

Design Manual

LIVE RAY



GROUP 19
AMARASINGHE D.I.(E/18/022)
SANDUNIKA S.A.P. (E/18/318)
THARAKA K.K.D.R. (E/18/354)

CONTENTS

1. Introduction	4
1.1. Project overview.....	4
1.2. Useful links	4
2. System overview	5
2.1. High level system	5
2.2. User categories	6
3.Database	6
4. Desktop application.....	9
4.1. Introduction.....	9
4.2.Cloud connectivity	9
4.3. ui & their functionalities.....	9
5.Mobile application	14
5.1. Cloud connectivity	14
5.2. UI and their functionalities.....	14
6.Embedded hardware device.....	17
6.1.Microcontroller	18
6.1.1. Nodemcu ESP8266 CP2102.....	18
6.1.2.MAX30100 Heart-Rate Sensor.....	19
6.1.3.DS18B20 – Temperature sensor	20
6.1.4.NEO6MV2 GPS sensor	21
6.2.Other components	21
6.2.1.OLED DISPLAY	21
6.3.Procedure	22
6.3.1.General Procedure.....	22
6.3.2. Managing the network connection.....	23
6.3.3. Managing the AWS connection	23
6.3.4. Indicators	24
7.CAD model	24

7.1.Dimensions	24
7.2.Components.....	25
7.3.Component placement	26
7.4.Design decisions	26
8.1. Final product	27
9. Testing.....	27
9.1. Software testing.....	27
9.2. API testing.....	28
9.3. hardware testiing.....	29
9.3 Hardware testing	29

1. INTRODUCTION

1.1. PROJECT OVERVIEW

The aim of this project is to provide a system to measure the health parameters of a patient in an ambulance and send real-time data to the hospital through a device. So that hospitals can monitor them via a dedicated web interface and also ambulances can maintain a direct connection with the hospital via a mobile application. Also, some features are included to make this procedure more convenient.

There's no existing system to monitor and send the real-time health parameters of the patients while they are taken to the hospital by Ambulances. In some scenarios, due to the lack of facilities and resources, patients are transferred from one hospital to another hospital. This can be inconvenient for both parties; the patient and the hospital. It might be a risk to the patient's life as well. Even after admitting to the hospital, it may take some time to arrange things for the patient and check the status of the patient. Other than the test results, the Status and the condition of the patient are normally conveyed by a guardian of the patient and there can be reliability issues in such information. This is the problematic situation we're going to address.

1.2. USEFUL LINKS

This is a product designed for the project LiveRay. To get more information about the project, refer to this links.

- [Project Repository](#)
- [Project Page](#)

For Further details [Contact Us](#)

2. SYSTEM OVERVIEW

2.1. HIGH LEVEL SYSTEM

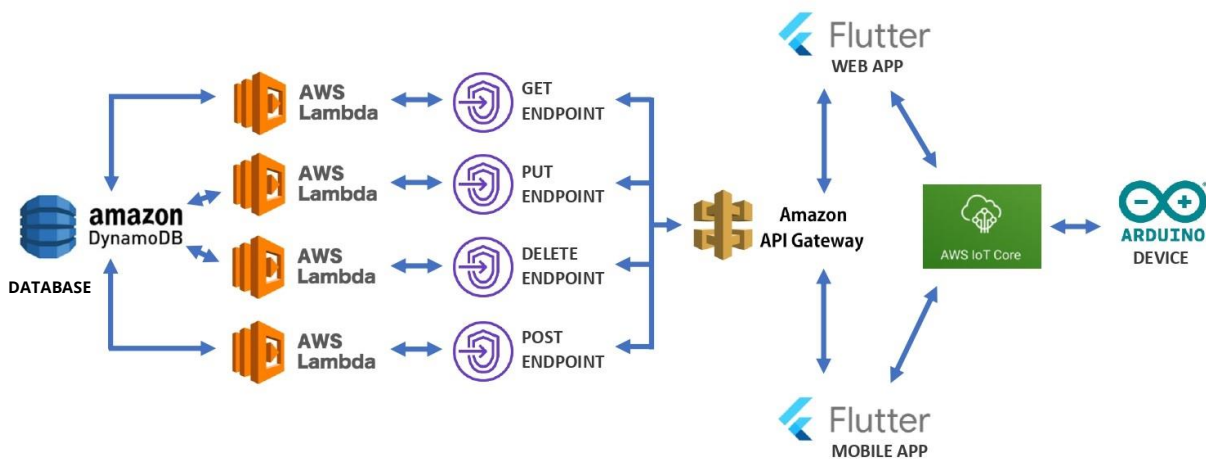


Figure 1 : High Level System

1. Device
2. Mobile application
3. Desktop application

Device is used by the paramedic (inside the ambulance) and it can work as a stand-alone unit. Basic functionalities are implemented using various sensors, buttons, indicators, display etc. Its basically measures patients' health parameters and sends them to a cloud platform.

Mobile application is used as a supportive interface to implement more functionalities in the system like chatting, selecting hospitals etc. in addition to the main functionalities.

Desktop application is used by the hospitals to mainly, monitor the patients data and transfer patients among other hospitals and for chatting.

2.2. USER CATEGORIES

There're 4 main user types can be recognized in this system

1. Paramedics (need credentials to work with mobile app)
2. Devices (need credentials and certificates to connect with the system)
3. Hospital staff (need credentials to work with desktop app)
4. Admins (need credentials to add/remove users and hospitals)

3.DATABASE

In this system two main information must be stored.

1. User Credential Details
2. Hospital details

User Credential details are useful when login into the system. Those credentials belong to 4 categories of users.

