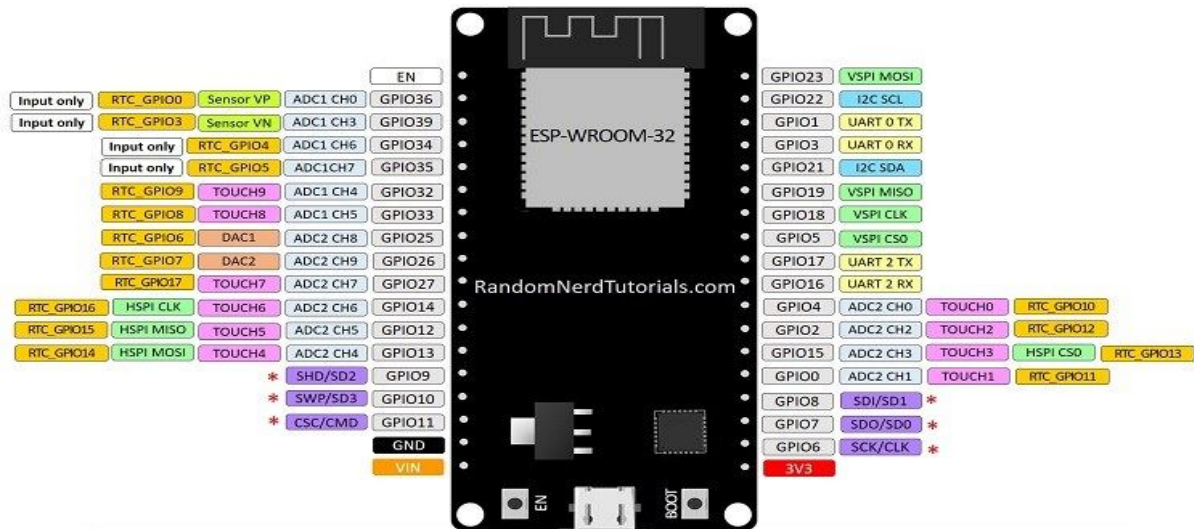
Before digging deep into the version of the module boards, we have the pin layout of the Node-MCU ESP32



All the detail description of all pins can be found in the datasheet of ESP32 from the manufacturer

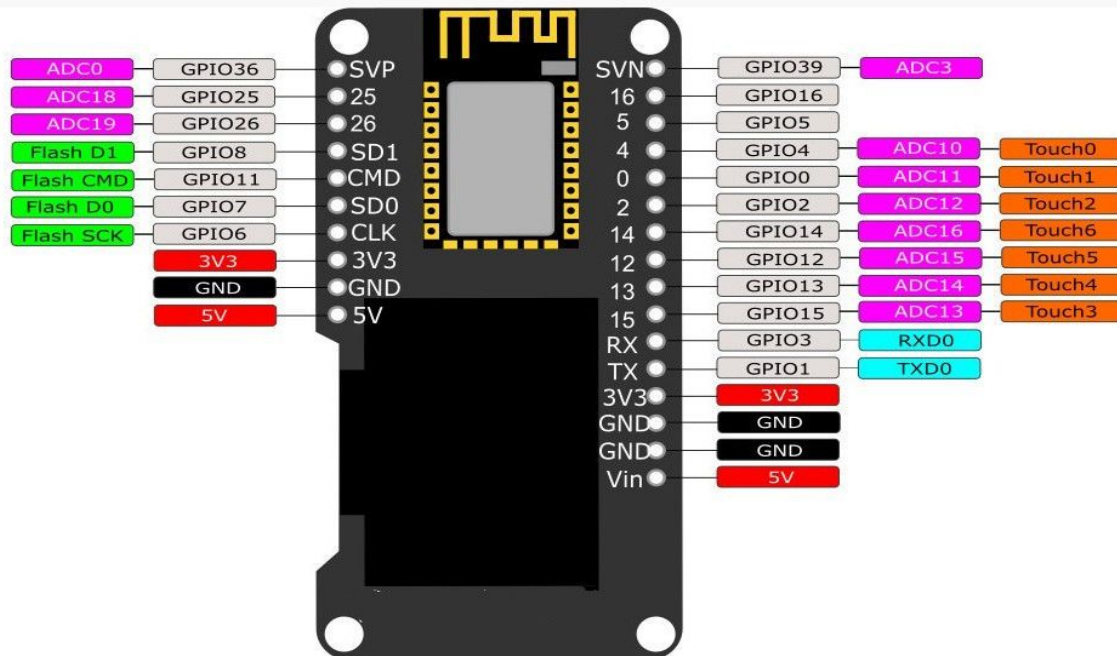
ESP32 DEVKIT V1 – DOIT

version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SC5/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

At first, the team had some ESP32 module boards which are small PCBs directly contain the ESP32 chip. This board was designed to be easily used by other circuit boards. There are some version of ESP32 module board. In this project, the team only tried out the version of 36 GPIO pins (ESP32 DEVKIT V1 - DOIT) and the less GPIOs version that contained a built-in OLED screen (WEMOS/Lolin)



Implementation

To implement the ESP32, software group can use micropython which is specialized for MCU chip or we can use the C programming language by using Arduino IDE to programming it. However, this is a new chip, therefore, its libraries for implementation are not really common as its brother ESP8266. By comparing between Micropython and Arduino IDE, all members in team decided to go for the Arduino IDE for implementing the chip which has many more available library to use and program it.

For instance, a program of blinking LED in the ESP32 module board can be represented in Micropython and Arduino IDE as below:

- Micropython

```
import time
from machine import Pin
led=Pin(15,Pin.OUT)

while True:
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
```
- Arduino IDE

```
const int ledPin = 15;
void setup() {
  pinMode (ledPin, OUTPUT);
}
void loop() {
  digitalWrite (ledPin, HIGH);    delay(500);
  digitalWrite (ledPin, LOW);    delay(500);
}
```

The only problem here is how common is the implementation version? Micropython or Arduino IDE ? To decide which one, let's have a look at the other two critical part of the mobility device: Pulse sensor and Alexa.

Pulse Sensor

Pulse sensor featuring

- Biometric Pulse Rate or Heart Rate detecting sensor
- Plug and Play type sensor
- Operating Voltage: +5V or +3.3V
- Current Consumption: 4mA
- Inbuilt Amplification and Noise cancellation circuit.
- Diameter: 0.625"



- Thickness: 0.125" Thick

However, this is the premade version from the manufacturer. In this project, the hardware team have to rebuild or re-used this version of pulse sensor.

Pulse Sensor Pinout and operating

As the figure above shows that there are 3 pins of this sensor which are:

- Ground: connect to the ground of the system
- Vcc: connect to the power supply voltage with the range of 3.3 to 5V
- Signal: the output pulsating signal of the sensor. This is an analog pin.

This pulse sensor can be said that it uses plug-and-play mode. The design of the sensor is simple. It only have 3 pins. To use it, just simply power it up by connecting the Vcc and Ground pin to the power supply and ground. From 3.3 to 5v is a suitable voltage supply range for the sensor that can work out its best. The other pin is the analog signal pin. Therefore, an ADC pin in the microcontroller is needed.

There are some notices when using the pulse sensor:

- The surface is really sensitive, therefore covering the sensor with vinyl tape or some other non conductive materials is recommended.
- Using dry hands when measuring for the best measuring as well as not broke the sensor
- When measuring, the flat side of the sensor should be placed on top of the vein or on the fingerprint. Also slight pressure should be applied on it for detecting the beat or the velcro tapes are recommended to attain this pressure.

OLED SSD1306

SSD1306 is a single-chip CMOS OLED/PLED driver with controller for organic / polymer light emitting diode dot-matrix graphic display system. This IC is designed for Common Cathode type OLED panel. The SSD1306 embeds with contrast control, display RAM and oscillator, which reduces the number of external components and power consumption. It has 256-step brightness control. It is suitable for many compact portable applications, such as Smart watch, real-time image display of camera on smart car, battery management device. Oled ssd 1306 use I2C communication to connect with esp32.

There are some advantages of SSD1306:

- More elaborate and beautiful screen than LCD with more functions.
- High contrast, supporting clear display with no need of backlight
- Working voltage 2.7V - 5.5V.
- Low power consumption: 0.04W during normal operation.

Echo Dot v2

Echo Dot V2 is a device which detects the voice command from the owner which is based on in order to control the other devices on or off. With Echo Dot, we can modified to control the esp32. Then through ESP32, user can control other devices (This project is heartbeat sensor)

GUI

The software team decided to use the c# for implement the window application for users. C# is a powerful and modern language with all the features. The most rapid and convenient way to create a user interface is to do so visually by using the windows forms designer and toolbox. There are two parts name as Windows Form control and a control.

Windows Forms controls are reusable components that encapsulate user interface e control functionality and are used in client side Windows based applications.

A control is a component on a form used to display information or accept user input. The control class provides the foundation functionality for all controls which are displayed on a form.

Database (Mysql Database)

Also play an important role in this project, the database is the soul that connects the stationary and mobility part together. Database table is created to store the information of users and the heartbeat rate with the real-time data.

MQTT broker

MQTT is a critical part in this project due to its role. To push and pull data from the device to the database, MQTT, as a bridge that can open the connection to allow data transfer from device to database and in opposite way. To use MQTT broker as a bridge to transfer sent and received data from both sides of the bridge, the installation is required. In this project, the team only focus on the mosquitto MQTT broker.

To setup MQTT broker, there are some key phases:

Install the mosquito. After installing, go to the source folder of mosquito and run it by command prompt

```
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Program Files\mosquitto>mosquitto.exe
```

Install Node.js (run PowerShell as administrator) with the following command:

```
npm install -g --unsafe-perm node-red
```

