Team 28
Developer Document

Wearable Device

Trinadh Venkata Satyanarayana Dusanapudi,
N.V. Sree Harsha Reddy,
Sri Ram Arumulla,
Varul Srivastava,
Ashwin Mittal.

Contents

1 Introduction

1.1 Problem Statement and Scope

Our problem statement was to develop a wearable device that can measure temperature, blood pressure, skin-conductance response, SPO_2, and pulse rate. Users can access their data directly from the device interface or a web application. User data should be kept private and dealt with security. Along with the collected data, the web application should also alert the user if there is an anomaly seen in the data like COVID and if a person has a panic attack, etc.

Existing devices don't have continuous monitoring of blood pressure but we have that using optical non-cuffed sensors which are easy to wear and do continuous monitoring of the user's blood pressure and update the data on the server. The future scope of this project can range from making this wearable device deployable. Other than this we can also feature things like calling, updating the location, etc to make it versatile. Also, the bulkiness of the system makes it physically difficult to carry. We can reduce the size of it by using microprocessors, etc.

1.2 Purpose of the System

Due to the outbreak of pandemics and increasing stress levels of people, constant health monitoring has become a necessity. The early recognition of symptoms of increasing stress levels or covid is proven to be very effective to stop the deterioration of the health of the person.
Early detection: Studies (Leenen et al.)[6] show that wearable devices allow early detection of clinical deterioration even before symptoms / need for medication arises.
Lack of vital measurements: Current wearable devices have basic measurements like pulse and oxygen saturation levels, but lack vital measurements like Blood-Pressure.
Integration: There also is a lack of an integrated wearable device making use of multiple measurements and their correlation to make clinical analysis, predictions, and suggestions.
Our main purpose is to provide a solution to cover the above three criteria.

1.3 Overview of the system

The ESP32 maintains all the connections and obtains the measurements from all the sensors. This data will be pushed to the oneM2M cloud. The data would be encoded and hashed. A backend server is developed and hosted using the flask. The backend server obtains the data from the cloud. It decodes the data, analyses it,

and stores the data. From time to time, the backend would send the data to the frontend(web-app) to the display. The telegram bot would also be used to display the measurements and various analytics.

Apart from this given the situation, we are living in, it is very important to know if a person is COVID positive. So based on the parameters that we have access to from the readings of the sensor we have used an ML model to predict if a person is COVID positive. Apart from this, we have also provided a feature to predict if the person is suffering from panic attacks. Notice that all these are predicted using the sensor readings we took and some general information about the person which we take as input in the frontend. Apart from these, we are storing the sensor readings in a file for a further and deeper understanding of the data.

2 Design Document

2.1 System Requirements

We aim to build an IoT based health monitoring system, the system should have the capacity of

- Temperature, Blood Pressure, Skin-Conductance Response, SPO_2, and Pulse Rate measurement.
- Ability to upload data on cloud/ OneM2M, from where it can be accessed for analytics and other inferential purposes.
- Provide security of the data we upload, from unauthorized access, and maintain the integrity of data using the checksum feature.
- Provide a web-app-based option to access data from the IoT device and show it on a dashboard. Provide remote access to measurement of the system through a chatting platform
- Provide alerts based on prediction on the covid status of the user.

2.2 System Specifications

The following are the specifications for each part:

- **ESP32**: Microcontroller responsible for interfacing each sensor. It also has a built-in WiFi Module capable of detecting at a range of 5 meters.
- **ProtoCentral MAX30205**: This sensor is responsible for reading body temperature. For reading the body temperature the body should be in contact with the sensor. It measures via I2C protocol.
- **Pulse Express Pulse-Ox & Heart Rate Sensor with MAX32664:** This sensor is responsible for getting the readings of Bp, HeartRate, and SPO_2. For reading the data we should keep one finger on the optical sensor region. It measures via I_2C protocol.

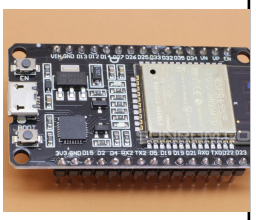
- **SeeedStudio Grove GSR sensor**: This sensor reads the human resistance, which gives an idea about the person's emotion. We have to touch the two electrodes with two fingers of the same hand. It measures it via conductivity between the electrodes.
- **Waveshare 2 Inch LCD Display Module 240×320**: This is to display the readings of the sensor's data.

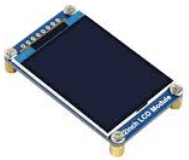
2.3 Stakeholders

The product enables us to monitor the (parameters of our) health and since it also provides remote access to the measurements of the system, the major stakeholders are the people who wish to have a constant check on their or others' health. Nowadays, many people are afraid to go to crowded places or interact in groups because of the COVID-19 threat. As we also provide alerts about our prediction on the covid status to the user of the wearable watch, the people whose work involves interacting with people or people who travel through crowded areas will be the stakeholders of our product.
Nowadays, anxiety and stress have become major problems in almost all periods of life. Skin conductance response can be correlated to the stress or anxiety levels of a person. The Measurement of Skin Conductance Responses can be used to understand the emotional experience. The people who want to monitor their or others' anxiety and stress levels could act as stakeholders of our product.

2.4 Design Entities and Main Components

Summary of entities used in our system:

Entity type	Entity symbol	About Entity	Image
ESP32 Wroom Board	ESP32	maintains all the connections and obtains the measurements from all the sensors	
MAX30205	MAX30205	measures temperature and provides an over-temperature output. Communication is through an I2C-compatible 2-wire serial interface.	

MAX32664	MAX32664	Integrated biometric sensor hub. Easy-to-use I2C interface to connect to any host microcontroller.	
Grove GSR sensor	GSR	measures the electrical conductance of the skin. Communicates through GPIO	
LCD Display	LCD	A general LCD display Module with an IPS screen, 2 inches diagonal, 240×320 resolution, embedded controller, communicating via SPI interface.	
Power bank (Testing Power supply)	Power Bank	For Testing purposes, we use this as a main power source for the system.	
Cell (Deployment Power supply)	Cell	For Deployment purposes, we use this as the Main power source for the system.	
oneM2M server	OneM2M server	Web app requests and downloads data from oneM2M server.	

ThingSpeak Cloud	ThingSpeak Server	Push the data for visualization purposes.	