- Hospitals
- Devices
- Admins
- Paramedics

User Credential Details

<u>ID</u>	password
-----------	----------

<input type="checkbox"/>	UserID	▼	Password
<input type="checkbox"/>	A003		password12
<input type="checkbox"/>	D004		password4
<input type="checkbox"/>	H004		password4
<input type="checkbox"/>	D001		password14
<input type="checkbox"/>	H001		password1
<input type="checkbox"/>	D003		password3
<input type="checkbox"/>	D100		pwd100
<input type="checkbox"/>	D005		password5
<input type="checkbox"/>	H002		password2

Figure 2 : Database table for Credentialas

Hospital details

<u>Hospital ID</u>	Contact No	Hospital Name	Latitude	Longitude
--------------------	------------	---------------	----------	-----------

<input type="checkbox"/>	HospitalID ▾	ContactNo ▾	HospitalName ▾	Latitude ▾	Longitude
<input type="checkbox"/>	H007	0212061412	Mullaitivu Distr...	9.2269065	80.7983009
<input type="checkbox"/>	H010	0632222261	Ampara Distric...	7.2990188	81.6872563
<input type="checkbox"/>	H004	0252264261	Kekirawa Distri...	8.0438335	80.5937109
<input type="checkbox"/>	H011	0332222261	Gampaha Distri...	7.0910635	79.9987803
<input type="checkbox"/>	H001	0112548755	Colombo Gene...	6.918923	79.8658637
<input type="checkbox"/>	H002	0812571344	Kadugannawa ...	7.2612606	80.3936769
<input type="checkbox"/>	H006	0272222261	Polonnaruwa G...	7.9432059	81.0073628
<input type="checkbox"/>	H003	0662222261	Matale District ...	7.4610981	80.6225245

Figure 3 : Database table for hospitals

Mainly 3 AWS services are useful in the data retrieving & storing process in the database.

1. For the Database Amazon Dynamo DB which is a fully managed, serverless, key-value NoSQL database has been used. It consists of built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data import and export tools.
2. Functions to access the database are implemented inside AWS Lambda which lets to run code for virtually any type of application or backend service without managing a server.
3. To create RESTful APIs which enable real-time two-way communication applications Amazon API Gateway is used. (Here REST APIs act as the front door for applications to access data, business logic, or functionality from the backend services)

4. DESKTOP APPLICATION

4.1. INTRODUCTION

Front End of the application is build up with Flutter. You can refer [flutter documentation](#) to get a complete idea of that.

4.2.CLOUD CONNECTIVITY

Each hospital and device have subscribed to a relevant topic that was created in the AWS IoT Core with their names. Each device has a corresponding mobile application. On the other hand, mobile applications also have subscribed to those topics relevant to its device. Once a client sends a message to a particular topic corresponding subscribers will be notified and data will be updated on the desktop and mobile application interfaces.

4.3. UI & THEIR FUNCTIONALITIES

1. Login validation is done by the application itself
(Click [here](#) to refer to the code)

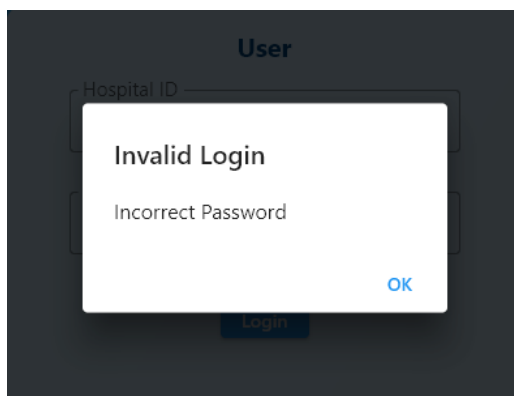


Figure 5 : Error messages for invalid logins

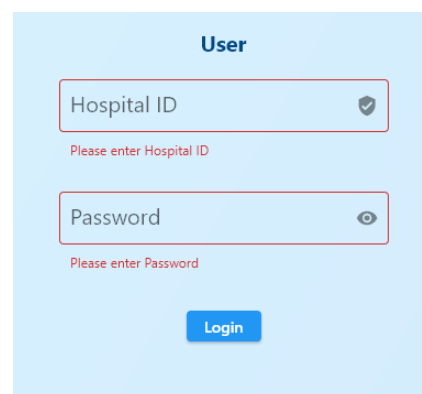


Figure 4 :Error messages for empty values

2. Then it leads to the main dashboard of the application

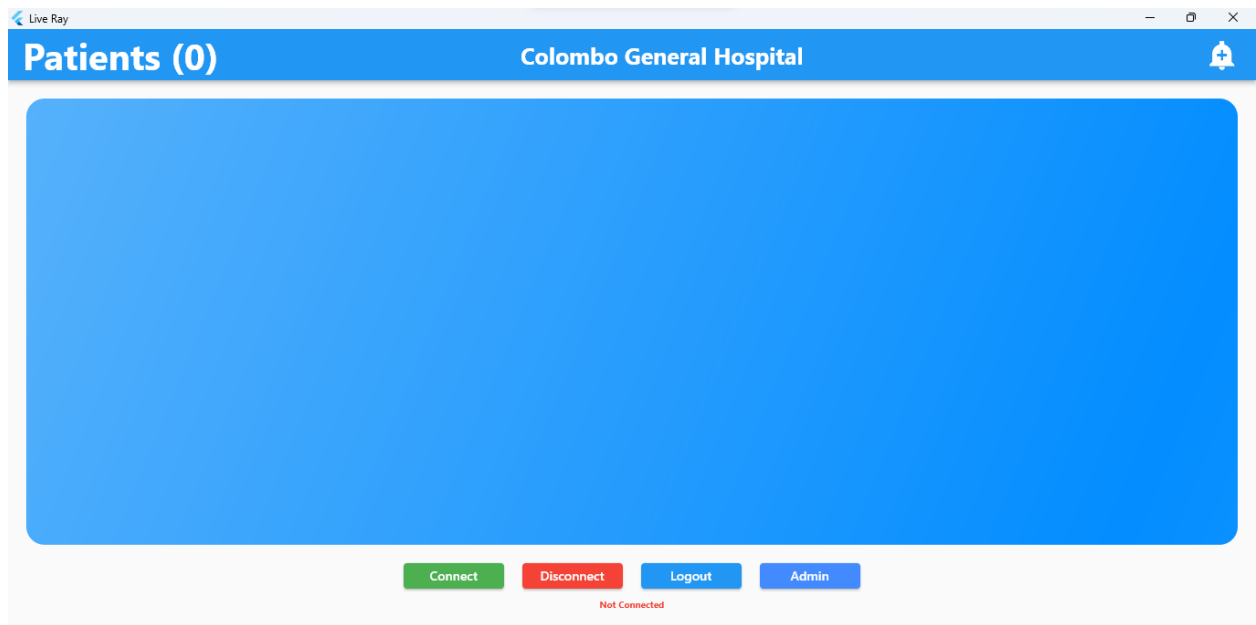


Figure 6 : main dashboard

AWS connection is established using the **Connect** button. That method is created as a part of the interaction with the MQTT protocol. Here we call the connect function. And then we change the state by setting the new connection State. If some of these two processes fail, the disconnect function is called.