To check the installation, use command:

```
Node-red
```



```

C:\Windows\system32> npm install -g --unsafe-perm node-red
C:\ProgramData\npmglobal\node-red -> C:\ProgramData\npmglobal\node_modules\node-red\red.js
C:\ProgramData\npmglobal\node-red -> C:\ProgramData\npmglobal\node_modules\node-red\bin\node-red-pi

bcrypt@3.0.5 install C:\ProgramData\npmglobal\node_modules\node-red\node_modules\bcrypt
node-pre-gyp install --fallback-to-build

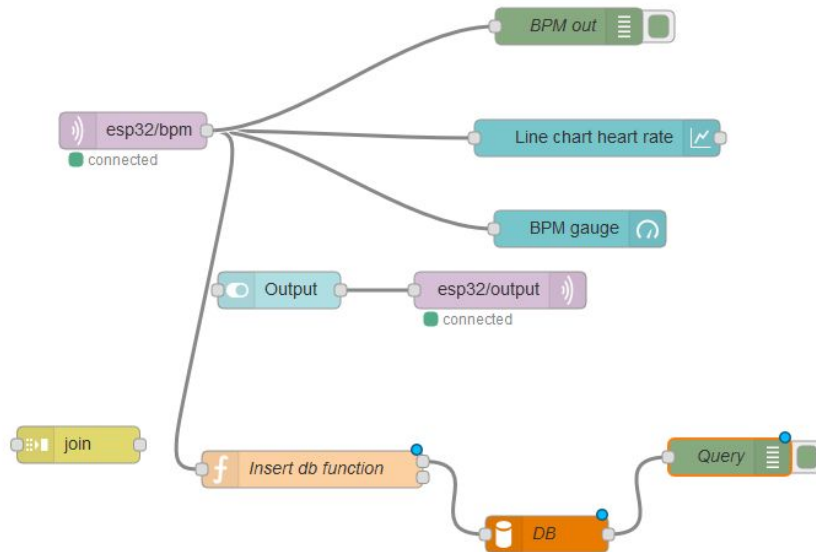
node-pre-gyp WARN "C:\ProgramData\npmglobal\node_modules\node-red\node_modules\bcrypt\lib\binding\bcrypt_lib.node" is installed via re
s
node-red@0.20.5
added 362 packages from 350 contributors in 57.457s
C:\Windows\system32> node-red
5 Jun 23:27:24 - [info]
Welcome to Node-RED
=====
5 Jun 23:27:24 - [info] Node-RED version: v0.20.5
5 Jun 23:27:24 - [info] Node.js version: v11.4.0
5 Jun 23:27:24 - [info] Windows_NT 10.0.17134 x64 LE
5 Jun 23:27:26 - [info] Loading palette nodes
5 Jun 23:27:38 - [warn] rpi-gpio : Raspberry Pi specific node set inactive
5 Jun 23:27:39 - [info] Settings file : C:\Users\lhpke\node-red\settings.js
5 Jun 23:27:39 - [info] Context store : 'default' [module=memory]
5 Jun 23:27:39 - [info] User directory : C:\Users\lhpke\node-red
5 Jun 23:27:39 - [warn] Projects disabled : editorTheme.projects.enabled=false
5 Jun 23:27:39 - [info] Flows file : C:\Users\lhpke\node-red\flows_DESKTOP-B1MQ33E.json
5 Jun 23:27:39 - [info] Creating new flow file
5 Jun 23:27:39 - [warn]
-----
Your flow credentials file is encrypted using a system-generated key.
If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.
You should set your own key using the 'credentialSecret' option in
our settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----
5 Jun 23:27:40 - [info] Starting flows
5 Jun 23:27:40 - [info] Started flows
5 Jun 23:27:40 - [info] Server now running at http://127.0.0.1:1880/

```

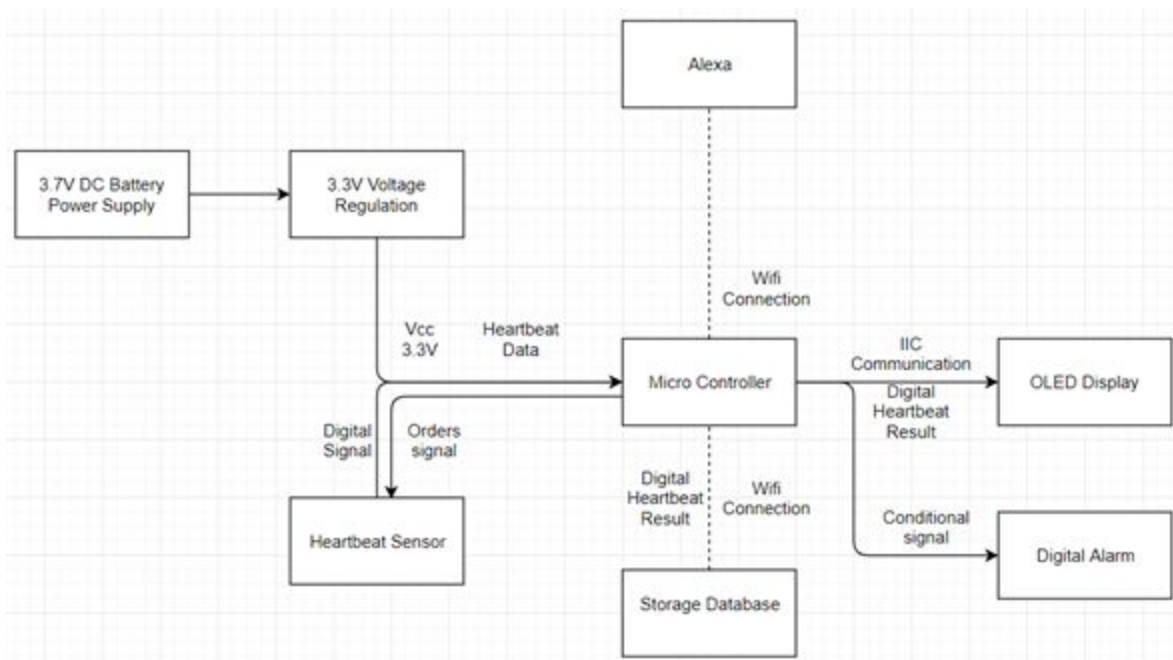
As it can be seen that the server is running at <http://127.0.0.1:1880/> which the IP address is 127.0.0.1 and port 1880

Then install the node-red dashboard with command `npm install node-red-dashboard`

Copy and paste the following link to web browser to access the node-RED

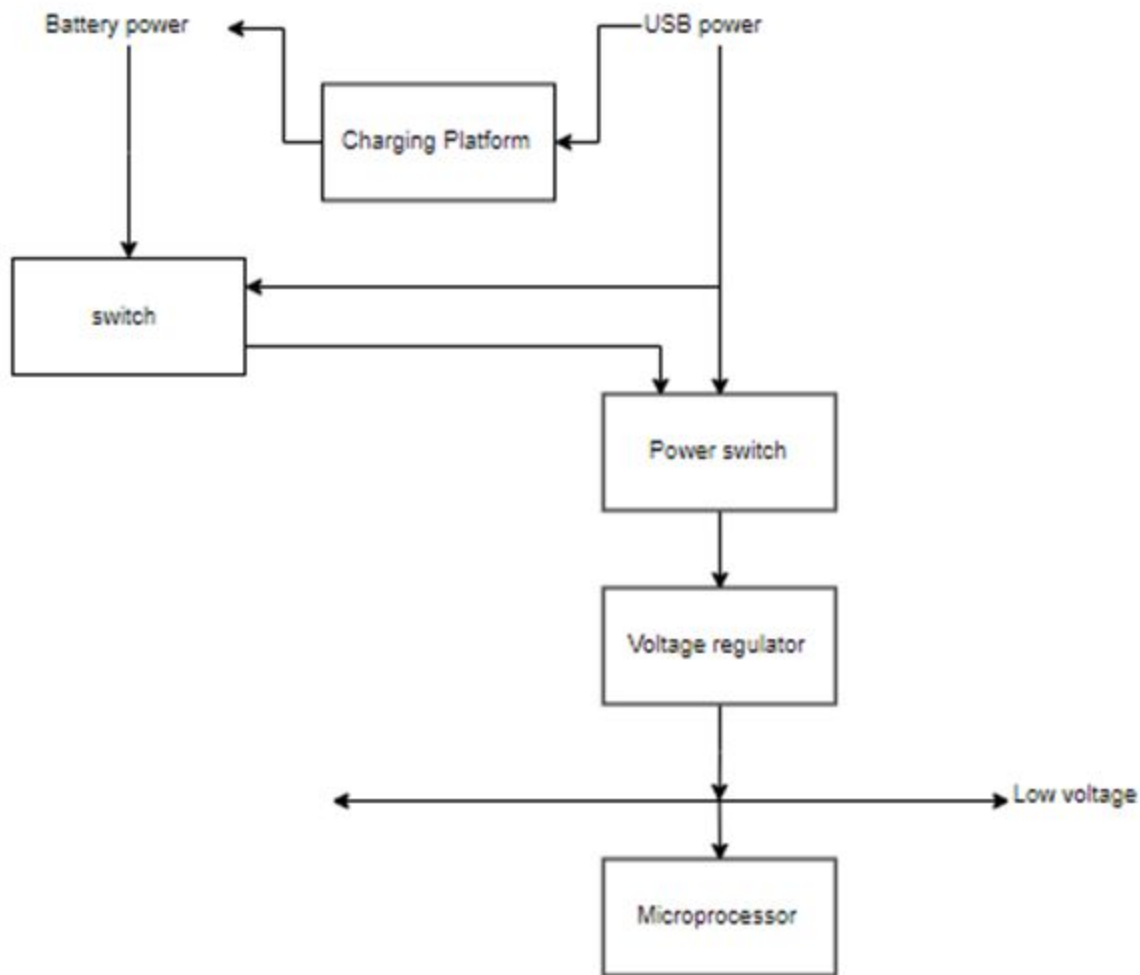


Hardware Technical overview



The Microcontroller will receive the order from Alexa to trigger the Heartbeat Sensor. As soon as the Sensor is trigger, it will measure the user's heart rate and transfer it back to Microcontroller. After that, Micro Controller will show the value on the Display. If the user's heart rate reach the dangerous limit, the Digital Alarm will be triggered in order to warn the user, Then the Microcontroller continue to receive the order from Alexa

Electrical



When the USB port is connected to the mobile device it will have two functions, the first one is to supply the battery line (battery – charging circuit) with 5 volts on the charging platform connected in parallel will give a 4.2 Vout to the battery for charging which is done by a special IC “MCP73831” that will transform 5 volts to 4.2 volts to charge it. Secondly it will disconnect the battery from the device while charging using a MOSFET switch and the device will be powered directly from the USB port 5 volts which will go to the linear regulator to make it a 3.3 volt to power up the esp32 module.

When the battery is charged, and the USB port is not connected the Vout of the battery will go to the converter and the Vout of the converter 3.3 volts will power up the esp32 module and the rest of the circuit.

Electronics

Linear voltage Regulator

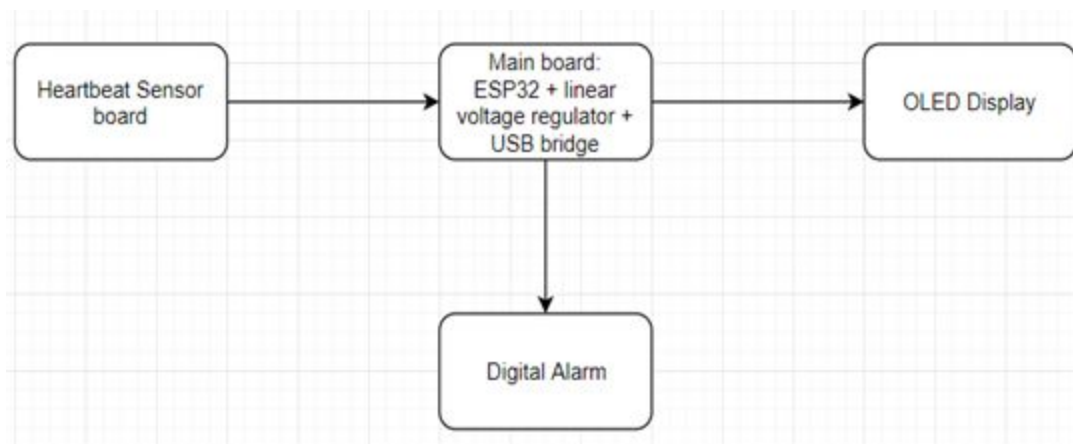
The power loss of the linear voltage source is mostly caused by the difference of the input voltage and the output voltage. When the difference is high, then the power loss increases. The main advantage of the linear voltage source is simple and low cost. In order to reduce the voltage from 3.7V to 3.3V by using a linear voltage source, we need the Low Drop-Out Voltage Regulator. In this case, Low Drop-Out Voltage Regulator TPS715A33DRVT is used to build the linear voltage regulator because of its low power loss.

Digital Alarm

Digital alarm will be triggered when the user's heartbeat reaches the limitation. The circuit needs the small piezo buzzer and the transistor 2N3904.