2.5 Design Details

2.5.1 Conceptual Flow

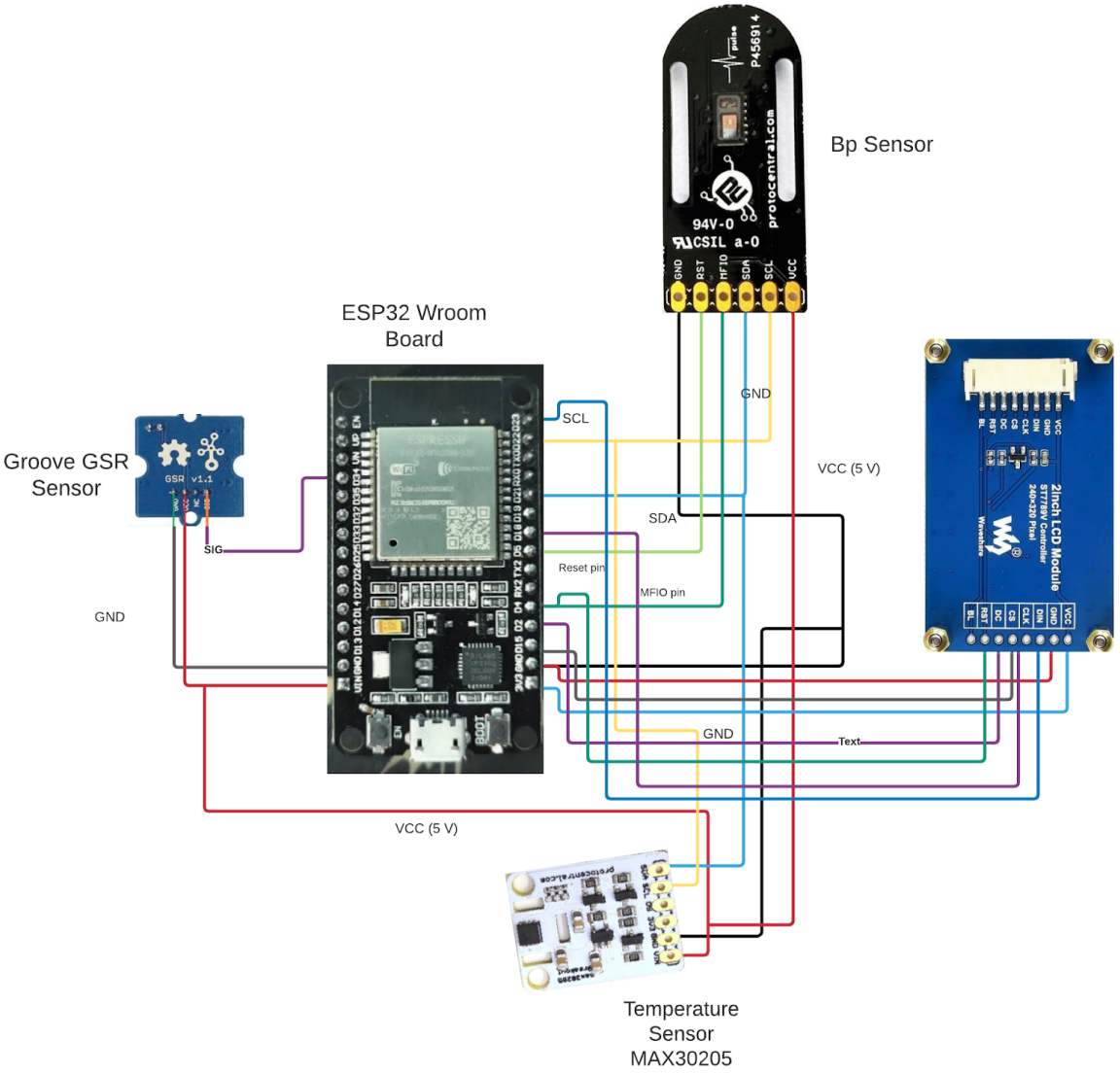
Our workflow could be divided into seven steps:

1. **Choosing and testing the sensors:** Various sensors were chosen and tested individually for health monitoring purposes. When choosing a sensor like a cost, precision, accuracy, etc.
2. **Integration and circuit design:** Once all the sensors are tested, a combined circuit containing all the sensors is assembled and tested. Along with this, a circuit diagram was also made for easy reference.
3. **Connectivity and Data Processing:** After all the sensors are well tested in a common circuitry, the next step is to post the data onto the desired platform and make the device an IoT device. This was done by creating a thingspeak channel for the sensor's reading posting and making the required changes in the code to push the data onto the thingspeak channel.
4. **Preliminary Deployment:** We did this to test our system and to give demonstrations during the mid evaluations and our system worked fine with good predictions.
5. **Final Deployment (ML model):** After doing all the testing, we added a machine learning model in the backend to make better predictions than before because the model has 94% accuracy. Then we make our deployments with all the hardware and software components working fine.