3. Create patient cards and keep updating them

Once a mobile application or a device itself sends a message (to connect) with the topic of a particular hospital name, the corresponding hospital will be notified. Then a patient card will be created and appeared on the hospital dashboard with that requested device data.

And inside the card, the data will be kept updating with the sending messages (patient's health parameters and location coordinates are sent as messages) of that particular device.

(Click [here](#) to refer to the code)

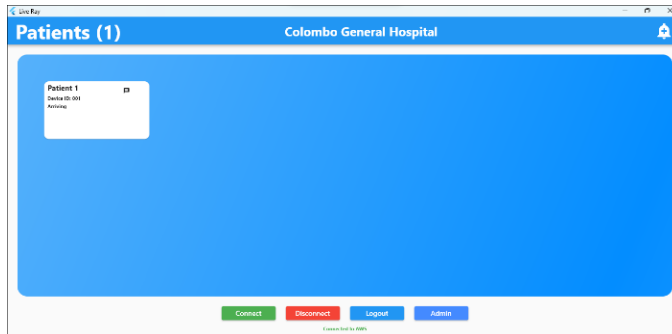


Figure 8 : Patient cards

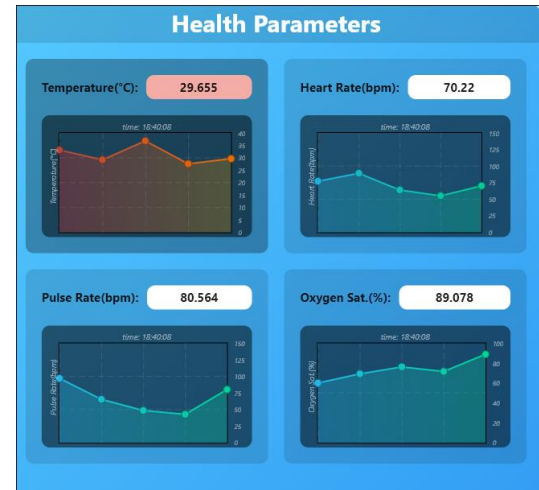


Figure 7 : Health parameter display



How to represent patient's health parameters graphically?

For this we've used [fl_chart 0.55.2](#) package

We can highly customizable line charts in our UI and update data on it easily.

4. Update the current location of the ambulance

For this, we've used **flutter_map 3.1.0** plugin that allows to display the map in Our applications. It is similar to Some other mapping libraries.

It gets the coordinates from the device via cloud services and

Accordingly update the current location.

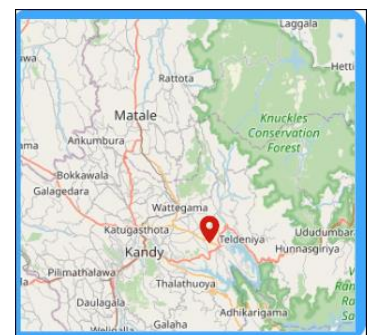
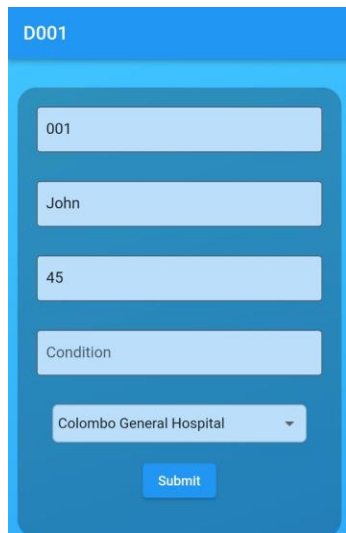


Figure 9 : Location tracking

5. Update the patient's personal details through the mobile application

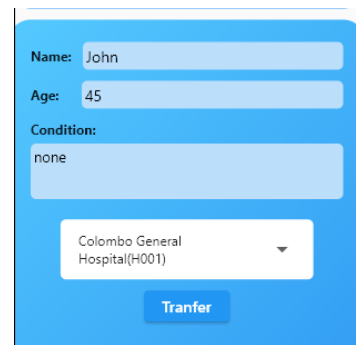
Patient's personal data can be updated only using the mobile application. Each device has a dedicated mobile application and it also has subscribed to the same hospital to which the device keeps sending messages. So mobile application itself can send a message with patient's data to the topic under that hospital name. Corresponding hospital can be displayed that data on the following section.

(Click [here](#) to refer to the code)



The mobile app interface shows a form titled 'D001' with the following fields: '001' (device ID), 'John' (Name), '45' (Age), 'Condition' (empty), and 'Colombo General Hospital' (selected from a dropdown). A 'Submit' button is at the bottom.

Figure 10 : mobile app entering data



The desktop app interface displays the entered data: Name: John, Age: 45, Condition: none, and Colombo General Hospital(H001) (selected from a dropdown). A 'Transfer' button is at the bottom.

Figure 11 : Entered data shown in desktop app

6. Transfer patients between hospitals

Transferring patient to another hospital can only be done by a hospital itself .As an example;

If hospital 1 wants to transfer a patient (with device id 001) to hospital 2, then hospital 1 will send a request as a message (including the device id) to the topic under hospital

2. If hospital 2 accepts it, message will be sent to the topic of device 001 to notify the device and mobile app, subscribed to it . Then the device and the mobile app will subscribe to the new topic under hospital 2 and data flow will now happen among them.



Figure 13 : Hospital 1 dashboard transfer request

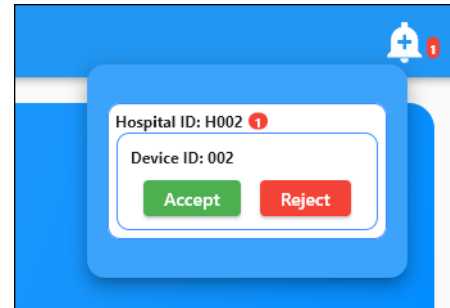


Figure 12 : Hospital 2 notifications

7. Chatting between hospital and device

Two-way chatting is only available between the mobile app and desktop app.