PCB for the mobile device

The mobile device is divided into 4 parts: Sensor board, main board, alarm board and OLED Display. Each board connects to each other by wires.



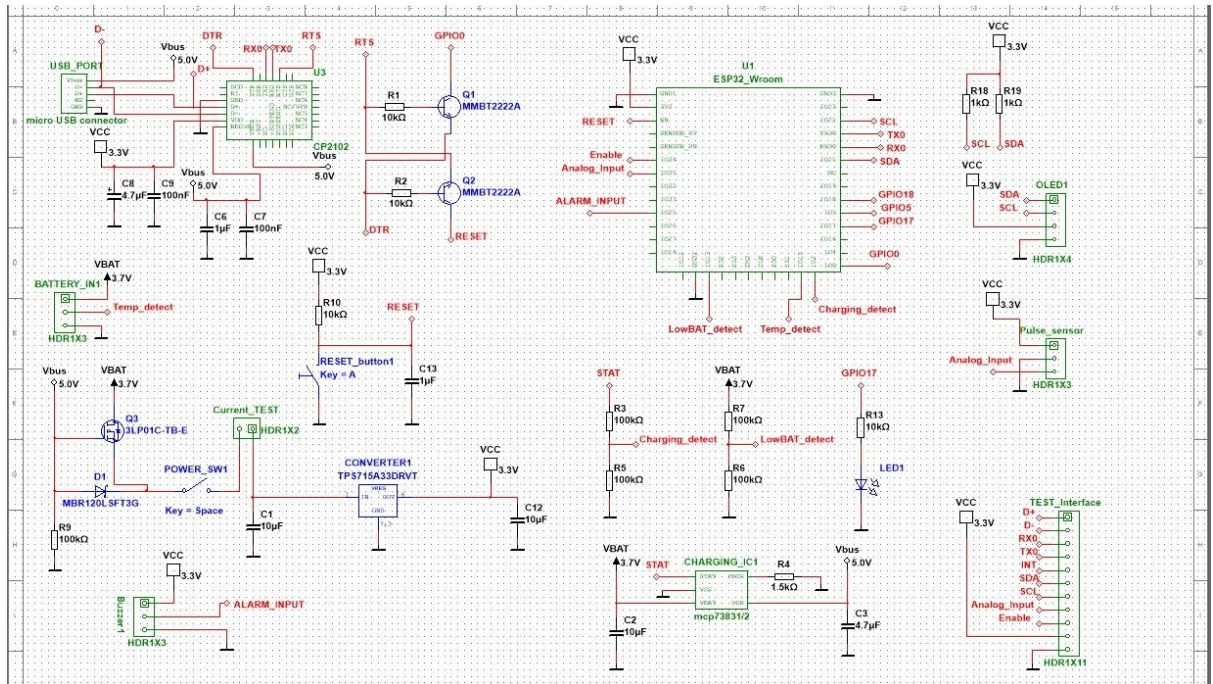
ESP mainboard:

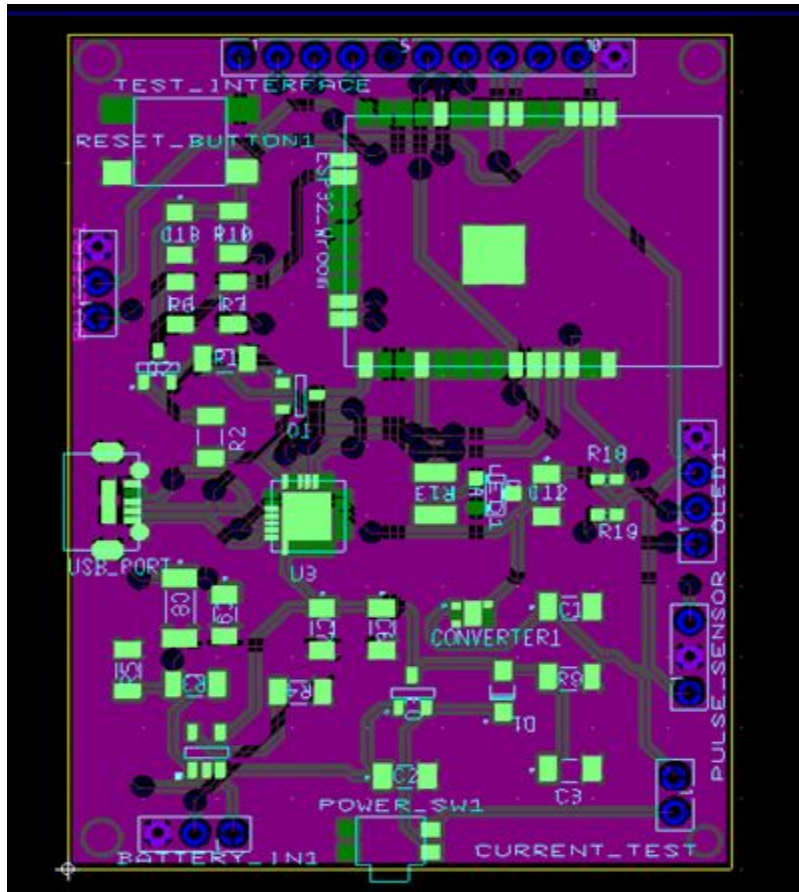
ESP mainboard is the main PCB of the mobile device. It includes the ESP32, linear voltage regulator and connects with the other 3 parts of mobile device.

The mainboard includes:

- ESP32 chip
- USB circuit: CP2102 USB to UART bridge controller.
- Charging circuit: using MCP73831/2 chip for charging the battery.

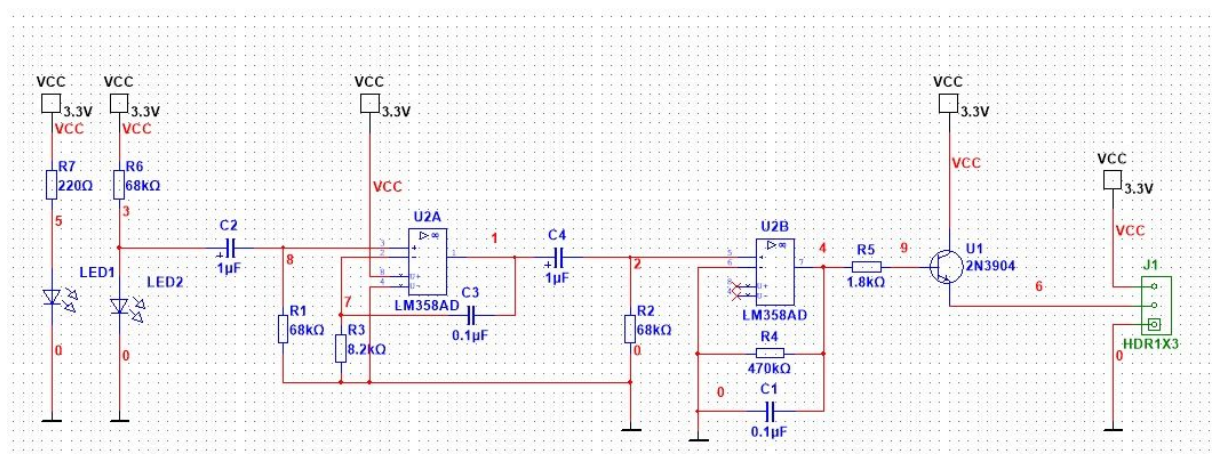
- Power button: turn ON/OFF the device
- Design footprint for OLED screen, Sensor, Buzzer and Test Interface.
- Using the mosfet as a switch. The device has 2 options for operation:
 - Vbat = 3.7V
 - Vbus = 5V