2.5.1 Entity Interaction

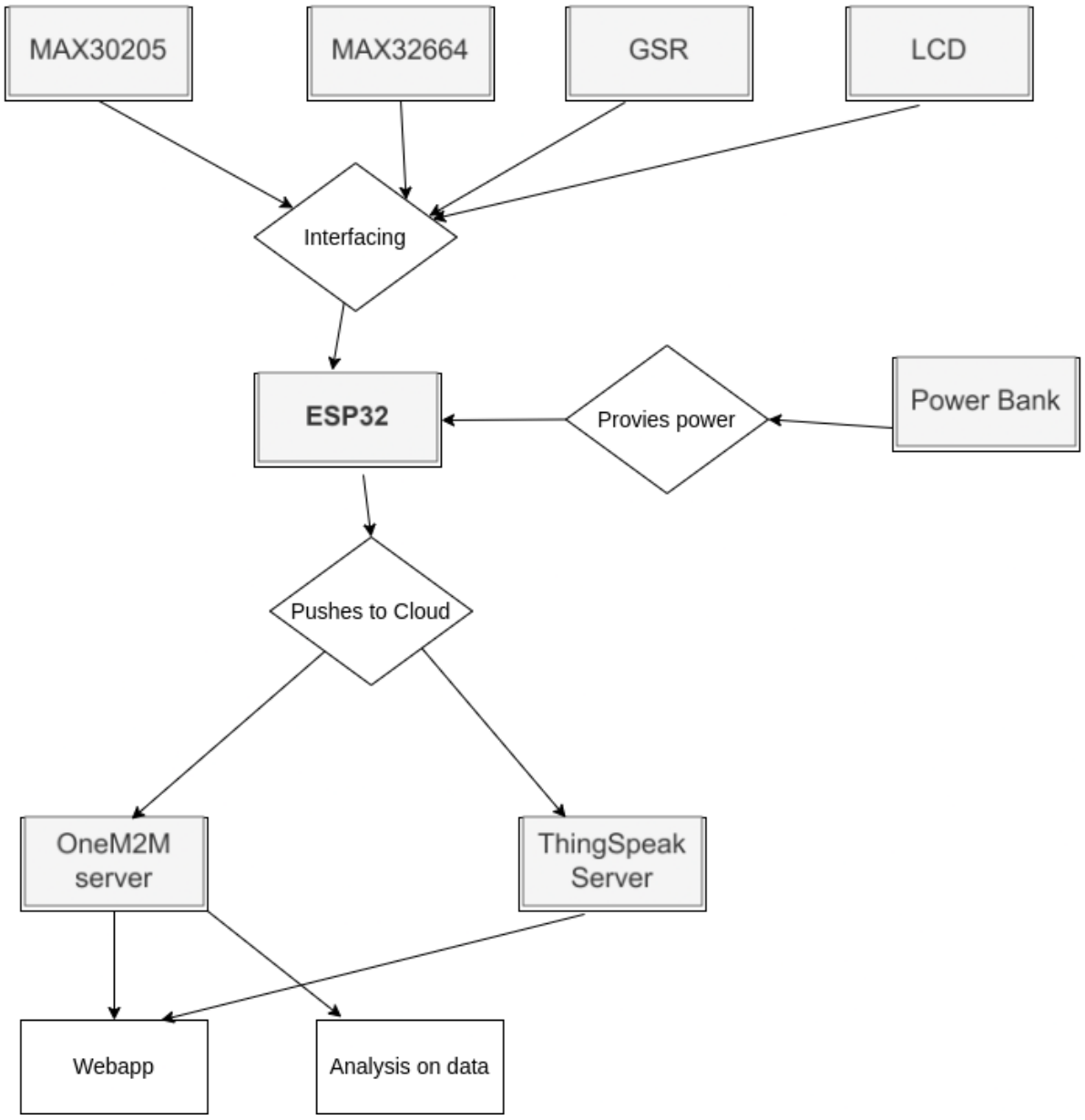
The interaction types ares listed as follows:

- **Interfacing of node with sensors:** The ESP32 is connected to all sensors, and the circuit diagram is given below.



ESP32 connects to MAX30205 and Bp Sensor via I2C pins(D21,D23) , LCD display via SPI pins(D18, D22) and GSR sensor via GPIO pins(D34).

- **Provides power** : power bank provides power for the ESP32 board.
- **Pushes data to cloud :** ESP 32 used WiFi to push the data to cloud servers i.e oneM2M for displaying in web app and for analytics and also to ThingSpeak to visualize data in webapp.



2.6 Operational Requirements

2.6.1 System Needs

The system requires for it to function properly, they are summarized below:

WiFi Connection: The system needs a stable WiFi connection to ensure that proper flow of data is maintained for various reasons like storage in oneM2M, handling telegram requests etc. We need a stable connection above 2.4GHz to be able to bypass any firewall.

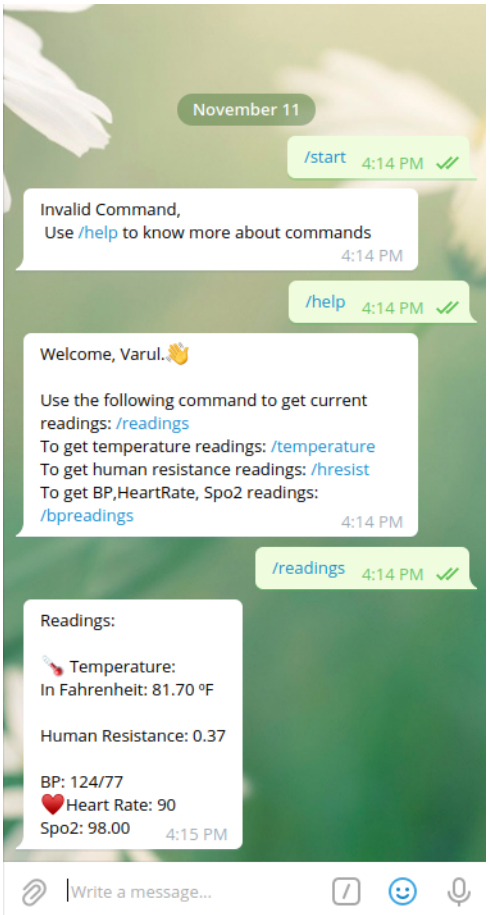
Power Supply: The system needs a battery to supply power for its functionality. We can always use the laptop for temporary usage and testing purposes. We are using

Battery Shield for the ESP32 power supply. We use 18650 li-ion batteries in the battery shield to give the power supply to the ESP32. We also used a power bank instead of a battery shield for the power supply.

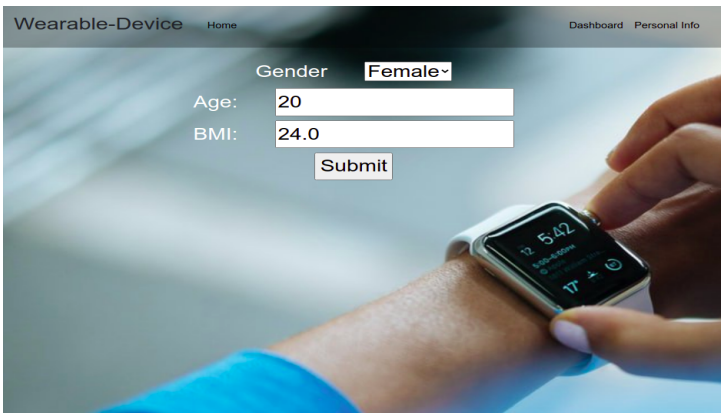
2.6.2 UI Design

2.6.2.1 Telegram

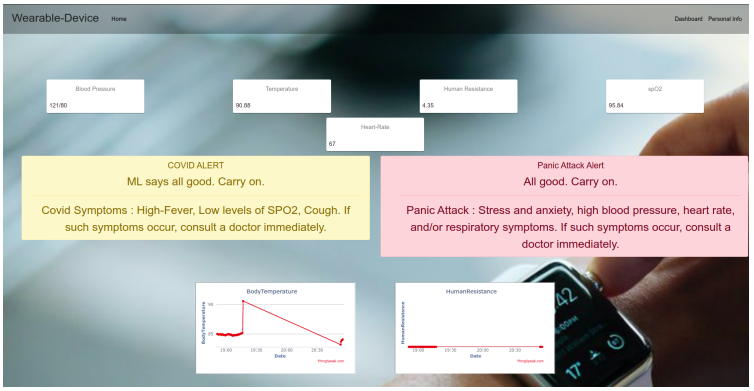
To get live updates about the condition of the person(to whom the device is attached) we have developed a Telegram bot. Remote access of the conditions and the readings of the Health Monitoring IoT device is made possible with the help of the Telegram bot. We can access the current/live temperature, heart rate, and blood pressure remotely from the telegram bot by issuing single commands.



