SAMSUNG

IoT Course

Capstone Project Final Report

HOME-BASED PERSONAL HEALTH MONITORING SYSTEM INTEGRATED WITH HEALTH ADVISORY CHATBOT

05/08/2025

The Red Warriors

Phan Duy Hoàng	SIC251464
Lý Tấn Lộc	SIC251437
Ngô Thành Đạt	SIC251439
Nguyễn Việt Hoàng	SIC251458
Lưu Nguyễn Thảo Nguyên	SIC251435

CONTENT

1. Introduction

1.1. Background Information

This project aims to develop a health monitoring system that tracks essential physiological parameters such as heart rate, blood oxygen saturation (SpO2), and body temperature, utilizing the ESP32 microcontroller in combination with technologies such Wi-Fi, Bluetooth.

A key feature of the system is the integration of an intelligent chatbot within the mobile application, enabling patients to ask simple health-related questions based on the collected data. For instance, if a patient asks, “Is a temperature of 38.5°C dangerous?”, the chatbot can respond with basic medical recommendations.

The system is designed to operate flexibly in two modes:

- Online mode: When Wi-Fi is available, the ESP32 sends data directly to a Raspberry Pi server, and the chatbot can utilize the latest data for real-time consultation.
- Offline mode: In the absence of Wi-Fi, data can be accessed via Bluetooth from the mobile app. The chatbot can still provide guidance based on the locally stored data.

By combining automated sensing, intelligent storage management, dual-mode operation, and chatbot interaction, this system aims to provide convenient and proactive health monitoring for both individual users and healthcare professionals.

1.2. Motivation and Objective

The main objective of this project is to develop a remote health monitoring system that can operate reliably even without continuous internet connectivity. The system should be capable of:

- Automatically measuring vital signs at predefined intervals.
- Uploading data to the Raspberry Pi server when a network connection is available (Online mode).
- Temporarily storing data locally when Offline, allowing access via Bluetooth.
- Providing a platform for a chatbot application that helps users interpret their health data.
- Enabling doctors and nurses to access patient data through an intermediate

directory server.

1.3. Theoretical framework

This project is grounded in a range of theoretical foundations across the fields of embedded systems, Internet of Things (IoT), network protocols, edge/cloud computing, and intelligent user interaction technologies. These concepts support the design, implementation, and deployment of the home-based health monitoring system.

Embedded Systems and Real-Time Operation:

- At the core of the system lies the ESP32 microcontroller, which plays the role of sensing, processing, and transmitting physiological data.

Internet of Things (IoT) Architecture:

The system follows a layered IoT architecture, composed of:

- Perception Layer: Includes health sensors and the ESP32, which collects heart rate, SpO_2, and temperature data.
- Network Layer: Enables data transmission via Wi-Fi or Bluetooth. The MQTT protocol, a lightweight publish-subscribe model optimized for low-resource devices, is used for efficient communication between ESP32 and the Raspberry Pi in online mode.
- Application Layer: Includes the mobile app, chatbot services, FastAPI backend, and the intermediate directory server that supports remote access.

This architecture ensures modularity, scalability, and fault tolerance under varying network conditions.

Edge and Cloud Computing:

The Raspberry Pi serves as an edge computing node, responsible for processing and storing real-time data close to the source, reducing latency and minimizing cloud dependency. It hosts:

- A Mosquitto MQTT Broker for message queuing,
- A FastAPI backend to provide RESTful APIs,
- A MySQL database to manage user and sensor data.

Meanwhile, a cloud-based VPS (Virtual Private Server) acts as an intermediate server to support remote lookup and access, particularly for healthcare professionals

who need to retrieve patient data from distributed environments.

Mobile Application and Chatbot Integration:

- The Flutter mobile application allows users to visualize real-time and historical health metrics. It functions in both online (Wi-Fi) and offline (Bluetooth) modes, ensuring continuous access to data.
- Integrated within the app is a rule-based chatbot, which leverages natural language inputs and API communication to provide users with personalized health insights based on their vital signs. This enhances the interactivity and usability of the system, especially for non-expert users.