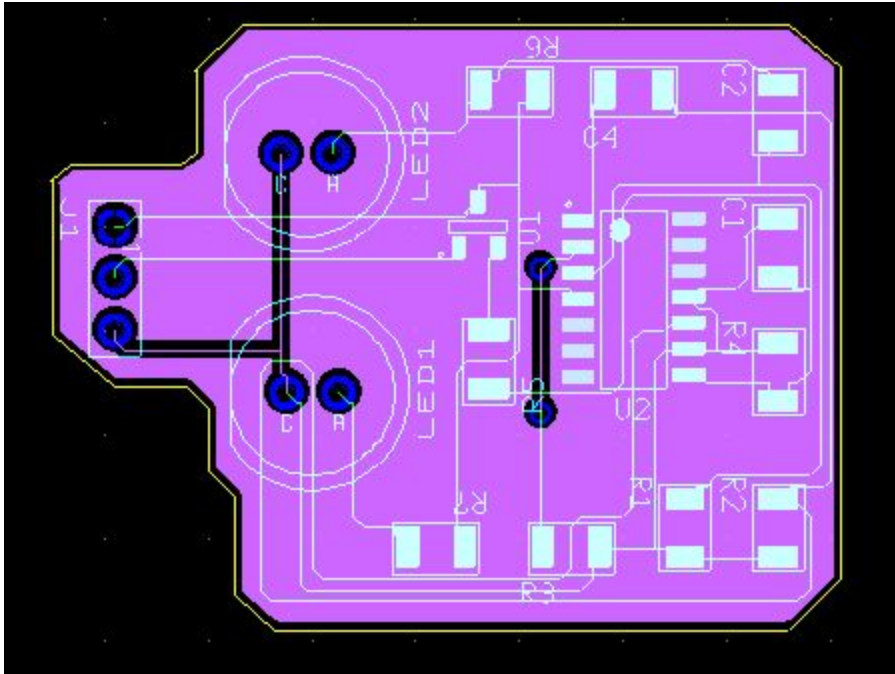




Heartbeat Sensor

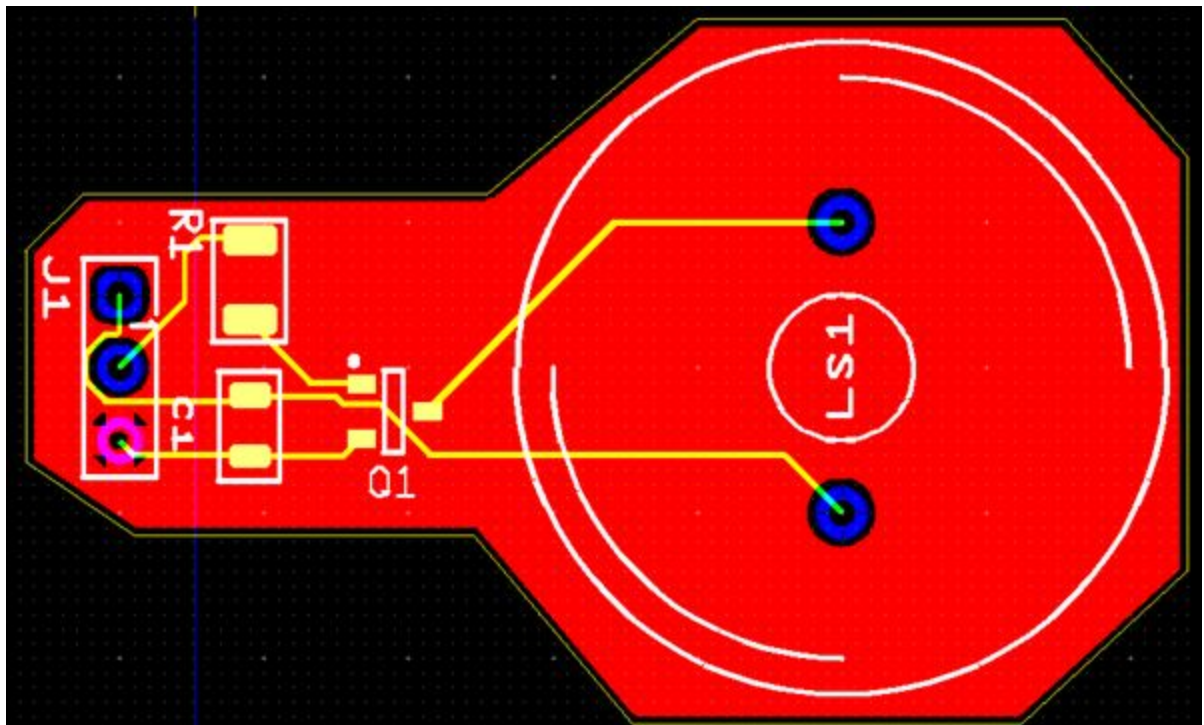
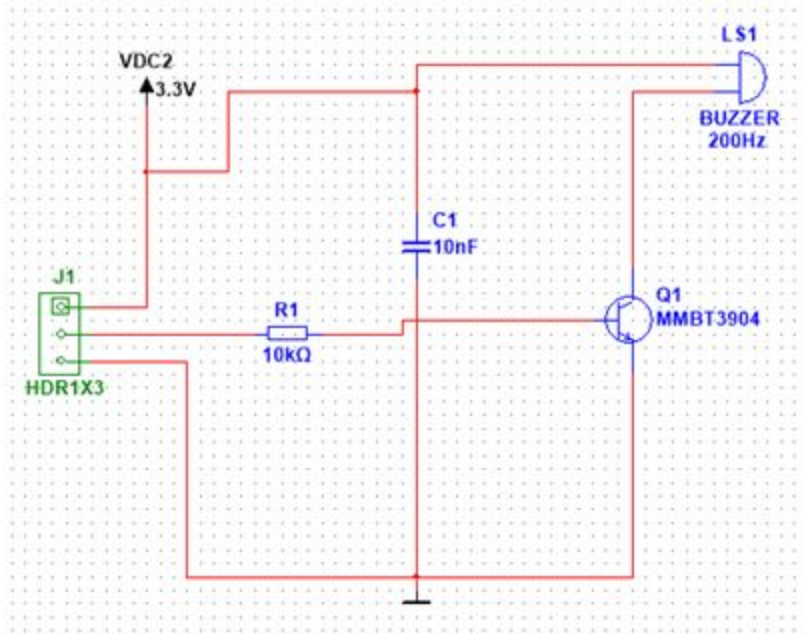
Heartbeat Sensor PCB is used to measure the user's heartbeat rate, it includes the detector and emitter LED, an amplifier which is built base on LM324 and a transistor 2N3904.





Digital Alarm

Digital Alarm is a simple PCB which is built by piezo buzzer and 2N3904 transistor, which is trigger when the heart-rate reach the limitation.



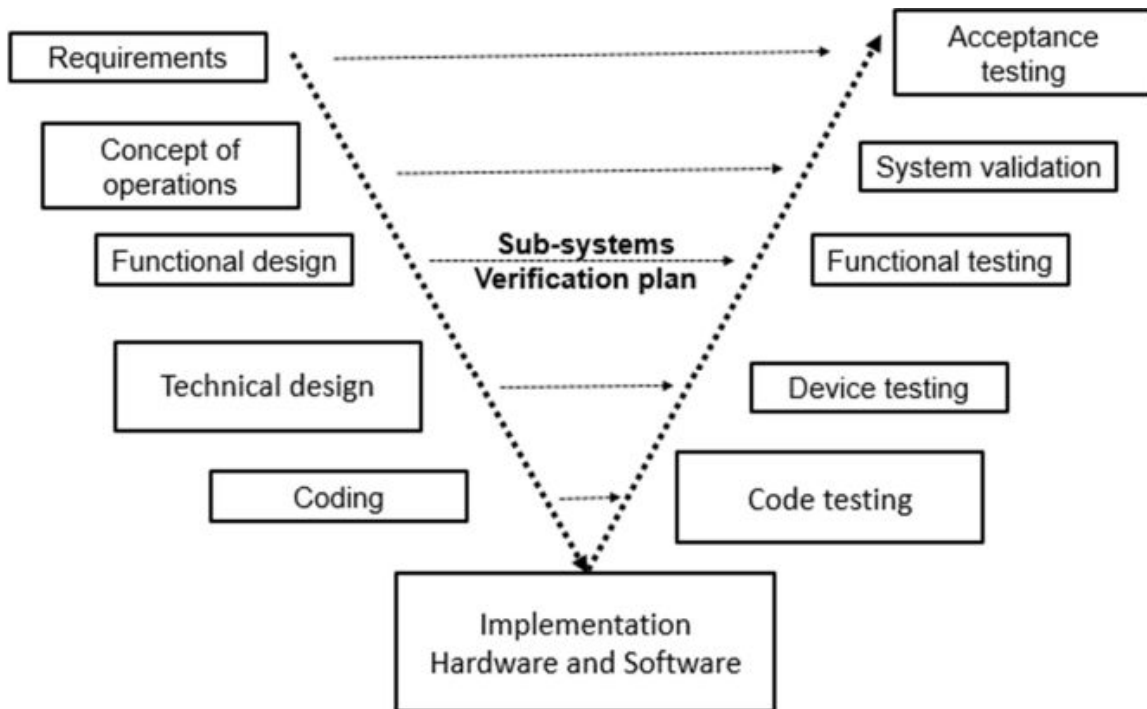
Projectresults:

The result of the product have to meet the client's requirements of both functional and technical:

- Build a mobility device for health assistant.
- Using Echo Dot and ESP32 for implementing the device
- Making the application for user
- The design of divide need to be suitable for the utility.
- The testing of the divide of each sub system.

Testing:

The testing is divided into sub-tests in software part as well as hardware part. The testing following the V-model in the initial project plan:



Acceptance testing:

After decided the concept of the project, the team had a meeting with client to discuss. The purpose of the meeting is the team drew the concept and plan which is decided by all members. Accordingly, client asked and compared with their require whether the concept meet with the requirement.

Functional testing:

The initial requirement of this project is implementation with micropython for ESP32. Therefore, at the beginning, most of the test is implemented with micropython (using uPyCraft and Thonny)

ESP32 testing:

Firstly, esp32 is tested with some simple cases such as blinking LED, alarm with timer counter and buzzer. For the project, the important thing is to connect with wifi as well as to be a wifi access station for Echo Dot.

OLED testing:

Pin connection SSD1306 - ESP32

GND - Ground

VCC - 5V

SDA - Pin GPIO21

SCL - Pin GPIO22

For performance with SSD1306, the library of ssd1306 need to be added. Micropython or Arduino IDE could be used in this case for testing.

Technical testing:

[ESP32 + OLED + Pulse Sensor] (stationary part) and Alexa

The very first test in the functional of the device is the mobility part which consists of ESP32, Alexa and Pulse sensor. To check if this part work or not, there are several check-list to follow:

- ESP32 can connect to wifi
- ESP32 + OLED and pulse sensor: a demonstration of measuring heart rate is required. If the pulse sensor can detect the beat, an attached LED to ESP32 will blink every time a beat is detected and a graph of live heart rate will be drawn in the OLED.
- Alexa can detect ESP32 as a new smart home device by voice controlling with command "Alexa, find/search for new device!"
If a new device (ESP32) is found, Alexa will announce to user the device named (the name can be configured in the code) is found!
- Alexa can turn on/off the device by saying "Alexa, turn on/off + (device name)". In this case, the device should be able to measure heart rate when the device is turned on by Alexa and stop measuring heart rate.

The implementation of stationary part [ESP32 + OLED + Pulse Sensor] and Alexa

As the expectation of the project requirements, micropython is the first priority for implementing ESP32. Due to its lack of libraries to implement and after discussing with software team as well as the client, the team had decided to use Arduino IDE to implement the stationary part. Here are some critical parts that are needed for the project

Libraries

The very first step to make things work is initialization. By adding libraries of ESP32, OLED display, and Alexa. The pulse sensor's library is not necessary to be added as its characteristic (plug-and-play)

```
#include <PubSubClient.h>
#if(true) // includes
#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <splash.h>
#include <stdlib.h>
#endif
#ifdef ESP32
    #include <WiFi.h>
#else
    #include <ESP8266WiFi.h>
#endif
#define SDA_PIN 21
#define SCL_PIN 22
#include "fauxmoESP.h"
```

In this case, the fauxmoESP.h which is the heart of the very first requirement from client. This is the library of Echo dot Alexa. Base on this library, the team could be able to implement the Alexa with the device.

Device setup

Next, the device should need some setup to be able to work properly. Here are the two setup, one for when device is turned on and the other for turn off

```
// Main setup for device
void deviceSetup(){
    // Device setup
    oled.I2Cpins(SDA_PIN, SCL_PIN);
    oled.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialize with
    delay(20);
    // Clear the buffer.
    oled.clearDisplay();

    x = 0;
    clearY();
    pinMode(blinkPin, HIGH); // pin that will blink to
    Serial.begin(115200); // we agree to talk fast
}

void deviceSetupOff(){
    oled.I2Cpins(SDA_PIN, SCL_PIN);
    oled.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialize with
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(0, 47);
    oled.println("Device has been");
    oled.println("Turned off!");
    oled.display();
}
```

As it can be seen that the setup function mainly focus on triggering the I2C pin of the OLED display. In the turning on device setup, there are two functions which are clearDisplay() and clearY() added. These two functions were written to clear the OLED screen (refreshing) everytime the setup function is triggered (everytime the Alexa turn on the device again). clearY() allow the OLED clean all the drew graph before turning on the device again. In other words, whenever Alexa turn on the device, the OLED will be cleaned and all drew graph will be erased. An attached LED is set to HIGH. This allows the LED will blink every beat that the sensor detects the heartbeat. About the setup of turning off device, it is just about clean the OLED then display a small text to announce that device has been turned off.