2.6.2.2 Front End (Dashboard)



Though web application dashboard users can see their data securely. This screen is used for updating users' personal information. These details are required by the ML for prediction. This webpage does client-server communication once.



This page shows the user data with the real-time updating of information with ML-prediction of COVID, monitoring for panic-attack. The user has to enter the key in the text box provided below to decrypt data and show it.

2.6.2.3 LCD display

As our device is wearable, we want it to be like a watch (fit band). Even though it sends the data to apps it displays the readings on their screen. Like that, we also used a 2-inch Colour LCD module to show the readings of the health metrics, and also indicate the color of the reading as the reading is dangerous or not.

2.6.3 Analytical System

We are using ML Model for covid-prediction [1][2] in the backend to predict the covid status of the user of the wearable watch. It takes input parameters like age, gender, systolic-bp, diastolic-bp, etc. While predicting if the patient is suffering from covid19. The model has **94% Accuracy.**

1. Other than this we are also analyzing the sensor readings to inform the user if he is suffering from panic attacks. For data aggregation in the backend, we are using a JSON file to store the data. We are updating this data in the JSON file every time there is a request from the frontend to get the data from the OM2M. The data stored in the JSON file is decrypted first, then stored. We are also checking if the values entered by a user for various fields are correct or not and assign them with default values if nothing is entered.

3 Technical Document

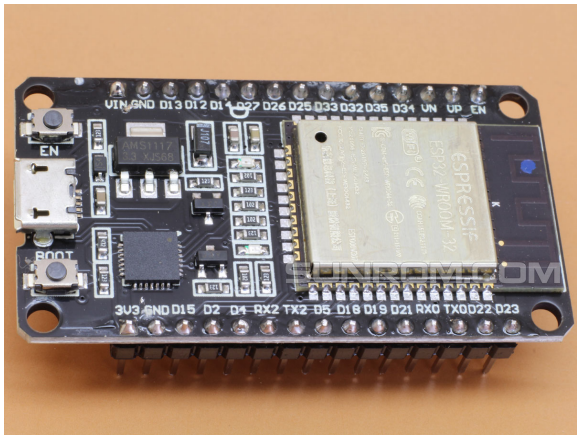
3.1 Design Docs

Refer to the detailed Design Document that is part of this project

3.2 Hardware Specifications

3.2.1 ESP32 Wroom Board

ESP32 Development board is based on the ESP WROOM32 WIFI + BLE Module. This is the latest generation of the ESP32 IoT development modules. This development board breaks out all ESP32 modules pins into a 0.1" header and also provides a 3.3 Volt power regulator, Reset and programming button, and an onboard CP2102 USB to TTL converter for programming directly via USB port.
For more detailed specifications, refer to the manufacturer Datasheet



The following pins of the esp32 board are used for this project.

- **Pin VCC**: Used to connect VCC of the sensors
- **Pin 3.3V**: Used to connect VCC of the low power sensors
- **Pin GND**: Used to connect GND of the sensors
- **Pin D21**: Used for data transfer of I2C (SDA pin)
- **Pin D23**: Used for the clock of I2C (SCL pin)
- **Pin D18**: Used for the clock of SPI.
- **Pin D22**: Used for data transfer of SPI.
- **Pins D4,D34,D15,D5,D2** : Used for GPIO of sensors

3.2.2 ProtoCentral MAX30205 Body Temperature Sensor

The MAX30205 temperature sensor accurately measures temperature and provides an over-temperature alarm/interrupt/shutdown output. This device converts the temperature measurements to digital form using a high-resolution, sigma-delta, analog-to-digital converter (ADC). Communication is through an I2C-compatible 2-wire serial interface.

Specifications of the sensor:

- 0.1°C Accuracy (37°C to 39°C)
- 16-Bit (0.00390625°C) Temperature Resolution
- 2.7V to 3.3V Supply Voltage Range
- 600µA (typical) Operating Supply Current
- 16-bit Resolution
- I2C interface
- Operating temperature: 0 to 50 °C
- “OS” open-drain pin can act as thermostat control
- I2C address 0x48

Symbol	Description
SDA	Serial Data
SCL	Serial Clock
Vin	Power(5V)
GND	Ground
3v3	Power(3.3V)

3.2.3 Pulse Express Pulse-Ox & Heart Rate Sensor with MAX32664

Pulse Express is an efficient and versatile breakout board with integrated high-sensitivity optical sensors MAX30102 and also a chip that does the calculations (biometric sensor hub MAX32664D. Integrating Maxim's MAX32664 Version D makes Pulse Express unique, with an internal algorithm that works to measure different data as you start. With its built-in low power capability, the board is suitable for any wearable health for finger-based applications. It measures Pulse Heart Rate, Pulse Blood Oxygen Saturation (SpO2), Estimated Blood Pressure.

Specifications of the sensor:

- Integrates a high-sensitivity pulse oximeter and heart rate sensor (MAX30102).
- Integrated biometric sensor hub (MAX32664)
- In-built accelerometer for robust detection and compensation of motion artifacts (Optional).
- Easy-to-use I2C interface to connect to any host microcontroller.
- Ultra-Low Power
- Dimensions: 35 mm x 17 mm
- I2C address 0x55

Symbol	Description
SDA	Serial Data
SCL	Serial Clock
Vin	Power(5V)
GND	Ground
MFIO Pin	MFIO
Reset Pin	Reset

3.2.4 SeeedStudio Grove GSR sensor

Grove – GSR Sensor stands for galvanic skin response and it is a method of measuring the electrical conductance of the skin. It can be used to reflect human emotional activity. When we are emotionally stressed or have strong expressions on the face, sympathetic activity increases and promotes the secretion of sweat glands, which increases the skin's electrical conductivity.