Device can only see the messages but can't reply to those. Here, chatting is also implemented using MQTT protocol and the above mentioned pub-sub pattern.

8. Admin features

(Click [here](#) to refer to the code)

Admins can login to the system with their user id and password with admin privileges. Validation is done within the application itself.

 A screenshot of a web form titled 'Add New Hospital' and 'Remove Hospital'. The 'Add New Hospital' section has five input fields: 'Hospital ID', 'Hospital Name', 'Contact No', 'Latitude', and 'Longitude'. Below these fields is a green 'Submit' button. The 'Remove Hospital' section has one input field: 'Hospital ID'.

Figure 15 : Add/remove hospitals

 A screenshot of a web form titled 'Add New User' and 'Remove User'. The 'Add New User' section has two input fields: 'User ID' and 'Password'. Below these fields is a green 'Submit' button. The 'Remove User' section has one input field: 'User ID'. Below this field is a red 'Remove' button.

Figure 14 : add/remove users

5. MOBILE APPLICATION

5.1. CLOUD CONNECTIVITY

After login to the mobile application it can connect with a device by submitting its device ID. Here it checks particular device with that entered ID is active. While submitting the device ID it automatically / user intentionally send message to a hospital. Therefore, by now mobile application has subscribed to both device and the hospital. Once the device starts sending data mobile application is also getting updated since it has subscribed to a hospital.

5.2. UI AND THEIR FUNCTIONALITIES

1. Login validation is done by the application itself
(Click [here](#) to refer to the code)

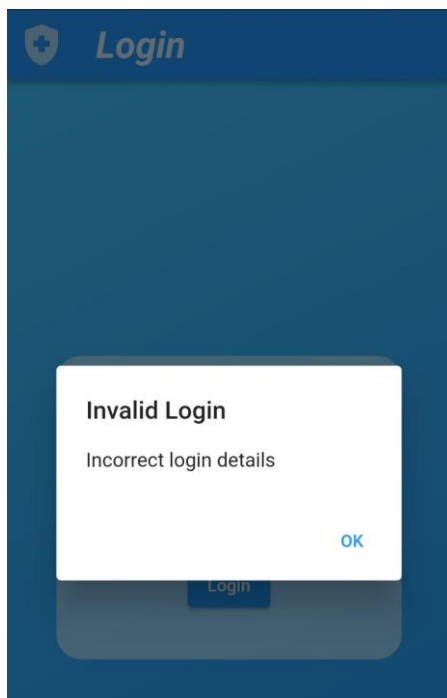


Figure 17 : Error message for invalid logins

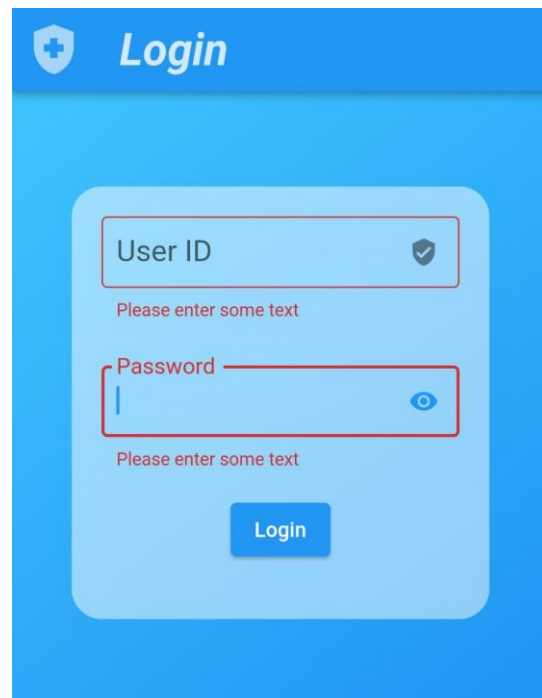


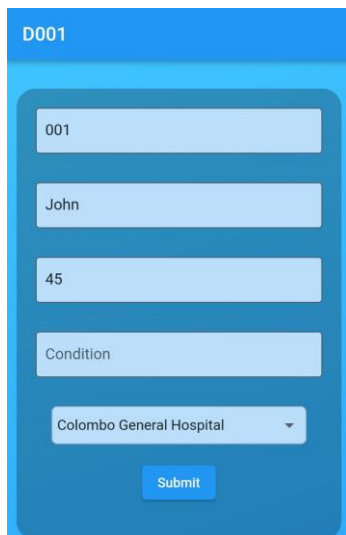
Figure 16 : Error message for empty sections

2. Section to enter patient's data

Here the required data to start a ride are Device ID and the hospital name. After entering those two data and submitting means you're establishing a connection with those two.

Data (Name, age, condition) can be sent anytime in any number of time if the connection is ongoing.

Even after starting a ride through the device itself we can send these data. For that we have to login to the mobile application by giving that device ID.



The screenshot shows a mobile application interface for entering patient data. At the top, there is a blue header bar with the text 'D001'. Below this, there is a white rounded rectangle containing five input fields. The first field contains '001', the second 'John', the third '45', and the fourth 'Condition'. The fifth field is a dropdown menu currently showing 'Colombo General Hospital'. Below these fields is a blue button labeled 'Submit'.

Figure 18 : Personal data entering section

3. Section to display real-time health parameters of the patient

Mobile application is subscribed to a topic of hospital. Whenever the hospital (topic) get data mobile application will notified and display those data on this section. This section keeps updating those data until the ride ends.

(Click [here](#) to refer to the code)

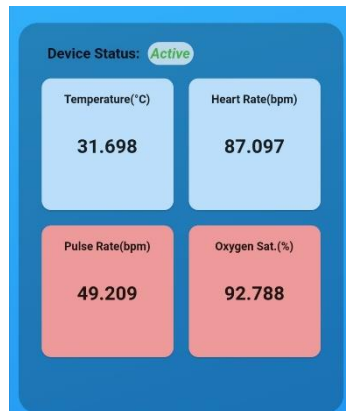


Figure 19 : Health parameter updating section

4. Chat section

Chat section works as same as the chat section in desktop application.