Distributed System Architecture:

The system adopts a distributed system architecture, with loosely coupled components - ESP32 devices, Raspberry Pi, mobile app, and VPS-hosted services interacting through well-defined protocols and APIs. This architecture promotes robustness, scalability, and flexibility, allowing the system to function reliably in both connected and disconnected environments.

1.4. Members and Role Assignments

Đạt, Duy Hoàng: Responsible for setting up the Raspberry Pi, designing the MySQL database, developing the FastAPI backend, building the Flutter mobile application, and implementing the Intermediate (Directory) Server.

Việt Hoàng, Lý Lộc: In charge of programming the ESP32 to read sensor data, send data via Wi-Fi, and handle Bluetooth communication.

Duy Hoàng: Develops the chatbot, integrates it into the mobile application, and manages API communication.

Nguyên: Writes the report and organizes the project documentation.

1.5. Schedule and Milestones

Table 1.1.5.1: Schedule and Milestones

Time	Tasks and Deliverables
1/7/2025 - 4/7/2025	Analyze requirements and design overall system architecture
5/7/2025 - 24/7/2025	System Development

24/7/2025 - 28/7/2025	Finalize report and test the complete system in real-world scenarios

2. Project Execution

2.1. IoT Service Model

The system adopts a layered IoT service model consisting of three main components: the perception layer (this is the physical layer), the network layer, and the application layer.

- At the perception layer, the ESP32 microcontroller is responsible for collecting health-related data such as heart rate, blood oxygen saturation (SpO2), and body temperature using integrated sensors.
- The network layer handles data transmission through either Wi-Fi or Bluetooth, depending on the system's mode. In online mode, when a Wi-Fi connection is available, the ESP32 publishes sensor data to a Mosquitto MQTT Broker running on the Raspberry Pi. In offline mode, when Wi-Fi is unavailable, data can later be accessed via Bluetooth by the mobile application.
- At the application layer, the Raspberry Pi acts as an edge server, running a FastAPI backend and a MySQL database to store and manage patient records. A mobile application, developed with Flutter, allows users to view their health data and interact with an integrated chatbot that can answer basic health-related questions. Additionally, a lightweight directory server enables doctors and nurses to look up patient server addresses remotely, enabling secure and flexible access to health information from multiple locations.

This multi-layered architecture ensures that the system remains functional and useful even in the absence of a constant internet connection, providing a reliable and scalable IoT-based health monitoring service.

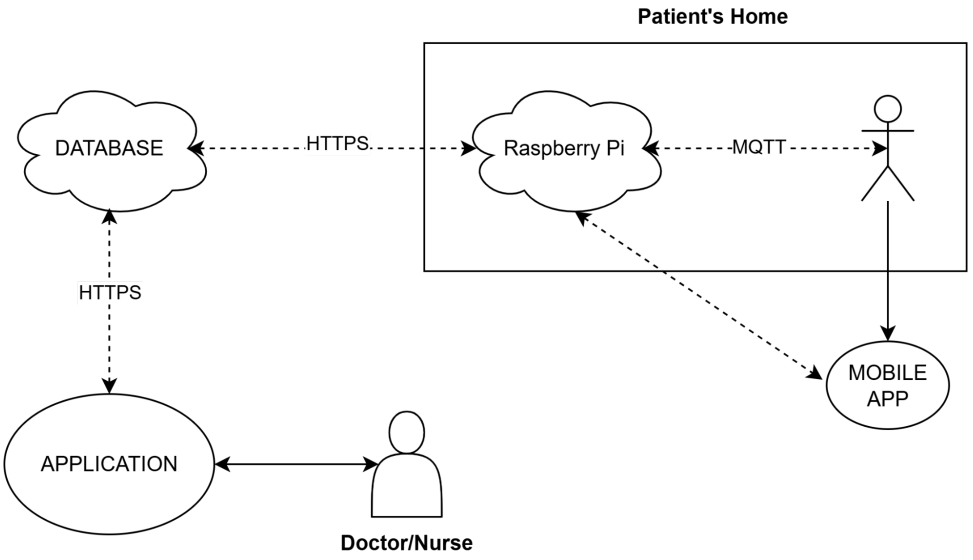


Figure 2.2.1.1: Software flowchart

This is a comprehensive IoMT (Internet of Medical Things) system designed for efficient remote health monitoring and empowering patients in managing their own health.