This sensor allows us to spot such strong emotions by simply attaching two electrodes to two fingers on one hand.

Specifications of the sensor:

- Input Voltage: 5V/3.3V
- Sensitivity adjustable via a potentiometer
- External measuring finger cots

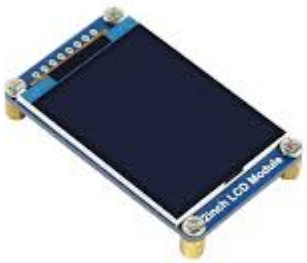
Symbol	Description
VCC	Power(5V)
GND	Ground
SIG	General Input-output pin

3.2.5 Waveshare 2 Inch LCD Display Module 240×320

This is a Waveshare 2 Inch LCD Display Module. A general LCD display Module with an IPS screen, 2 inch diagonal, 240×320 resolution, embedded controller, communicating via SPI interface. Supports Raspberry Pi, STM32, Arduino, etc.

Specifications of the display:

- Driver: ST7789
- Interface: SPI
- Display colour: RGB, 262K colour
- Resolution: 240×320
- Backlight: LED
- Operating voltage: 3.3V
- SPI interface requires minimum GPIO for controlling.



Symbol	Description
VCC	Power (3.3V input)
GND	Ground
DIN	SPI data input
CLK	SPI clock input
CS	Chip selection, low active
DC	Data/Command selection
RST	Reset, low active
BL	Backlight

3.2.6 Robustness

We are using Battery Shield for the ESP32 power supply.
We use 18650 li-ion batteries in the battery shield to give the power supply to the ESP32.
We also used a power bank instead of a battery shield for the power supply.

3.3 Communication

3.3.1 GPIO

The GPIO pins on the board are used for communication with GSR-Groove Sensor. And also with other sensors for reset type of pins which resets the sensor's state etc. We are an ADC type of GPIO pin, in which we can control the input resolution. As for the groove sensor we used 12-bit resolution, so we will get values from 0- 4095.

3.3.2 I2C

I2C (Inter-Integrated Circuit) is used for interfacing the board and the Pulse Express Pulse-Ox & Heart Rate Sensor and ProtoCentral MAX30205 Body Temperature Sensors. And the pins for I2C Communication:

Pin label	Description
SDA	I2C data
SCL	I2C clock
VCC	Power(3.3 or 5V)
GND	Ground

The addresses used for I2C communication were 0x48 and 0x55 for temperature and Bp sensor.

3.3.3 SPI

Serial peripheral interface (SPI) is used for interfacing the board and the 2-inch LCD display. Sensors have the following pins for SPi communication:

Pin Label	Description
VCC	Power (3.3V input)
GND	Ground
DIN	SPI data input
CLK	SPI clock input
CS	Chip selection, low active
DC	Data/Command selection

3.3.4 WiFi

We use the WiFi feature of the board to connect to the Internet to push data to oneM2M server and ThingSpeak channels

The WiFi module has following features:

- 802.11 b/g/n (802.11n up to 150Mbps) Wi-Fi protocol
- Bluetooth v4.2 BR/EDR and BLE
- SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, and IR module interfaces
- Hall sensor
- 2.7V to 3.6V operating voltage supply
- 80mA average operating current
- -40°C to 85°C operating temperature range

3.3.5 OneM2M

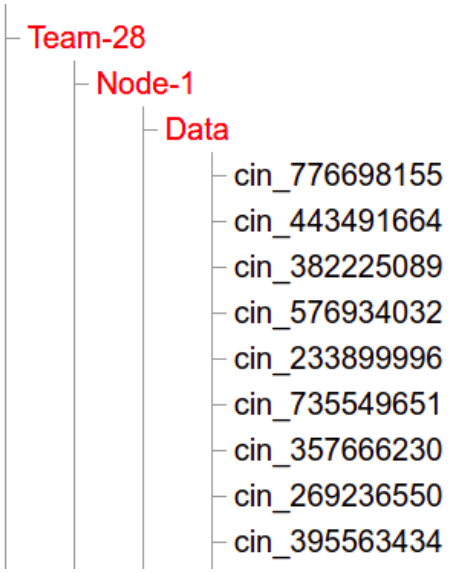
3.3.5.1 Overview:

A global initiative to develop IoT standards to enable interoperable, secure, and simple to deploy services for the IoT ecosystem. Helps us to increase reusability. We have used the following parameter pushing data to the server.

```
// oneM2M : CSE params
const char* CSE_IP      = "esw-onem2m.iiit.ac.in";
//String CSE_IP      = "192.168.1.egin(9600);233";
int   CSE_HTTP_PORT = 443;
String CSE_NAME     = "~/in-cse/in-name";
String CSE_M2M_ORIGIN  = "3ULsBQex0w:0z8Uh6re@6";
```

OM2M is an open source implementation of OneM2M standard by LAAS-CNRS. We have used stored information at esw-onem2m.iiit.ac.in. All the sensor readings which are taken at IoT nodes will be encrypted and stored here. Whenever we want to access the information we will do so by making GET requests to the OneM2M platform.

The hierarchy of storage is /Team28/Node-1/Data inside this all the instances of the readings are stored. When we are storing/accessing the data instances we will do from the above-mentioned path. Each of the data instances will have a unique id. In each instance, we will have data in the encrypted form sent by IoT nodes. For verification of the APIs of POST and GET requests, we have used the Postman app extensively for testing.