(Click [here](#) to refer to the code)

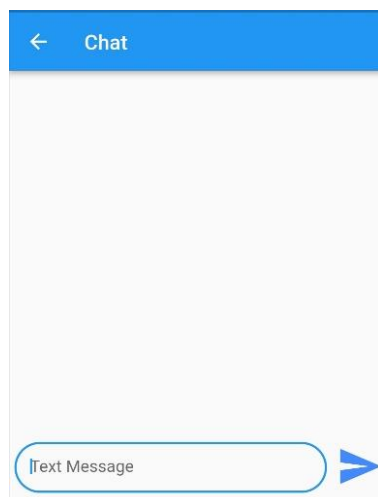


Figure 20 : Chatting page

5. Stop the device

To stop the current ride this button can be pressed. Then the relevant device and the hospital will be notified. Just logging out from the mobile application won't stop the current ride.

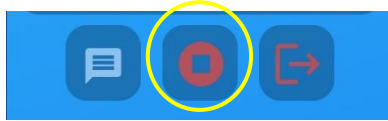
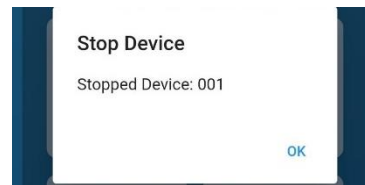


Figure 21 : Stop button



6. EMBEDDED HARDWARE DEVICE

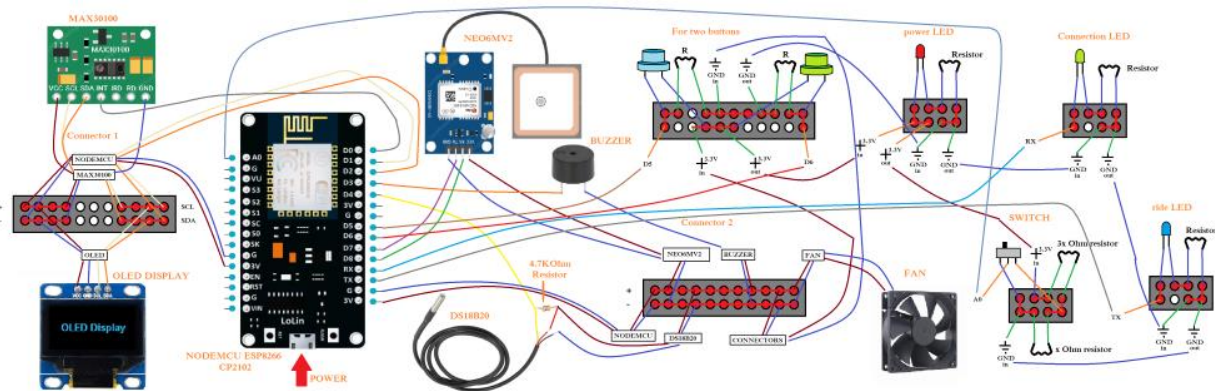


Figure 22 : Hardware connections

Microcontroller : nodemcu ESP8266 CP2102

Other : OLED display 0.96"

Sensors : MAX 30100

Buzzer

DS18B20

CPU Fan

NEO6MV2

Push buttons

LEDs

6.1.MICROCONTROLLER

6.1.1. Nodemcu ESP8266 CP2102

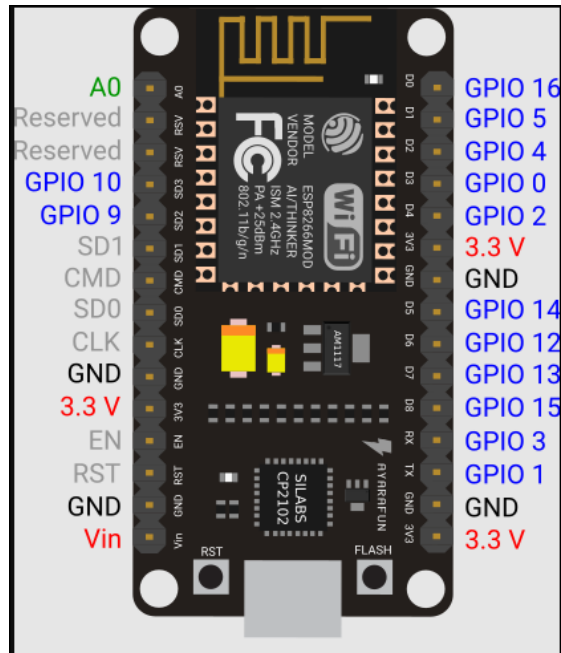


Figure 23 : Pin configuration of nodemcu

First we need to import following libraries before working with nodemcu:

```
#include "FS.h"

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
```

NodeMCU has this Wi-Fi module built in. So, nothing to worry. But there are some important points.

- Wi-Fi module uses analog pin, A0. We can use A0 for dedicated work. But if we use the pin frequently, It will make troubles to the internet connection. Usually, we can read inputs using A0 once in 2seconds. If the frequency is more than that, most of the time internet connection breaks down.

There are three **Wi-Fi configuration styles**.

- Other than the one we used, other two can be used to store multiple ssids and to make the device a Wi-Fi resource.
- The one we used can manage a single Wi-Fi connection.
- If we want to change the connection, the current connection has to be stopped and start a new connection again.

When selecting the microcontroller we've chosen nodemcu ESP8266 CP2102 and we can see quite similar microcontroller 'nodemcu ESP8266 CH340' . So why have we chosen type CP2102 over type CH340 ?

- They're different types of USB interface chips. They translate the USB signals from your computer to/from serial (TTL) signals that the processor can understand.
- CH340 is very cheap, and some people seem to have problems with the drivers, especially on MacOS.
- CP2102 seems to be more reliable
- If the difference in price isn't too much it's probably sensible to go with CP2102.
- It is the smallest one so that it is easy to use within our device

For learn more about nodemcu ESP8266 click [here](#).

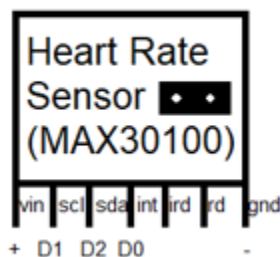
6.1.2.MAX30100 Heart-Rate Sensor

To program MAX30100 You need to include following Libraries:

- MAX30100_PulseOximeter.h
- Wire.h



Its pin configuration is as follows.



VIN, SCL, SDA, INT and GND pins are used.(corresponding D1, D2, D0 pins of nodemcu are marked in Figure 23

I2C communication is used. (SCL, SDA).If sensor works correctly, red led on the sensor will lit up.