The interruptSetup

```
void interruptSetup(){
    // Use 1st timer of 4 (counted from zero).
    // Set 80 divider for prescaler (see ESP32 Technic
    timer = timerBegin(0, 80, true);
    // Initializes Timer to run the ISR to sample ever
    // Attach ISRTr function to our timer.
    timerAttachInterrupt(timer, &ISRTr, true);
    // Set alarm to call isr function every 2 millisec
    // Repeat the alarm (third parameter)
    timerAlarmWrite(timer, 1000, true);
    // Start an alarm
    timerAlarmEnable(timer);
}

// set interrupt to turn off measuring procedure
void interruptTurnOff(){
    timer = timerBegin(0, 80, false);
    timerAttachInterrupt(timer, &ISRTr, false);
    timerAlarmWrite(timer, 1000, false);
    timerAlarmEnable(timer);
}
```

This is the function to call all interrupt function during the working time of the device. Every 2 millisecond, an alarm that is set before will trigger the interrupt routine of the device

ISRTr (Interrupt Setup Routine Triggering)

```

// THIS IS THE HW-TIMER INTERRUPT SERVICE ROUTINE.
// Timer makes sure that we take a reading every 2 milliseconds
void ISRTr(){
    Signal = analogRead(35);           // triggered when timer
    Signal = map(Signal, -128, 4095, 16, 890); // read the Pulse Sensor
    sampleCounter += 1;                 // Map the value back to voltage
    // keep track of the time since the last beat
    int N = sampleCounter - lastBeatTime; // monitor the time since

    // find the peak and trough of the pulse wave
    if(Signal < thresh && N > (IBI/5)*3){ // avoid dichrotic noise
        if (Signal < T){                // T is the trough
            T = Signal;                 // keep track of lowest point
        }
    }

    if(Signal > thresh && Signal > P){    // thresh condition help
        P = Signal;                    // P is the peak
    }                                   // keep track of highest point

    // NOW IT'S TIME TO LOOK FOR THE HEART BEAT
    // signal surges up in value every time there is a pulse
    if (N > 250){                       // avoid high frequency
        if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) ){
            Pulse = true;               // set the Pulse flag
            digitalWrite(blinkPin, HIGH);
            IBI = sampleCounter - lastBeatTime; // measure time between beats
            lastBeatTime = sampleCounter;      // keep track of time
            // if this is the second beat
            if(secondBeat){
                secondBeat = false;        // clear secondBeat flag
                for(int i=0; i<=9; i++){    // seed the running total
                    rate[i] = IBI;
                }
            }

            // if it's the first time
            if(firstBeat){
                firstBeat = false;         // clear firstBeat flag
                secondBeat = true;          // set the second beat flag
                sei();                      // enable interrupts again
                return;                    // IBI value is unreliable
            }
        }
    }
}

```

```
// keep a running total of the last 1
word runningTotal = 0;

for(int i=0; i<=8; i++){
    rate[i] = rate[i+1];
    runningTotal += rate[i];
}

rate[9] = IBI;
runningTotal += rate[9];
runningTotal /= 10;
BPM = 60000/runningTotal;
QS = true;
// QS FLAG IS NOT CLEARED INSIDE THIS
}
}

if (Signal < thresh && Pulse == true){
    digitalWrite(blinkPin ,LOW);
    Pulse = false;
    amp = P - T;
    thresh = amp/2 + T;
    P = thresh;
    T = thresh;
}

if (N > 2500){
    thresh = 512;
    P = 512;
    T = 512;
    lastBeatTime = sampleCounter;
    firstBeat = true;
    secondBeat = false;
}

}

} // end isr
```

This is the core of the main function of the device. The routine here is about calculating the heart rate as well as the measured time. So, the point here is whenever the interrupt is triggered (it should be triggered every 2 millisecond), the interrupt routine will do its best which is detecting beat and calculating the heartbeat and also the measuring time.

Alexa and its features

Alexa, the most interesting part in this project which will help client do everything, not with hand, but with their voice. In this project, Alexa was implemented as below:

```
void wifiSetup() {  
  
    // Set WIFI module to STA mode  
    WiFi.mode(WIFI_STA);  
  
    // Connect  
    Serial.printf("[WIFI] Connecting to %s ", WIFI_SSID);  
    WiFi.begin(WIFI_SSID, WIFI_PASS);  
  
    // Wait  
    while (WiFi.status() != WL_CONNECTED) {  
        Serial.print(".");  
        delay(100);  
    }  
    Serial.println();  
  
    // Connected!  
    Serial.printf("[WIFI] STATION Mode, SSID: %s, IP address: %s\n", WiFi.SSID().c_str(), WiFi.localIP().toString().c_str());  
}
```

To detect the device, a function of wifi setup should be declared as above. From that point, all the team has to do next is just defined the WIFI_SSID and WIFI_PASS which is the wifi id and password of the current network that client is using. To be able to realize and connect to device, the device should have a callback function. This function allow software team able to check that the voice commanding work or not.


```

fauxmo.addDevice("Zeus");

// fauxmoESP 2.0.0 has changed the callback signature to add the device_id,
// this way it's easier to match devices to action without having to compare strings.

// Callback device
fauxmo.onSetState([](unsigned char device_id, const char * device_name, bool state) {
  Serial.printf("[MAIN] Device #%d (%s) state: %s\n", device_id, device_name, state ? "ON" : "OFF");
  Serial.println("Device "); Serial.print(device_name);
  Serial.println("Changing output to ");
  //Serial.print(" state: ");
  if (state)
  {
    Serial.println("ON");
  }
  else
  {
    Serial.println("OFF");
  }
  //Switching action on detection of device name
  if ( (strcmp(device_name, "Zeus") == 0) )
  {
    if (state)
    {
      digitalWrite(LED, HIGH);
      digitalWrite(PULSE, HIGH);
      Serial.println(PULSE);
      interruptSetup();
      deviceSetup();
    }
    else
    {
      digitalWrite(LED, LOW);
      digitalWrite(blinkPin, LOW);
      digitalWrite(PULSE, LOW);
      ledcWriteTone(0,0);
      interruptTurnOff();
      deviceSetupOff();
    }
  }
}
}

```

To control a device, the Alexa must recognize the device first. By using `addDevice()` built-in function in `fauxmo` library. By saying very simple command “Alexa, find new device” and after a while, Alexa will respond with an answer with the device’s name that the code defined above. After discovering new device, to be able to control it, the callback function will be used here to active the device’s routines depend on its state. As the above code, when state ON, device interrupt is called and device setup is called as well. It means measuring procedure has begun. If state is OFF, all turn off setups are set.

MQTT + Database + GUI (Data part)

The next test is about the database storage and the window application. The goal of the test is that the window application (GUI) can show the real-time heartbeat in beats per minute (BPM). Moreover, the database which locally store in laptop also should be able to receive the live data through MQTT broker. There are some points that need to be achieved to pass the test:

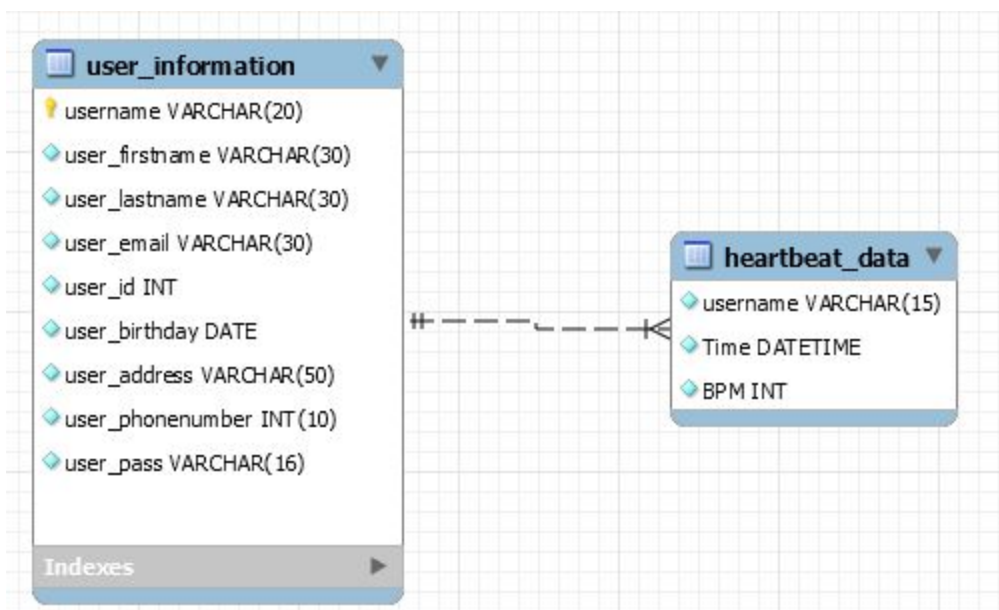
- Firstly, the database should be inserted with some static data which will be displayed on the GUI. If it does, this small test pass!
- The GUI is also need to be able to do some basic function as registering account, display local date time, display user information. After registering, if these functionalities can be done in the GUI, it will be a pass!
- After the stationary part test with Alexa passed, the output which is the heart rate can be collected to the local database through MQTT broker by using `node-Red`. Now software team

should remove the static database, instead of that, check the payload of the node-Red to see if the data is sending or not

- If yes, the message of the payload will be sent every fixed period of time (configurable in the code) and also get the data in the database by reloading it every fixed period of time. From here, the GUI should be able to display the live heartbeat. This means it is a pass!
- If no, this test is a fail and yet there are some issues and it needs to be checked and debug.

Code testing:

GUI and Database testing:



In MySQL, two table is created including user_information and heartbeat_data.

```
CREATE TABLE user_information
(
    user_firstname VARCHAR(30) NOT NULL,
    user_lastname VARCHAR(30) NOT NULL,
    user_email VARCHAR (30),
    user_id INT(1) NOT NULL, -- 1 = doctor, 2 = nurse, 3 = patient
    user_birthday DATE NOT NULL,
    user_address VARCHAR(50) NOT NULL,
    user_phonenumber INT NOT NULL,
    user_user VARCHAR(15) NOT NULL,
    user_pass VARCHAR(16) NOT NULL,
    PRIMARY KEY (user_user)
);

-- INSERT INTO user_information VALUES ();
```

```
CREATE TABLE heartbeat_data
(
    username VARCHAR(15) NOT NULL,
    Time DATETIME NOT NULL,
    BPM INT NOT NULL,
    FOREIGN KEY (username) REFERENCES user_information(user_user)
);
```

The Gui is executed in C# with Window form. Firstly, the connection to the database need to be defined:

```
class Database
{
    MySqlConnection connection = new MySqlConnection("server=localhost;port=3306;username=root;password=Nguyet@anh123;database=healthbot");
    public void initConnection() //initialize db connection
    {
        if (connection.State == System.Data.ConnectionState.Closed)
        {
            connection.Open();
        }
    }
    public void stopConnection() //terminate db connection
    {
        if (connection.State == System.Data.ConnectionState.Open)
        {
            connection.Close();
        }
    }
    public MySqlConnection GetConnection() // get connection
    {
        return connection;
    }
}
```

At first, the static data of heart rate is used to test the display of the window application. Afterthat, we tried to adding the time from local time to the database with the same heart rate value.

username	Time	BPM
a	2019-06-12 16:29:21	30
a	2019-06-12 16:29:31	30
a	2019-06-12 16:29:41	30
a	2019-06-12 16:30:07	30
a	2019-06-12 16:30:17	30
a	2019-06-12 16:30:27	30
a	2019-06-12 16:30:37	30
a	2019-06-12 16:30:47	30
a	2019-06-12 16:30:57	30
a	2019-06-12 16:31:07	30
a	2019-06-12 16:31:17	30
a	2019-06-12 16:31:27	30

After succeed of testing with real-time, we continued adding the data heart rate with random value from 50bpm to 120bpm.

```
class HeartbeatData
{
    public HeartbeatData()
    { }
    public void hearBeat()
    {
        String username = login.username;
        Database db = new Database();
        MySqlCommand cmd = null;
        db.initConnection();
        int bpm;
        Random temp = new Random();
        bpm = temp.Next(50, 120);
        String query = "Insert into heartbeat_data (username, timePeriod, BPM) VALUES (@username, @timePeriod, @BPM)";
        cmd = new MySqlCommand(query, db.GetConnection());
        cmd.Parameters.Add("@username", MySqlDbType.VarChar).Value = username;
        cmd.Parameters.Add("@timePeriod", MySqlDbType.DateTime).Value = DateTime.Now;
        cmd.Parameters.Add("@BPM", MySqlDbType.Int16).Value = bpm;
        cmd.ExecuteNonQuery();
        db.stopConnection();
    }
}
```

username	Time	BPM
a	2019-06-18 01:22:52	75
a	2019-06-18 01:22:54	84
a	2019-06-18 01:22:56	57
a	2019-06-18 01:22:58	90
a	2019-06-18 01:23:00	100
a	2019-06-18 01:23:02	91
a	2019-06-18 01:23:04	82
a	2019-06-18 01:23:06	73
a	2019-06-18 01:23:08	64
a	2019-06-18 01:23:10	55
a	2019-06-18 01:23:12	106
a	2019-06-18 01:23:14	97