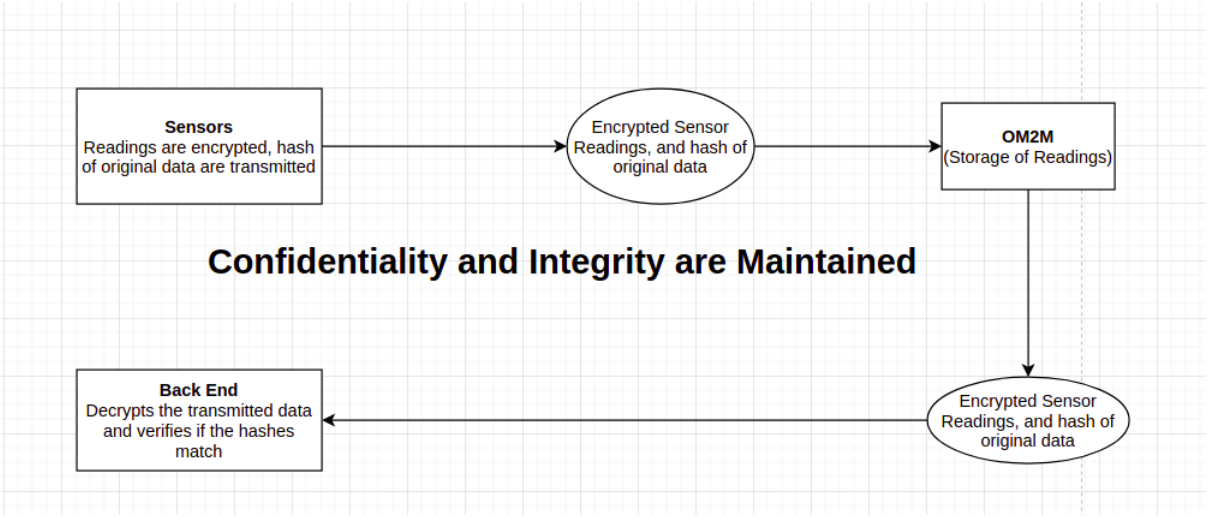
3.3.5.2 OneM2M on IoT nodes side

All the sensor readings (temperature, GSR, bp sensor readings) which are taken at IoT nodes end will be in raw form. Before transmitting data we process the data so the security is maintained. We encrypt the data and hash it such that the original

data is not visible in its naive form in OneM2M. We push this data to the OneM2M server using a POST request.

3.3.5.3 OneM2M on Backend

In the backend, we access the information of the sensor readings (which are encrypted) using a GET request. The flow of the data from IoT nodes to OneM2M to the backend is such that the information is always encrypted. We then verify this information which is decrypted with the hash of it, to check if the data is corrupted while transmitting.



3.3.6 ThingSpeak

3.3.6.1 Updation of data

We use the thingspeak library and its APIs to push the data from IoT nodes. We use the write key to push the sensor readings from the IoT nodes directly. Each of the sensor readings is stored in a separate graph to have an idea of how the readings are varying.

3.3.6.2 Retrieving the graphs

To plot the graphs of the sensor readings we use the links which are provided by thingspeak to access the graphs directly. These graphs are displayed in the dashboard for better visualization to the user.

3.4 Software Specifications

3.4.1 Libraries Used

1. **WiFi.h**: This library is used for establishing WiFiClient and connecting to WiFi
2. **WiFiClientSecure.h**: This library is used for establishing WiFiSecure Client connections with HTTPS requests as they are secure.
3. **Protocentral_MAX30205.h**: This library is used for collecting data from the MAX30205 Temperature sensor.
4. **WebServer.h**: This library is used to create the localhost webpage using the board.
5. **mbedtls/aes.h and mbedtls/md.h**: Encrypt and hash functions of these libraries are used to encrypt the data and also hash the data.
6. **Adafruit_GFX.h and Adafruit_ST7789.h**: These libraries are used for the ST7789 driver of the LCD display
7. **TFT_eSPI.h and SPI.h**: These libraries are used to display text and explore other features (color, etc) of LCD display
8. **UniversalTelegramBot.h**: This library is used for communication between the board and telegram bot which gives updates on data.
9. **ArduinoJson.h**: This library is used to convert string to JSON string which is uploaded to oneM2M server
10. **ThingSpeak.h**: This library is used to send the data to ThingSpeak channels using the write API key of the channel.
11. **Max32664.h**: This library is used for collecting data from the protocentral BP Sensor.

3.4.2 Security

3.4.2.1 Our goals for security

We aim to provide security to our system, and data integrity. Our goal for security is to provide the following security features:

a. **Data-integrity**: There should be a near-guarantee probability of detecting any possible data manipulation that might have happened so that we only process and/or use untampered data.

b. **Data privacy**: We also try to ensure the privacy of users' data. Since we are using third-party (untrusted) cloud services, we use symmetric encryption to ensure that our data is not compromised/leaked.

3.4.2.2 How we implement encryption - Privacy

For the privacy aspect, we use Symmetric-key encryption. One of the problems that we faced was the incompatibility between the encryption techniques (implemented using libraries). Thus as a solution, we used our variant of secure symmetric-key encryption which is a variant of the one-time pad, which avoids repetition of key, by using SHA-256 hash function for a sub-key generation.

The data is encrypted before uploading on the OneM2M cloud, and after being downloaded on the backend, it is decrypted when the user provides the key using the frontend dashboard.

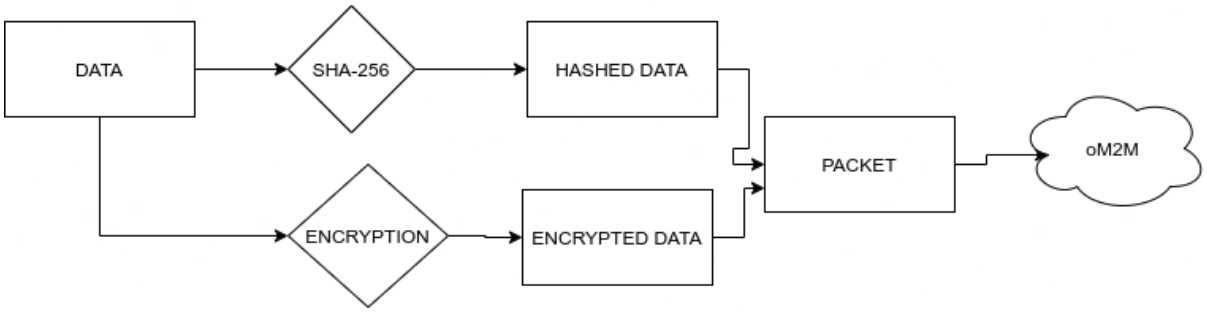


Fig. Implementation of security on the device end

3.4.2.3 How we implement hashing - Integrity

We use hashing for maintaining data integrity. There is a possibility of data being compromised due to

a. Bit-flipping: This happens during communication, due to interference, noise, etc. transmitted messages have some of their bits changed. This is an external influence not adversarial
b. Malicious Cloud: There is a possibility of the cloud being malicious and trying to tamper the data. However, it cannot tamper with the encrypted data such that the hash of the decrypted message is the same as the hash stored in the cloud.

Notice that due to (b) we are **storing a hash of original data** (not encrypted data). It is because in doing so, for a successful attack, the malicious adversary will have to either break encryption or solve the pre-image problem of SHA, both of which are infeasible.