At home, patients wear a compact ESP32-based device that collects vital signs. The device stores data offline when outside and automatically uploads it via Wi-Fi using MQTT once connected. The data is sent to a Raspberry Pi at home.

The Raspberry Pi acts as the local hub, performing two key tasks: (1) securely forwarding health data via HTTPS to a central cloud database, where doctors can monitor and analyze it through a specialized app with an integrated chatbot; (2) directly communicating with the patient's mobile app over the local network for near-instant access to health data. This app also includes a chatbot to help patients understand their metrics.

A dynamic DNS server supports the system by mapping each Raspberry Pi's IP, ensuring reliable communication between the central system and individual patients.

2.2. Data Processing

Data processing in this system occurs at two main levels:

Local Processing on ESP32:

- Sensor readings (heart rate, SpO_2 and temperature) are collected every 30 minutes or on-demand via a push button.

- In Offline mode, data can be accessed via Bluetooth from the mobile app.

Edge-Level Processing on Raspberry Pi:

- When the system is in Online mode, data is transmitted from the ESP32 to the Raspberry Pi using the MQTT protocol.
- The Raspberry Pi processes the incoming data and stores it in a MySQL database, ensuring secure and structured data storage for further analysis and visualization.
- Mobile application retrieves this data from the Raspberry Pi to display real-time readings and historical trends.

2.3. Service Implementation

Raspberry Pi:

- Runs Mosquitto MQTT Broker to handle communication between ESP32 and the Pi.
- FastAPI provides API endpoints for the mobile app and the intermediate server.
- MySQL stores user accounts, patient info and health indicators.

Mobile App:

- Developed using Flutter, supporting both Online mode (Wi-Fi) and Offline mode (Bluetooth).
- Displays health data in real time and allows historical trend analysis.
- Integrated chatbot using API to provide basic health-related advice based on user queries.

Intermediate Server:

- The system is hosted on a VPS and uses a MySQL database to store mappings such as Patient ID, Raspberry Pi domain name, Patient Name,...
- APIs are implemented using FastAPI to handle GET and POST requests.
- These APIs support doctor and nurse authentication, as well as patient lookup, allowing remote access to patient health data from home environments.

2.4. System Design

ESP32 -> (Wi-Fi/Bluetooth) -> Mobile App -> Raspberry Pi -> Intermediate Server -> Doctor App.

Hardware:

- Hardware Architecture: ESP32 (Sensors) -> Raspberry Pi (MQTT Broker + FastAPI + MySQL) -> Intermediate Server (API) -> Doctor's PC App.

Software:

- ESP32: tasks for sensor reading, Bluetooth and Wi-Fi.
- Raspberry Pi: FastAPI backend, MQTT, MySQL database.
- Mobile App: Dual-mode data access, chatbot, user authentication.

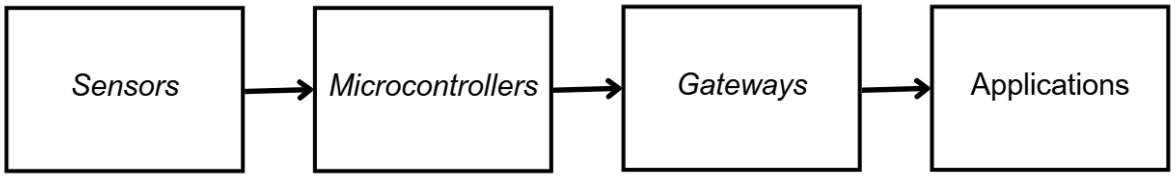


Figure 2.2.4.1: System Block Diagram

The system is divided into four main blocks: Sensors, Microcontrollers, Gateway, and Applications.