There are some important points regarding MAX30100 sensor.

- Updating frequency must be larger than 100Hz. It is directly written by the manufacturer. With the load we have to do in the process, it is really hard to update the sensor as frequently as that. Even using interrupts it cannot be done because some unit statements in the code takes longer time than 10ms to execute. So, the way we can follow is, once in the cycle sensor has to be begin again and update. If it isn't updated properly the values will be remaining in the same value.

6.1.3.DS18B20 – Temperature sensor

Libraris used:

```
#include <OneWire.h>
```

```
#include <DallasTemperature.h>
```

This is a waterproof sensor and the best sensor among the sensors we used to this product. You don't have to be worry much about this one. If connections are done properly. it will give the really accurate readings. Accuracy of the sensor is 0.01'C.

Pin configuration are as below.

However, we have to put a pullup resistor of 4.7 kΩ between the YELLOW and RED wires.

There is a useful point that if it is needed, number of temperature sensors can be connected to the same pin in the microprocessor and can read values without any trouble.



6.1.4.NEO6MV2 GPS sensor

Libraries used:

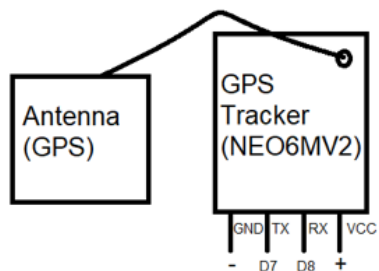
```
#include <TinyGPS++.h>
```

```
#include <SoftwareSerial.h>
```



This sensor track the location and gives us the latitude, longitude and the altitude. This time,altitude is not used by us.

The pin configuration is as below.



There is an antenna with the sensor. It has to be connected. However, when this gives readings, it gives really accurate readings. But unfortunately, it doesn't always work. This sensor has an inbuilt battery. It usually is not charged and it has been a common problem. If you buy a new neo6mv2 sensor and it doesn't work don't be afraid because it is the nature of the sensor. Also, nothing can be done to this, except wait for it to work. Even though it works, it again doesn't work after some time. If it works, red led on the sensor will blink.

6.2.OTHER COMPONENTS

6.2.1.OLED DISPLAY

Libraries used:

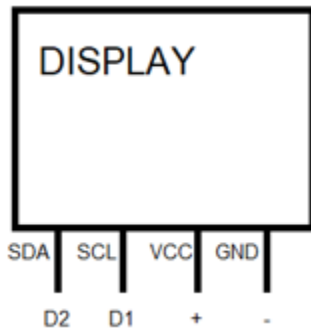
```
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_SSD1306.h>
```



Inside the library files, the size of the display has to be set. Most of the time its default setting is to the 128*32 display.

The pin configuration is as below.



I2C communication is used. Therefore, multiple displays can be connected to the same 2 pins, D1 and D2. Nothing much important about the display. If a single display is not enough, multiple displays can be used.

Note : Sample codes for each component can be downloaded from the [SampleArduino](#) file.

6.3.PROCEDURE

6.3.1.General Procedure

- Data from the sensors is sent once in 2 seconds.
- Internet connection is created.
- Connected with AWS using the certificates. Certificates are stored in the data file.
- After establishing both the internet connection and the AWS connection, it is waiting for a command from a device (button) or the mobile app.
- Once there is a command to start a ride from the mobile app, it sends a message to the mobile app informing that the device is 'Active'. Then it reads values from sensors and publishes them to a subtopic of the device ID under the hospital's name once in 2 seconds. Name and the id of the hospital received with the request.
- If the device itself starts the ride, the ride only can be started at the default hospital. When the ride is started it is informed through the display with a rough beep alert.
- Location and Temperature are measured at the instant of making the data chunk since they usually give accurate readings. But the heart rate sensor doesn't give accurate readings.
- Then, once data is sent all the measurements sensor is read when there is a beat are stored.

- Then at the instance of creating the chunk, the average is calculated and sent. Usually, the sensor read a value '0' even though the finger is at the correct position. since these readings can affect the average badly, they are neglected. Only if all the readings are '0' within the period, '0' is considered as the measurement.
- Once the data is sent, it will be shown on the display with a beep sound.
- If it is requested to change the hospital, the topic we are sending data should be changed and immediately it will be informed through the display with the rough beep sound.
- At the end of the ride, the request to end the ride can come from either device or the mobile app.
- Once it is received, sending the data should be stopped, it should be informed using the display with the rough beep sound and the configurations should be reset to the default.

6.3.2. Managing the network connection

- We usually manage two SSIDs. Once a connection is established, the connection should be checked frequently. If the established connection is failed, the device should try to connect with the other SSID. If it is not there, should try to connect with the last SSID again. The connection can be changed intentionally too. There is a dedicated button for that (YELLOW). If it is clicked, the current connection should be stopped and should try to connect with the other SSID. If it doesn't work back again to the other SSID.
- Even though during a ride this whole process should be available. Within that period data will not be sent. All these connection changes should be displayed through the display.

6.3.3. Managing the AWS connection

- There are certificates which can be used as credentials to log in to the AWS thing. They are stored in the data file. After the internet connection is established, using them, the device should connect with AWS. It should be shown on the display.
- There is a button to validate the device (GREEN). We do this validating process using the AWS certificates. Once the button is pressed, AWS connection should be dropped. Once it is pressed again, the device tries to connect again with the AWS using certificates. If the device doesn't have certificates, it will fail to connect.
- There is reason to have this option. When we power off the device it doesn't disconnect the AWS established connection since it is just like a sudden power failure. There is no process to terminate the program. Then when we power on the device again, it doesn't need those certificates to log into the AWS. Even when device doesn't have certificates, device can connect with AWS. So, to avoid this to happen, this option is created.

- But if the user can have the good habit of disconnecting with the AWS using the GREEN button, these things are not needed. But, we can't guarantee that user will do this.

6.3.4. Indicators

There are three indicators.

1. RED led will lit up when the device is powered on.
2. GREEN led will indicate the status of the connection. When device hasn't connected with internet, led will not lit up. If device only has connected with internet, but not with the AWS, brightness of the led will low. LED will lit up brightly when there are both connections.
3. BLUE led will indicate whether there is a ride or not. If the connection is failed during a ride, brightness will go down and once the connection is restored, it will light up with the normal brightness.