For verification, we download both cloud data and hash from the om2m server. Then, we decrypt the data and check it against the hash. If the hash is the same, then data is not compromised.

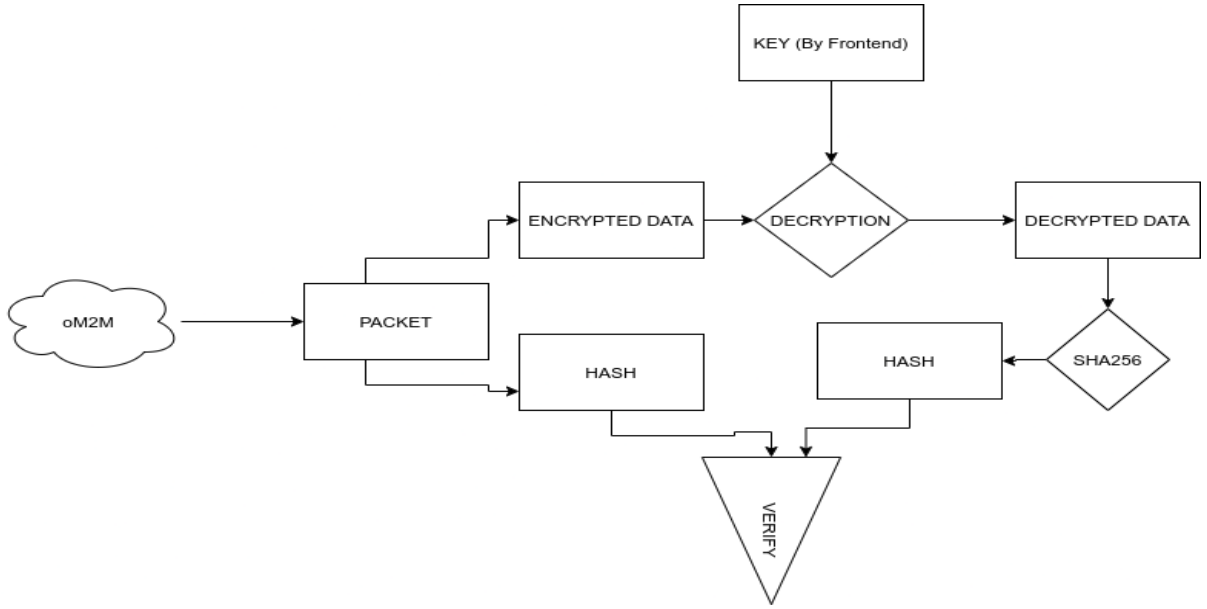


Fig. Security on the server-end

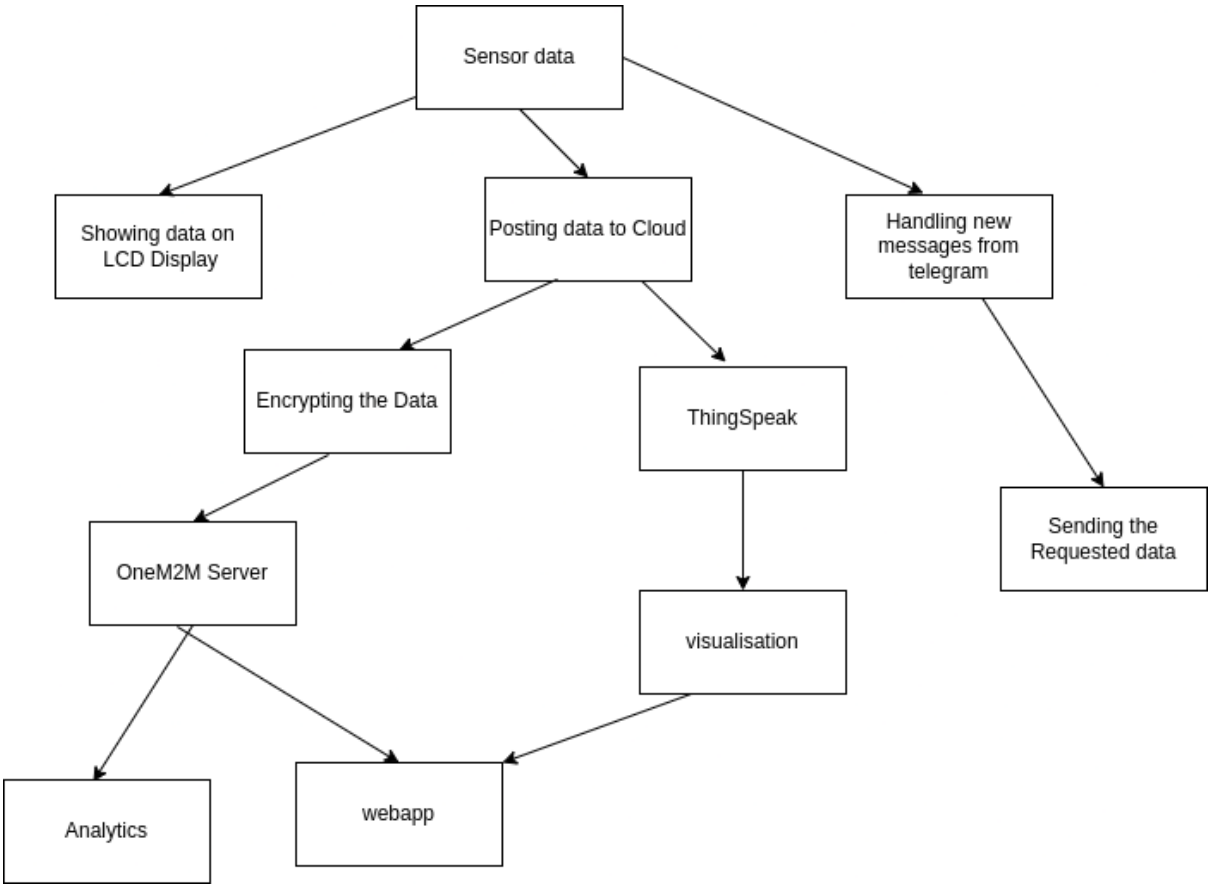
3.4.2.4 Other Security analysis

In this section we do the security analysis, i.e. we decide how secure the system is against common attacks.
A. Packet Sniffing: Since all communications happen in an encrypted format, we can guarantee packet sniffing is infeasible as no information is leaked.
B. Malicious Cloud: Our system is secure against malicious data tampering going undetected from the cloud end. Data-tampering is not avoidable (by any scheme, if they use a single trustless server as in our system architecture), however, we can detect data-tampering by the cloud by using pre-encryption hashing.