First, the Sensor block includes devices such as the MAX30100 and DHT11, which are responsible for collecting the user's biological parameters like heart rate, SpO_2, and body temperature. The data gathered from these sensors is sent to the Microcontroller block, specifically the ESP32. Here, the ESP32 performs preliminary data processing (such as noise filtering, averaging, etc.) and formats the information for transmission to the next device. It also handles data communication via Wi-Fi or Bluetooth, depending on the system's configuration.

Next is the Gateway block, represented in this system by the Raspberry Pi. This device receives data from the ESP32 and can either temporarily store it or forward it directly to a cloud server. The gateway acts as a relay station, allowing data from IoT devices to be integrated into a larger processing system while ensuring continuity and security.

Finally, the Application block consists of software applications running on the user's or doctor's mobile devices. Through an internet connection, the health data is displayed visually, allowing users to monitor their metrics, receive alerts, and enabling doctors to make remote diagnoses. Additionally, the cloud server plays a

key role in storing data, managing accounts, and assigning access permissions between users and healthcare professionals.

3. Results

3.1. Data Acquisition (Sensor, Actuator, Controller)

ESP32 reads data (captured every 30 minutes automatically or manually via button press):

In OFFLINE mode, ESP32 devices send data every 30 minutes. Additionally, a button can be used to send data immediately via the Bluetooth connection. The data is stored on the phone, and when the phone connects to the same Wi-Fi network as the Raspberry Pi at home, it will automatically sync the data to the Raspberry Pi database, as shown in the image below.

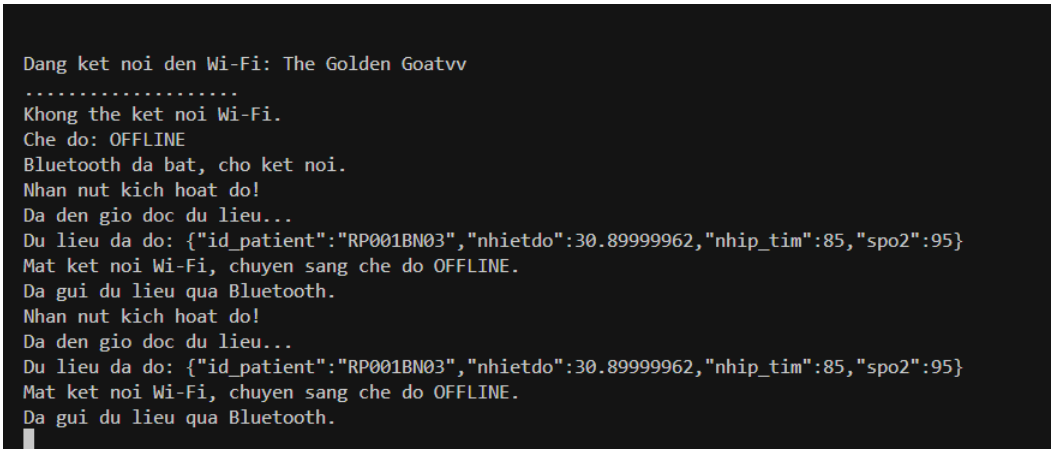
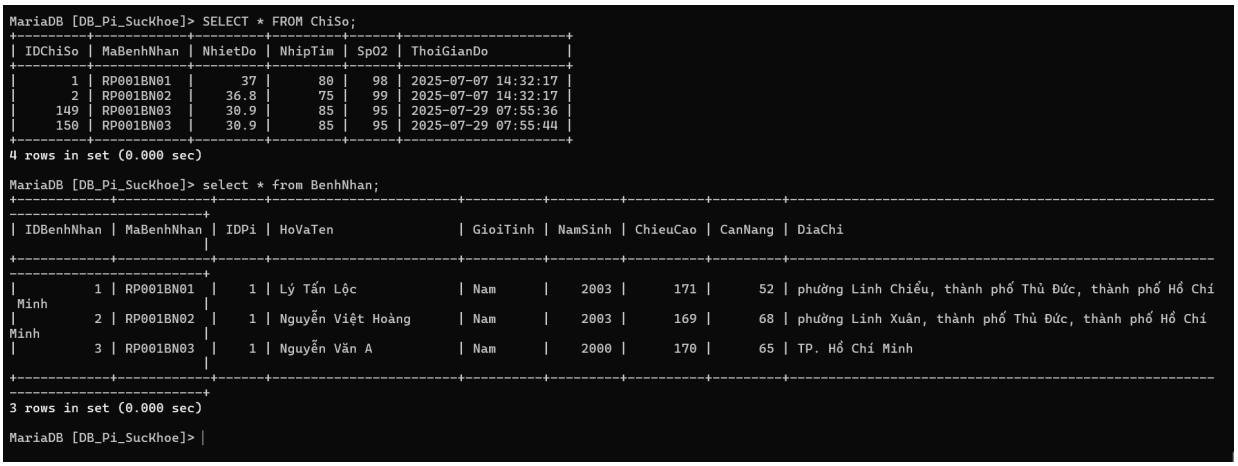