At the beginning, there are some problems with real-time data. When the database was running, the window form was not open. The solution is using the timer for display value in window form:

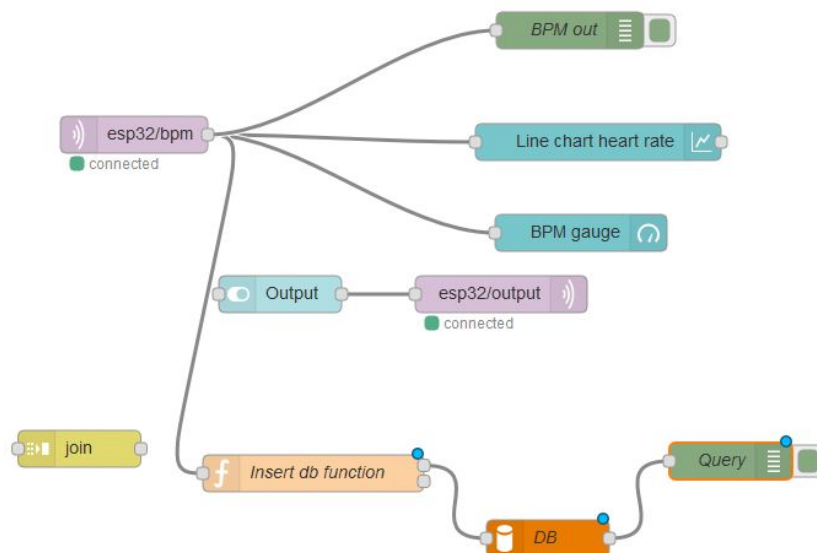
```
void chartTimer_Tick(object sender, EventArgs e)
{
    db.initConnection();
    String query = "SELECT timePeriod,BPM FROM healthbot.heartbeat_data ORDER BY timePeriod DESC LIMIT 1;";
    cmd = new MySqlCommand(query, db.GetConnection());
    MySqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        int bpm = reader.GetUInt16(1);
        DateTime time = DateTime.Parse(reader.GetString(0));
        String stime = time.ToString("HH:mm:ss");
        var series = chart1.Series["BPM"];
        series.Points.AddXY(stime, bpm);
        chart1.ChartAreas[0].AxisX.ScaleView.Position = series.Points.Count - 5;
    }
    db.stopConnection();
}
```

More functions are improved in the window app being choosing the period day in the calendar and the application will show the data of that period. If the heart rate is not steady or in the danger zone, the data is in red text.

```
private void buttonShow_Click(object sender, EventArgs e)
{
    //chart2.Series["BPM"].Points.Clear();
    dataGridView_bpm_show();
}

void dataGridView_bpm_show()
{
    int i;
    String startDay = monthCalendar.SelectionRange.Start.ToString("yyyy-MM-dd");
    String endDay = monthCalendar.SelectionRange.End.ToString("yyyy-MM-dd");
    textBox1.Text = startDay;
    textBox2.Text = endDay;
    String qr = "SELECT username,timePeriod, BPM FROM healthbot.heartbeat_data WHERE username='" + this.textBox_username.Text + "' and date(timePeriod) between '" + startDay + "' and '" + endDay + "'";
    cmd = new MySqlCommand(qr, db.GetConnection());
    try
    {
        adapter.SelectCommand = cmd;
        DataTable dataTable = new DataTable();
        adapter.Fill(dataTable);
        BindingSource bSource = new BindingSource();
        bSource.DataSource = dataTable;
        dataGridView1.DataSource = bSource;
        adapter.Update(dataTable);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

MQTT broker with Nodered



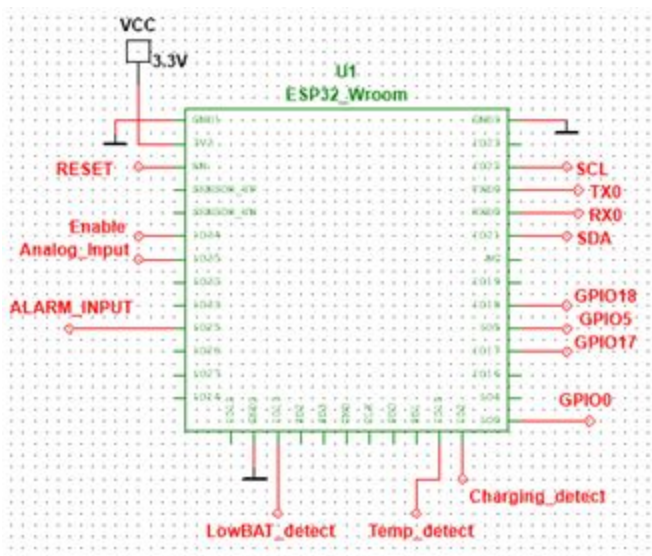
The function is created to pull the data to the Mysql local data:

```
var bpm = msg.payload[0] + msg.payload[1] + msg.payload[2];
msg.topic = "INSERT INTO `healthBOT`.`heartbeat_data` (`username`,`BPM`) SELECT user_user, "+bpm+" FROM user_information";
return msg;
```

Hardware Testing

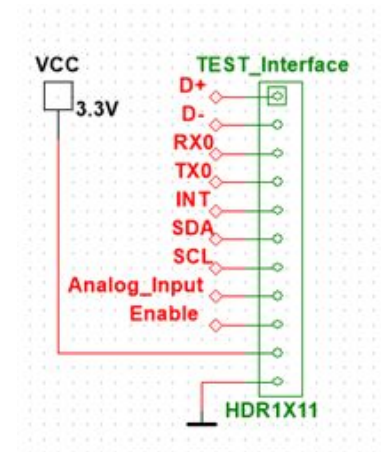
ESP32 mainboard

ESP32: First I had to create an ESP32 footprint on multisim and there are many inputs go into this chip each one of these inputs we must find a way to test it.



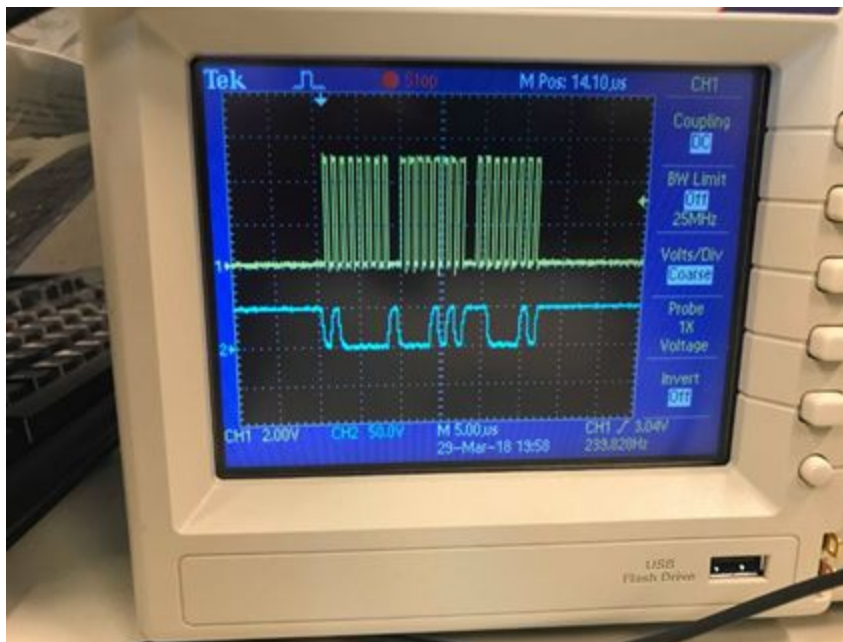
For this part I have create a footprint which called TEST_INTERFACE:

- D+ and D- used to measure the input data come from a micro USB port.
- RX0 and TX0 used for communication between the ESP32 and a computer (also known as a UART or USART).
- SDA and SCL used for testing I2C communication which is the OLED screen is the only component have this type of communication.
- Analog Input used for measurement the sensor input after amplifier the signal.



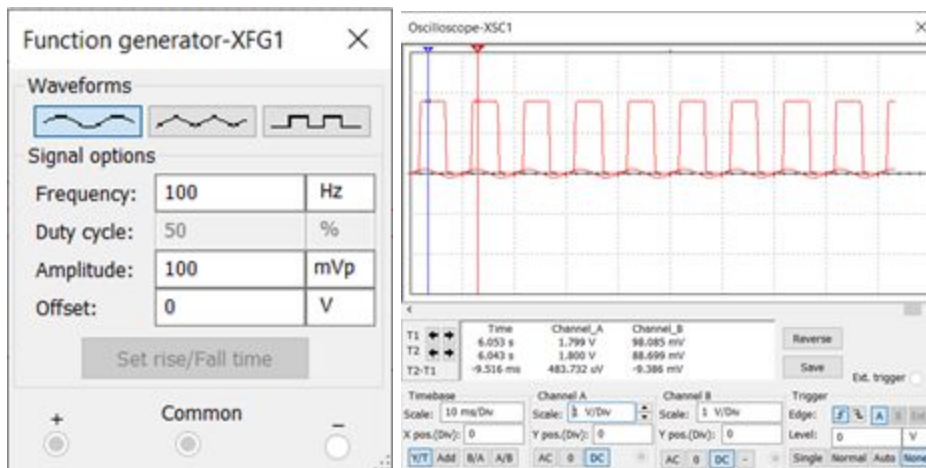
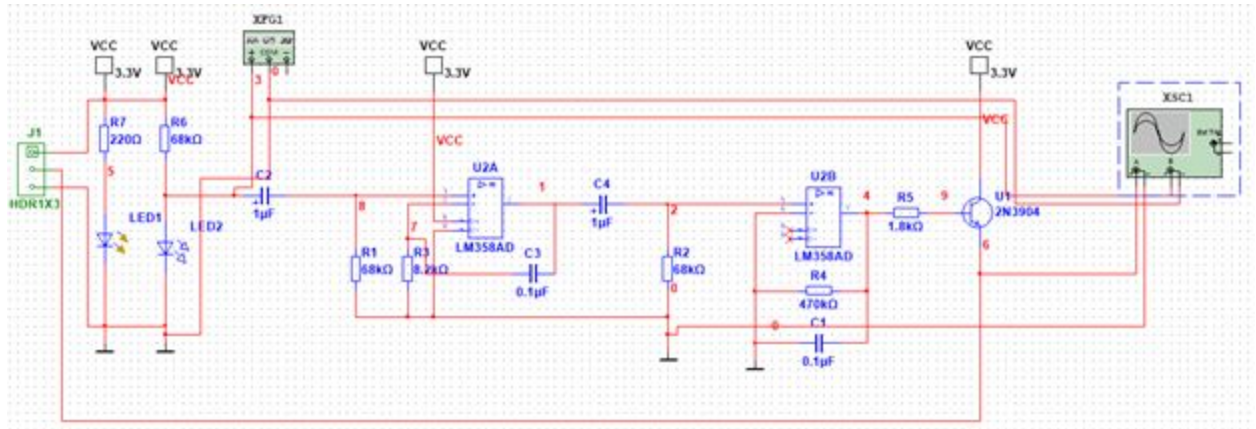
The footprint I make used for connecting the Oscilloscope which can be seen on the graph. For example:

The first signals are UART (RX0 and TX0) and the second is I2C signals.



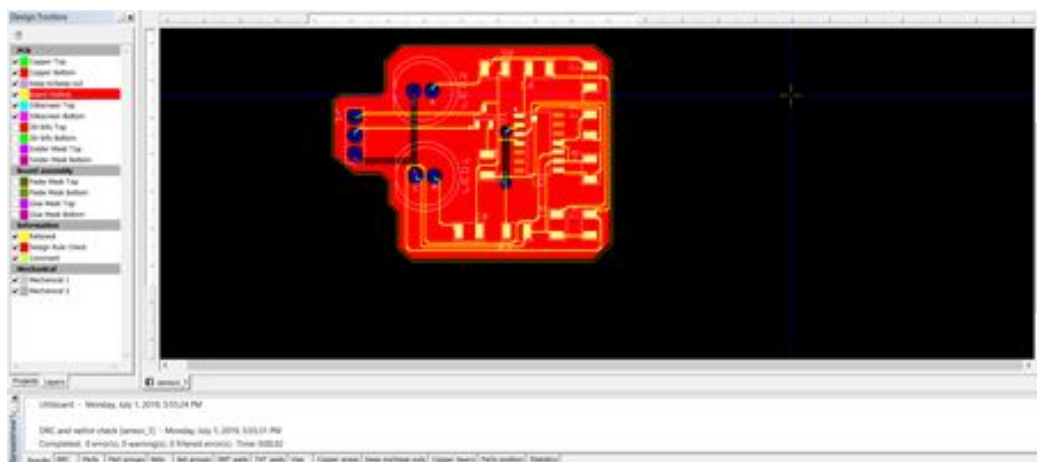
Heartbeat sensor

- For this part , we created a pulse sensor circuit using the Infrared emitter and detector LED and LM358 operational amplifiers chip. Firstly, the heartbeat sensor circuit was created and simulated on Multisim by using a signal generator to create a signal similar with blood circulating signal which given by uses the amount of infrared light reflected.



- As we can see, the input voltage signal have been amplified for 10 times and given a steady baseline for the signal, enlarge the peaks signal and filter out noises from the input signal.

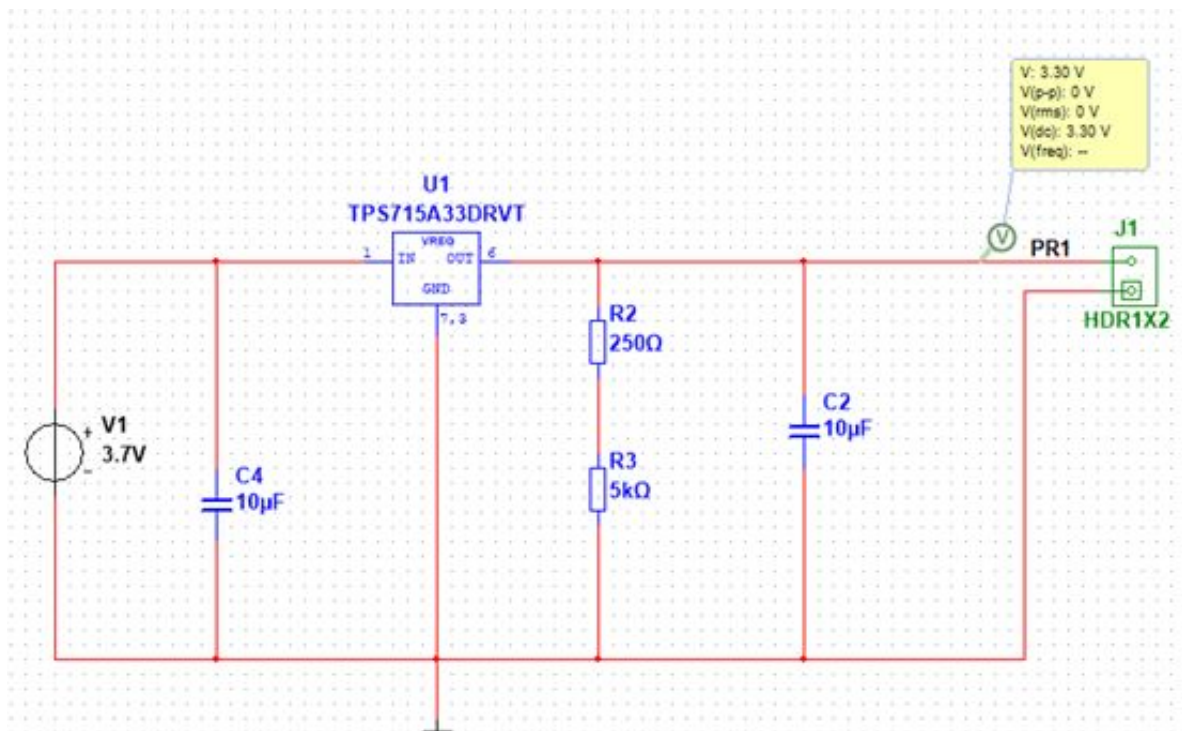
- For the PCB design, pins connection have to follow the EMC rule and also checked with DRC and netlist check.



- After the PCB layout have been created , all components should be soldering and testing for short-circuit in Hardware lab. After that, the Functionals Generator and Oscilloscope have been used to test the output signal. In the following step, the sensor was tested with the ESP codes to show the signal on LCD screens.

Regulator

The testing method was performed on Multisim, As we can see the Voltage input decrease from 3.7 to 3.3 which is important because we expected the V output should equal 3.3 V to operate our system Reflection.



- After simulate in multisim, we have to test the connection in the pcb. We use the tools in the Hardware Lab in school to test the connection in the regulator pcb
- To test the voltage out in regulator, First I give the power supply for the input regulator. The input voltage must be 3.7V to 5V, then I measured the voltage out from regulator by multimeter. If the output voltage is 3.3V, then the regulator had no problem.

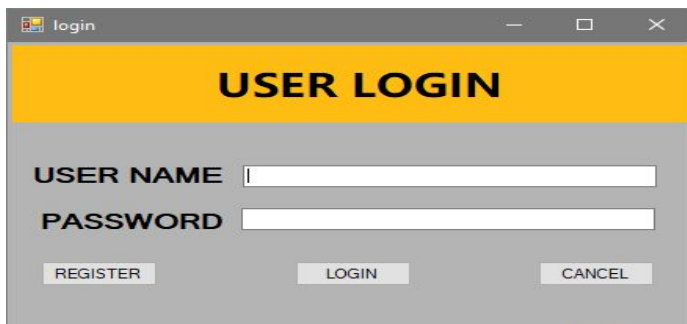
Digital Alarm

The Digital Alarm is used to announce the situation when the user's heart rate becomes high. There are also 3 step to test the digital alarm pcb:

- First, just like the regulator and ESP32. We need to build the circuit on multisim to be sure that the circuit can work. We simulate on multisim by give the signal to the input of the alarm and see the signal in the buzzer.
- The pcb connection must be correct. Same as the regulator pcb, we need to be sure that the connection in the digital alarm is correct by using the tools in the Hardware Lab.
- When the connection test is correct, we give the alarm to ACS guys then they can upload the code for it.

Project end result and discussion

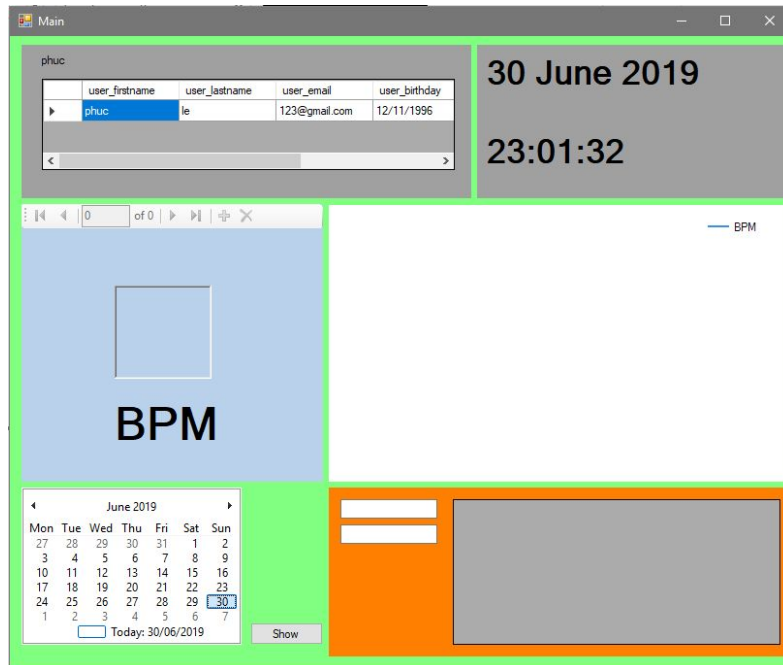
The result of the GUI:



The screenshot shows a window titled "login" with a yellow header bar containing the text "USER LOGIN". Below the header, there are two input fields: "USER NAME" and "PASSWORD". At the bottom of the window, there are three buttons: "REGISTER", "LOGIN", and "CANCEL".



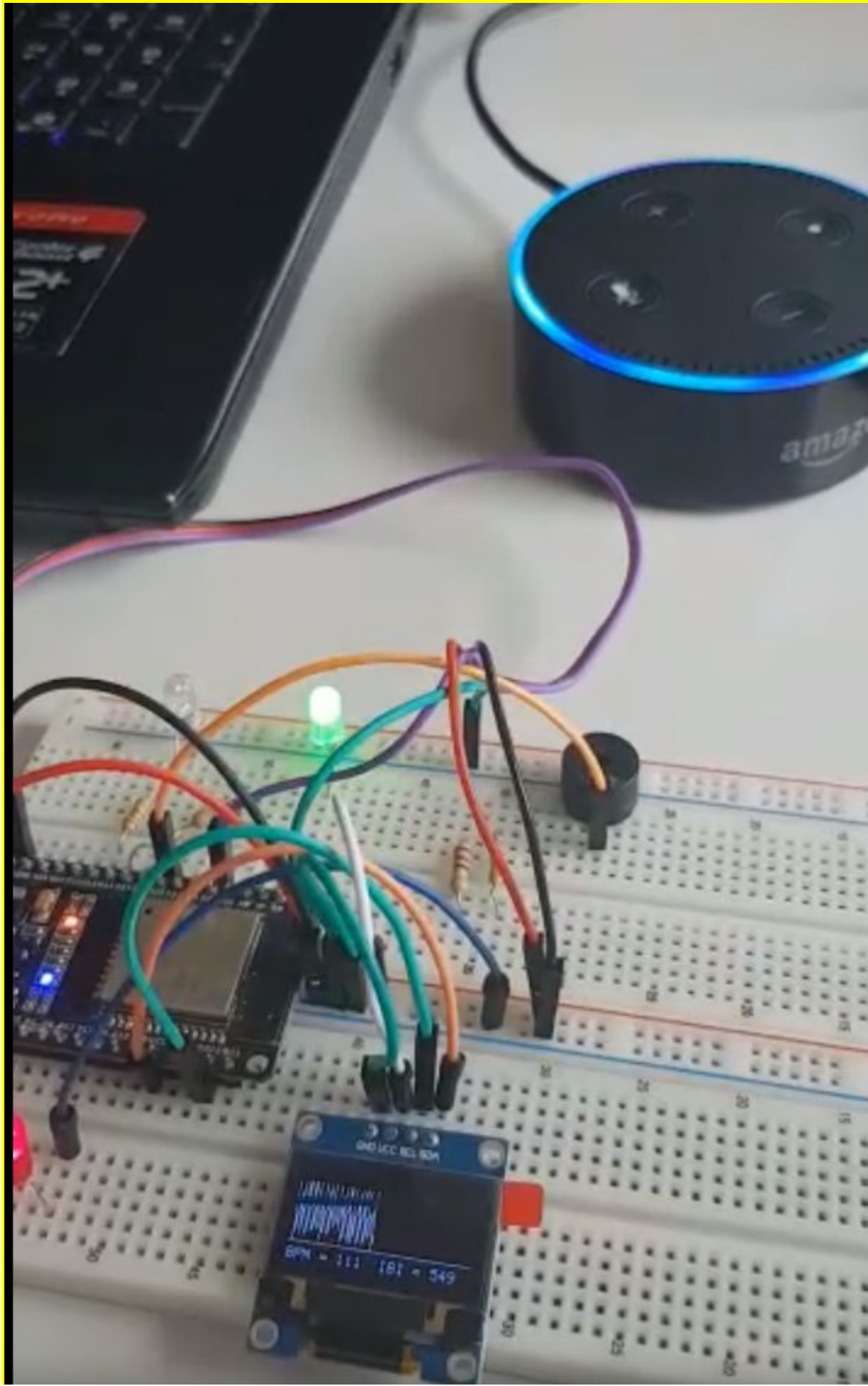