Our system is only insecure against physical-capture attacks because of which the key can be compromised which is stored on the device. However, we provide backward security in this case too, as it is impossible to obtain previous information without either compromising both the device and one of the OneM2M or backend servers.

3.5 Data Handling Model

The data is first measured from the sensors. After that data will be displayed in LCD Display, and Data is pushed to the cloud. Before pushing data into oneM2M we will encrypt the data and for ThingSpeak we will directly push the data. And also checks whether there are any new messages from the telegram, if they are then the data is sent to the telegram. By using oneM2M APIs, the data in the oneM2M cloud is decrypted when the user wants to display values on the web app and also do an analysis on the data. And using ThingSpeak APIs we are visualizing the data from graphs.



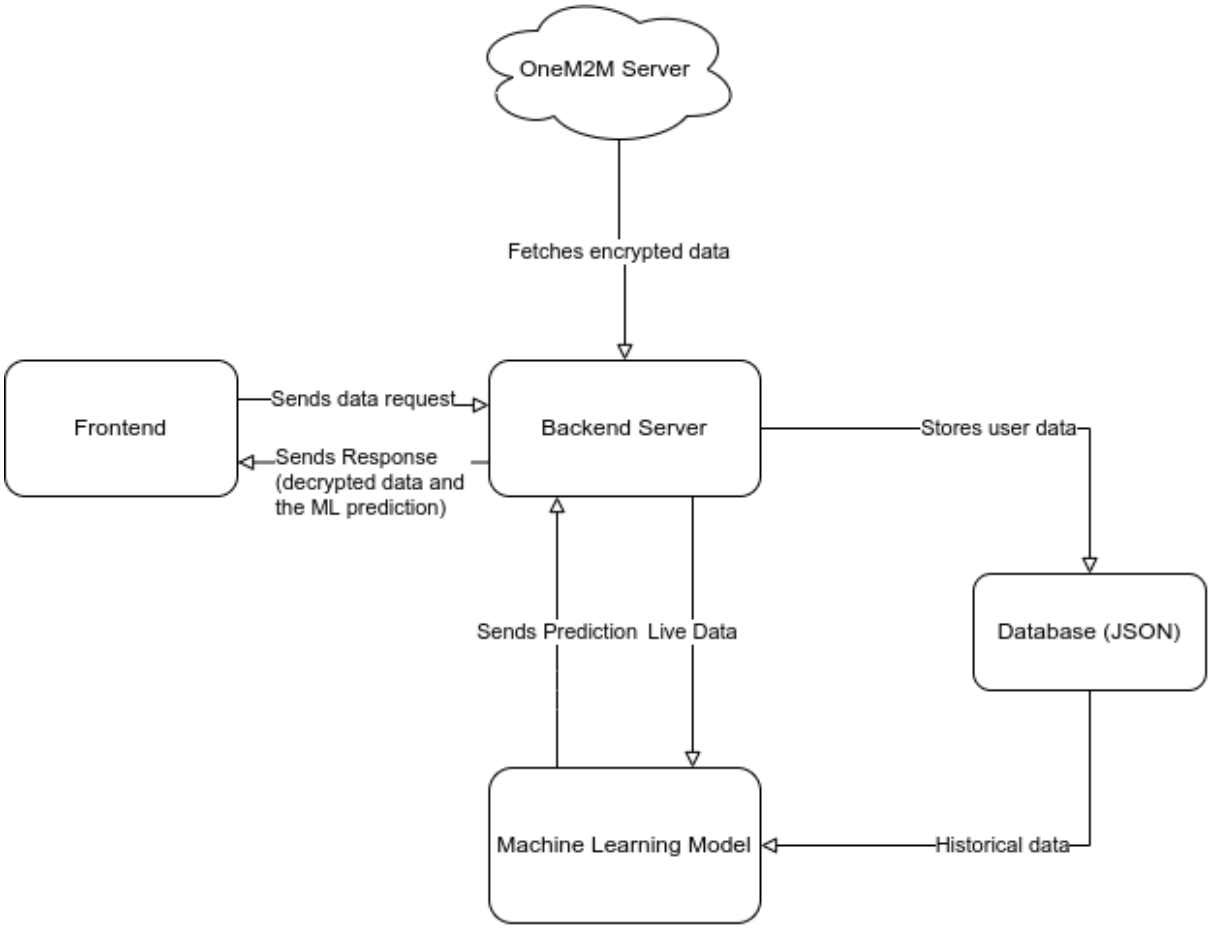
3.6 Flask Backend

Flask is a microframework for backend development in the Python language. What micro framework means specifically is that it comes with very little in terms of available features and boilerplate code at the start of a project's development. Now that's not to say there aren't libraries upon libraries of features and plugins available for the Flask framework, but by default, Flask will tend to not include a feature unless a developer has specified otherwise.

In our project, we are using the flask backend. In the Python backend, we are easily accessing web data from OM2M using inbuilt libraries and serving the data requests from the frontend.

3.7 Integration framework

To get the latest data from the OM2M, we are doing a get request to the IIIT OM2M server and getting the data. This data is then decrypted and updated in the user_data file in the database directory in the JSON format for machine learning predictions. Then we are sending this data to the frontend with the machine learning prediction to display on the dashboard.



3.8 Data Visualization and Analysis framework

3.8.1 Thingspeak:

Visualizations of data logged from sensors can be seen in ThingSpeak. In a ThingSpeak channel, you can analyze logged data and visualize live data in the form of graphs.

4 References

[1] https://www.nature.com/articles/s41746-021-00467-8

[2] https://github.com/ynaveena/COVID-19-vs-Influenza

[3] https://www.xtronical.com/colour-ecg-monitor/

[4] https://educ8s.tv/arduino-st7789-tutorial/

[5] https://create.arduino.cc/projecthub/iasonas-christoulakis/measure-spo2-heart-rate-and-bpt-using-arduino-68724d?ref=part&ref_id=10308&offset=9

[6] Al-Qatatsheh A, Morsi Y, Zavabeti A, Zolfagharian A, Salim N, Z. Kouzani A, Mosadegh B, Gharaie S. Blood Pressure Sensors: Materials, Fabrication Methods, Performance Evaluations, and Future Perspectives. Sensors. 2020; 20(16):4484. https://doi.org/10.3390/s20164484