Figure 3.3.1.1: Data transmission in OFFLINE

In mode ONLINE, have a little bit like OFFLINE mode but when connected to the same WiFi networks as the Raspberry Pi that send into Raspberry Pi (MQTT) not through the phone.

Figure 3.3.1.2: Data transmission in ONLINE

Data after send to Raspberry will stored at the local storage on Rasp



```
MariaDB [DB_Pi_SucKhoe]> SELECT * FROM ChiSo;
+---------+------------+---------+---------+------+---------------------+
| IDChiSo | MaBenhNhan | NhietDo | NhipTim | SpO2 | ThoiGianDo          |
+---------+------------+---------+---------+------+---------------------+
|       1 | RP001BN01  |      37 |      80 |   98 | 2025-07-07 14:32:17 |
|       2 | RP001BN02  |    36.8 |      75 |   99 | 2025-07-07 14:32:17 |
|     149 | RP001BN03  |    30.9 |      85 |   95 | 2025-07-29 07:55:36 |
|     150 | RP001BN03  |    30.9 |      85 |   95 | 2025-07-29 07:55:44 |
+---------+------------+---------+---------+------+---------------------+
4 rows in set (0.000 sec)

MariaDB [DB_Pi_SucKhoe]> select * from BenhNhan;
+------------+------------+------+-----------------------+----------+---------+----------+---------+-------------------------------------------------------------------------
-------------------------+
| IDBenhNhan | MaBenhNhan | IDPi | HoVaTen               | GioiTinh | NamSinh | ChieuCao | CanNang | DiaChi
                         |
+------------+------------+------+-----------------------+----------+---------+----------+---------+-------------------------------------------------------------------------
-------------------------+
|          1 | RP001BN01  |    1 | Lý Tấn Lộc            | Nam      |    2003 |      171 |      52 | phường Linh Chiểu, thành phố Thủ Đức, thành phố Hồ Chí
 Minh                    |
|          2 | RP001BN02  |    1 | Nguyễn Việt Hoàng     | Nam      |    2003 |      169 |      68 | phường Linh Xuân, thành phố Thủ Đức, thành phố Hồ Chí
Minh                    |
|          3 | RP001BN03  |    1 | Nguyễn Văn A          | Nam      |    2000 |      170 |      65 | TP. Hồ Chí Minh
                        |
+------------+------------+------+-----------------------+----------+---------+----------+---------+-------------------------------------------------------------------------
-------------------------+
3 rows in set (0.000 sec)

MariaDB [DB_Pi_SucKhoe]>
```