7.CAD MODEL

7.1.DIMENSIONS

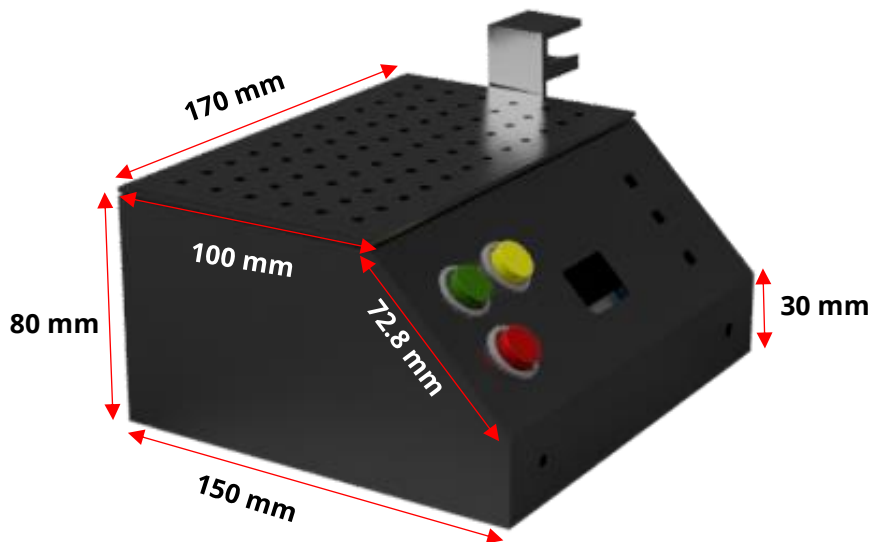


Figure 24 : Dimension of the Device

7.2.COMPONENTS

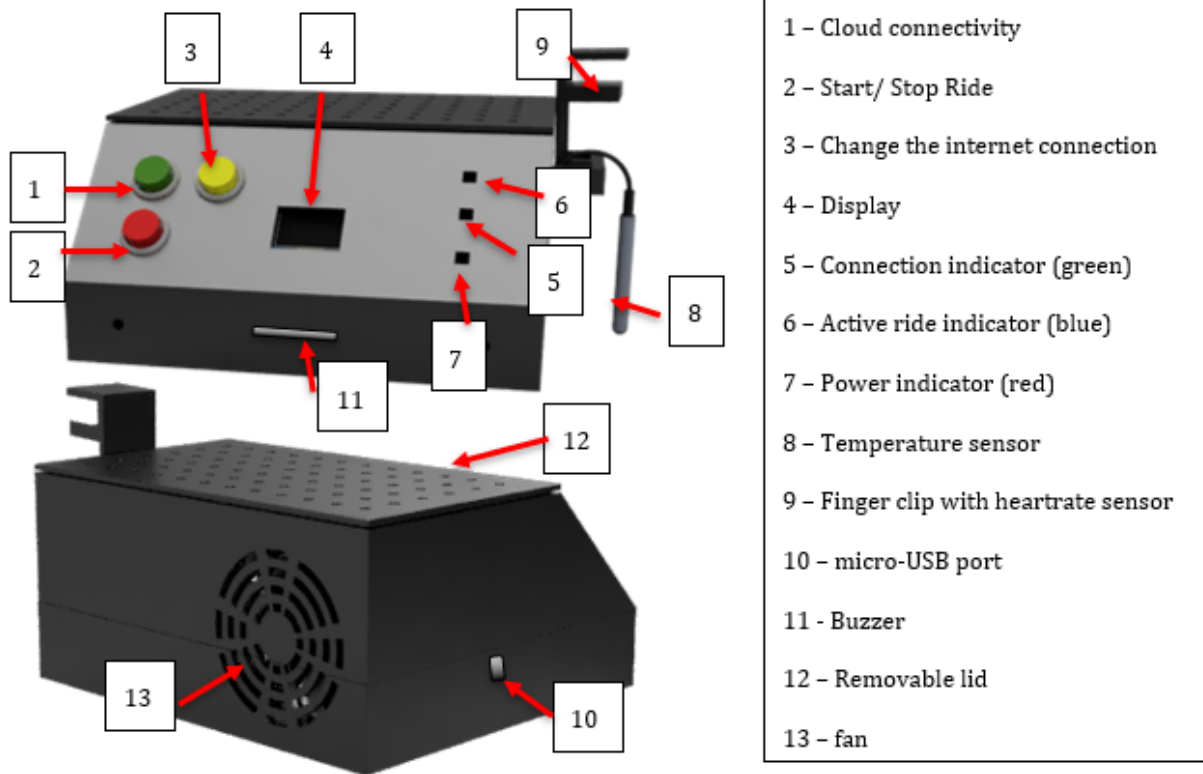


Figure 25 : Components of the device

7.3.COMPONENT PLACEMENT

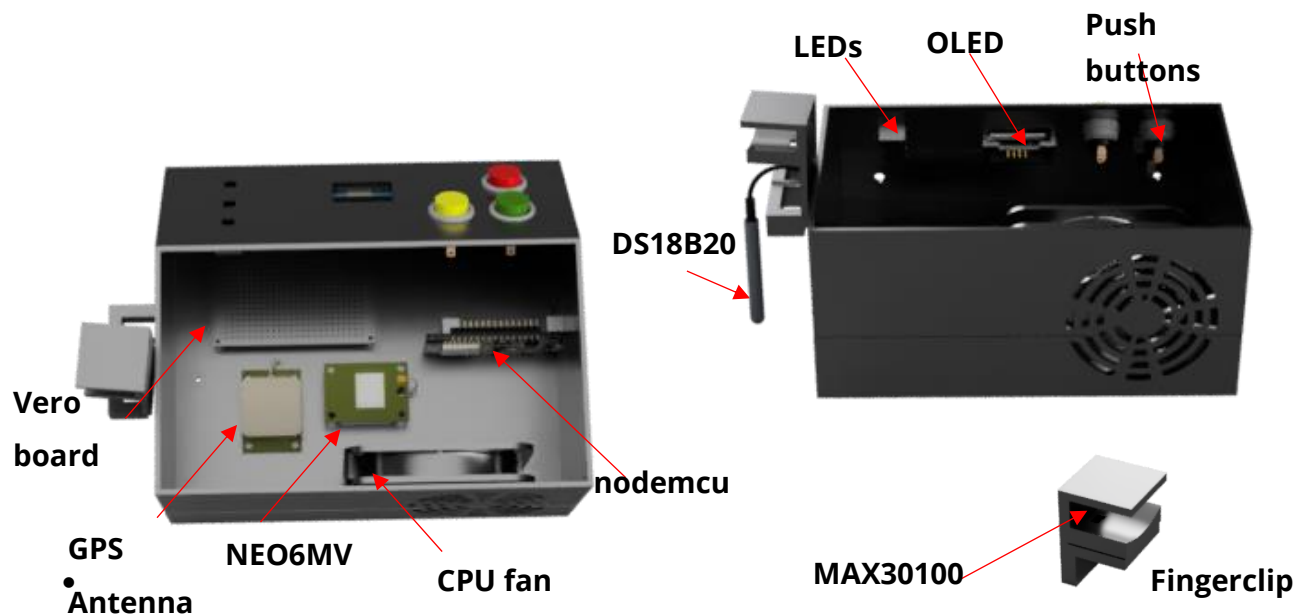


Figure 26 : Component placement inside the device

7.4.DESIGN DECISIONS

- Ventilation purposes in the device
It helps to dissipate heat generated by the components which can cause the device to malfunction or damage the components, to prevent the buildup of dust and other debris inside the device
Method: make the lid with holes
Insert a CPU fan inside the device
- Easy reachability to sensing components of the device
Method: heart-rate sensor was inserted into a finger clip and connected to an easily adjustable cable
- Proper visibility of the OLED screen
Method: The plane consists of the display made inclined.
- Thoroughly fixing the components to the base of the device
Method: Made holders, screw holes with a fixed base to hold them

8.1. FINAL PRODUCT

- Dimension: 170 mm * 150 mm * 80 mm
- Weight: 900 g
- Material (Box): Cladding



9. TESTING

9.1. SOFTWARE TESTING

For the software testing of desktop application and the mobile application, we've used flutter driver automation tests. To use flutter driver you need to add relevant dependencies to pubspec.yaml file

```
dev_dependencies:  
  flutter_driver:  
    sdk: flutter  
  test: any
```

```
dependencies:  
  flutter:  
    sdk: flutter
```

It can be used to test various aspects of our flutter apps, such as the functionality of individual widgets, the behaviour of the app's UI etc. As examples of tests that can be run with Flutter Driver include unit tests, widget tests, and integration tests.

We've used it to run login tests to the system, behaviour of widgets like buttons, text fields, app bars, chat fields etc.

```

test('Check for wrong username password inputs', () async {
  // ...
})

test('Check for correct username password inputs', () async {
  // ...
})

test('Check for initial states of second page', () async {
  // ...
})

test('Check for entering details to initialize the connection with the hospital', () async {
  // ...
})

test('checking chat', () async {
  // ...
})

```

Complete test script for the mobile application can be found [here](#).

Complete test script for the web application can be found [here](#).

9.2. API TESTING

Postman was used for API testing. With that tested each endpoint of the API to ensure that it returns the expected response.