The screenshot shows a window titled "Register" with a yellow header bar containing the text "USER REGISTER". Below the header, there are several input fields: "first name", "last name", "email", "user id", "user birthday", "user address", "user phone number", "user name", "password", and "confirm password". At the bottom of the window, there are two buttons: "REGISTER" and "CANCEL".



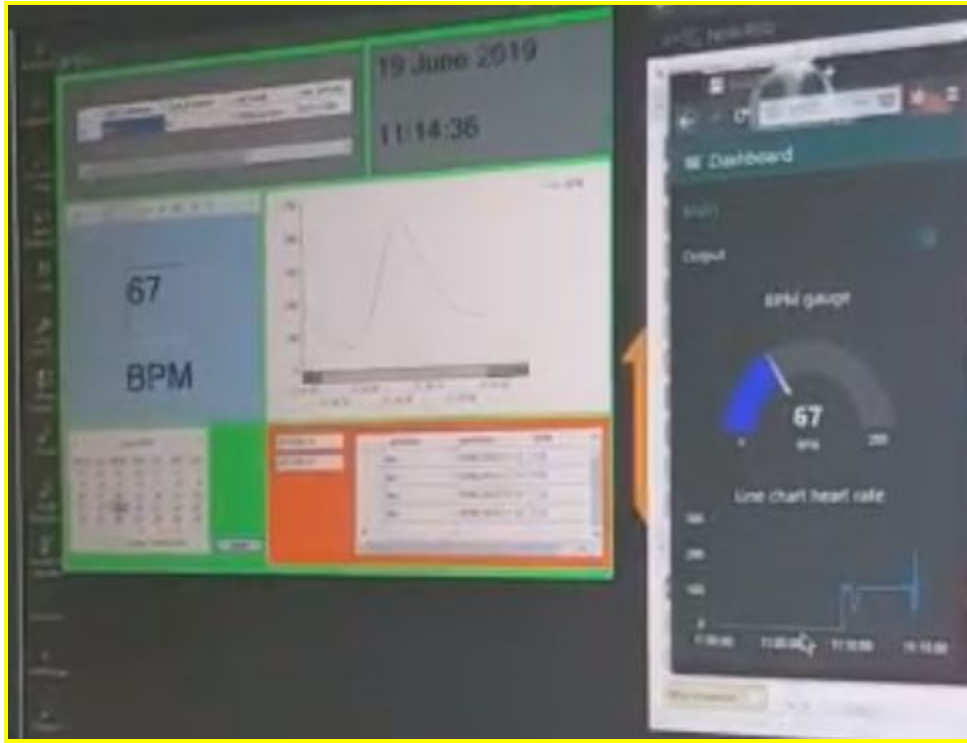
The login form and the register form of the window application. After log on the account, the information of the user will be added into the database table `user_information`

	user_firstname	user_lastname	user_email	user_id	user_birthday	user_address	user_phonenumber	user_user	user_pass
▶	phuc	le	123@gmail.com	1	1996-11-12	Emmastraat	62012345	phuc	le
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

For using alexa, user have to setting Alexa in the Amazon application, then send the command to the Alexa "Alexa, find new device". Alexa receive the command and start to find the device which is health bot. After Alexa detect the device, client can send the voice command "Alexa, turn on/off + (device name)", heartbeat sensor will measure the client's heartbeat. The result of heart rate is displayed on the oled screen by graph and the digital value



Together with the real-time database which storing in the local database as well as showing in the GUI.

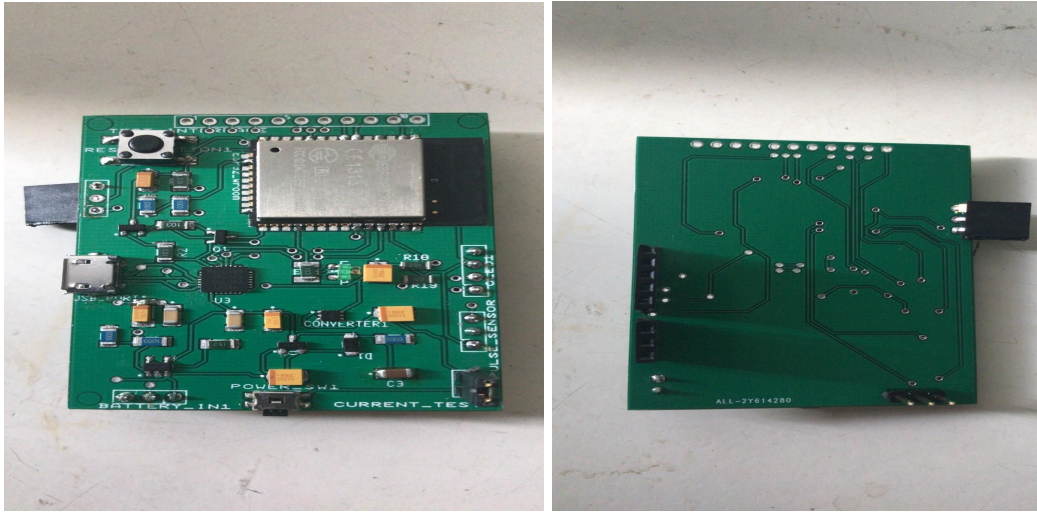


By the end of the demonstration day, the client had given some remarks about the product. The result in general was basically, the product worked as a prototype that was made by software group. All basic functionalities of the device contained and worked properly as well, meanwhile the design and rebuild component of the implementation from hardware group did not work well. In general, the project finished with a success

Hardware Results

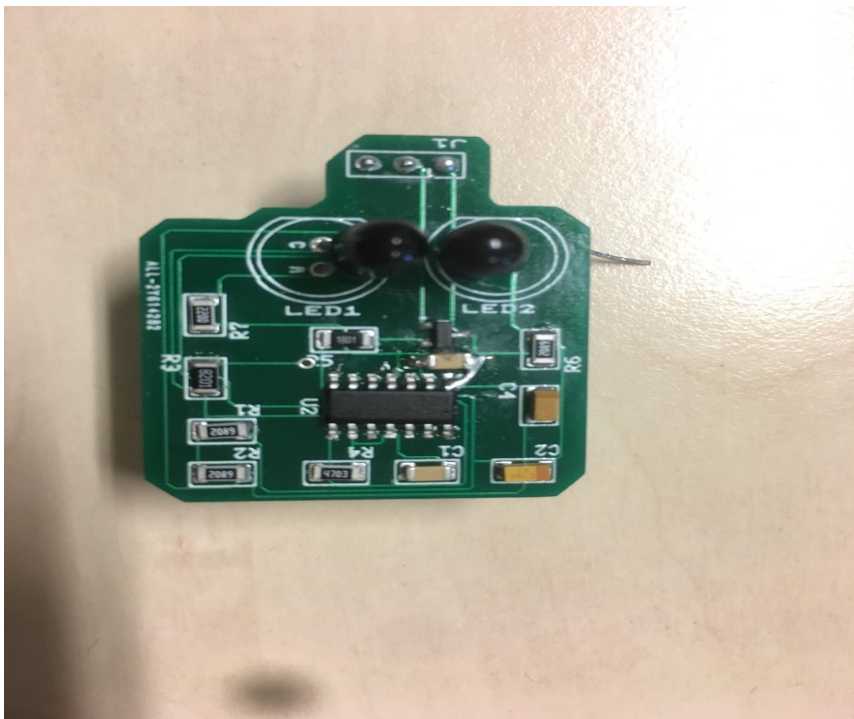
The performance of all the hardware pcbs did not as we expect. There still are some problems with the sensor and ESP32 mainboard.

ESP mainboard



The regulator actually give the output voltage at 3.3V. However, the USB port is broken and the driver in the USB bridge is not compatible with the USB port in PC

Heartbeat Sensor



The Performance of this sensor is not stable, the emitter and detector can not detect the user heartbeat signal.

Digital alarm



The performance of digital alarm is very well. It had been tested with the ACS code and can work without any problem. However, because it was built by piezo buzzer, its sound is not too loud.

Possible recommendations

Since the problem with time management and team professional, the project did not finish completely. There are some functions are able to develop more in the future to meet the full requirement which is setup initialization. For the future development can be improved a number of functions. For instance, the application for mobile to display the heart rate and notificate to the nurse in the emergency case. Further, the database in the cloud can be used for the web view.

References

1. (2019, May 24). Introduction to ESP32. Retrieved from <http://www.iotsharing.com/2017/05/introduction-to-esp32.html>
2. Espressif (n.d.). ESP32 datasheet. Retrieved from https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
3. Random Nerd Tutorials (2019, February). ESP32 pinout references. Retrieved from <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>
4. Johnnyfrx (n.d.). ESP32 With Integrated OLED (WEMOS/Lolin). Retrieved from <https://www.instructables.com/id/ESP32-With-Integrated-OLED-WEMOSLolin-Getting-Star/>
5. Component 101 (2018, July 22). Pulse sensor. Retrieved from <https://components101.com/sensors/pulse-sensor>
6. Yury & Joel, World Famous Electronics llc -. (2018). Works with Everything. Retrieved from <https://pulsesensor.com/>
7. Starting Electronics. (2017, October 17). OLED I2C Display Arduino Tutorial. Retrieved from <https://startingelectronics.org/tutorials/arduino/modules/OLED-128x64-I2C-display/>
8. Eclipse Foundation. (n.d.). Eclipse Mosquitto™. Retrieved from <https://mosquitto.org/>
9. Node-RED. (n.d.). Creating your first flow. Retrieved from <https://nodered.org/docs/tutorials/first-flow>
10. Node Guide. (n.d.). Node RED Programming Guide. Retrieved from <http://noderedguide.com/>