The screenshot shows the Postman interface for an API test. The request is a GET method to the URL `https://rrne8k3me4.execute-api.ap-northeast-1.amazonaws.com/prod/passwordlist?UserID=D002&Password=password3`. The query parameters are set as follows:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> UserID	D002	
<input checked="" type="checkbox"/> Password	password3	
Key	Value	Description

The response status is 200 OK, with a time of 464 ms and a size of 297 B. The response body is displayed in the 'Body' tab, showing the JSON response:

```
1 "False"
```

Overview | PUT https://rrne8k3me4.execute-api.ap-northeast-1.amazonaws.com/prod/passwordlist?UserID=D002&Pa...

Passw... / https://rrne8k3me4.execute-api.ap-northeast-1.amazonaws.com/prod/passwordlist?UserID=D002&Pa...

PUT | https://rrne8k3me4.execute-api.ap-northeast-1.amazonaws.com/prod/passwordlist?UserID=D005&Password=password5 | Send

Params • Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	UserID	D005			
<input checked="" type="checkbox"/>	Password	password5			
	Key	Value	Description		

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 3.48 s Size: 296 B Save Response

Pretty Raw Preview Visualize JSON

1 "Done"

9.3 HARDWARE TESTING

For hardware testing platform.io has been used to check the functionalities of the sensors.

PIO Home X testino main.cpp U c_cpp_properties.json

MAX30100 by Oxidlo Intersectans

Maxim-IC MAX30100 heart-rate sensor driver and pulse-oximetry components

Installation

1.2.1 released 4 years ago Add to Project More info

Examples Installation Headers Changelog

MAX30100_Debug

MAX30100_Debug

Platforms

PROBLEMS 710 OUTPUT DEBUG CONSOLE TERMINAL

```
Heart rate:36.34bpm / SpO2:99%
Beat!
Heart rate:38.19bpm / SpO2:99%
Beat!
Heart rate:39.75bpm / SpO2:99%
Heart rate:39.75bpm / SpO2:99%
Beat!
Heart rate:41.61bpm / SpO2:99%
Beat!
Heart rate:43.12bpm / SpO2:99%
Beat!
Heart rate:52.01bpm / SpO2:99%
Beat!
Heart rate:57.97bpm / SpO2:99%
Beat!
Heart rate:57.64bpm / SpO2:99%
Beat!
Heart rate:61.01bpm / SpO2:100%
Beat!
Heart rate:58.42bpm / SpO2:100%
Heart rate:58.42bpm / SpO2:100%
Beat!
Heart rate:42.13bpm / SpO2:100%
```

Figure 27 : Testing for MAX30100

test.ino Mmain.cpp U ×platformio.ini UPIO Home

AmbulanceProjectTesting > src > main.cpp > loop(void)

10 // Pass oneWire reference to DallasTemperature library
11 DallasTemperature sensors(&oneWire);

PROBLEMS 705OUTPUTDEBUG CONSOLETERMINAL

Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.00°C	78.80°F
Temperature: 26.31°C	79.36°F
Temperature: 26.31°C	79.36°F
Temperature: 26.88°C	80.38°F
Temperature: 26.88°C	80.38°F
Temperature: 27.44°C	81.39°F

Figure 28 : Testing for DS18B20