Figure 3.3.1.3: Data stored locally on Raspberry Pi after being received

3.2. Network and Communication

We use the FastAPI to can use for Hospital's side can connect with data from Raspberry from far distance so Render help us with this solution is create server in the middle so if you have connected with wifi and have the url that can connect with each others. Below that show you and the Terminal of Render server on running.

```
All logs   Search
Aug 2 11:16:51 PM  ==> Deploying...
Aug 2 11:17:02 PM  ==> Running 'uvicorn main:app --host 0.0.0.0 --port 10000'
Aug 2 11:17:09 PM  INFO:     Started server process [67]
Aug 2 11:17:09 PM  INFO:     Waiting for application startup.
Aug 2 11:17:09 PM  INFO:     Application startup complete.
Aug 2 11:17:09 PM  INFO:     Uvicorn running on http://0.0.0.0:10000 (Press CTRL+C to quit)
Aug 2 11:17:09 PM  INFO:     127.0.0.1:51506 - "HEAD / HTTP/1.1" 405 Method Not Allowed
Aug 2 11:17:11 PM  ==> Your service is live 🎉
Aug 2 11:17:11 PM  ==>
Aug 2 11:17:11 PM  ==> ///////////////////////////////////////////////////////////
Aug 2 11:17:11 PM  ==>
Aug 2 11:17:11 PM  ==> Available at your primary URL https://sic-api-healthcare.onrender.com
Aug 2 11:17:11 PM  ==>
Aug 2 11:17:11 PM  ==> ///////////////////////////////////////////////////////////
Aug 2 11:17:12 PM  INFO:     35.197.118.178:0 - "GET / HTTP/1.1" 200 OK
```

Figure 3.3.2.1: The terminal of the Render server while it is running.

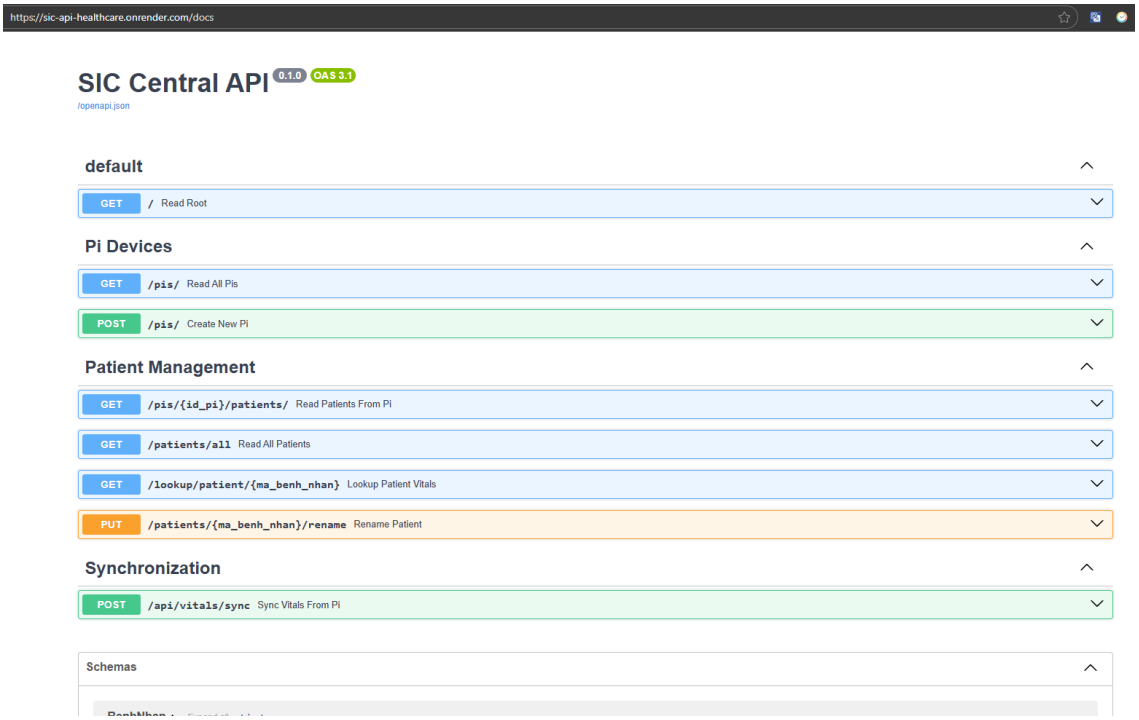


Figure 3.3.2.2: The FastAPI server is up and running, and has been successfully connected.

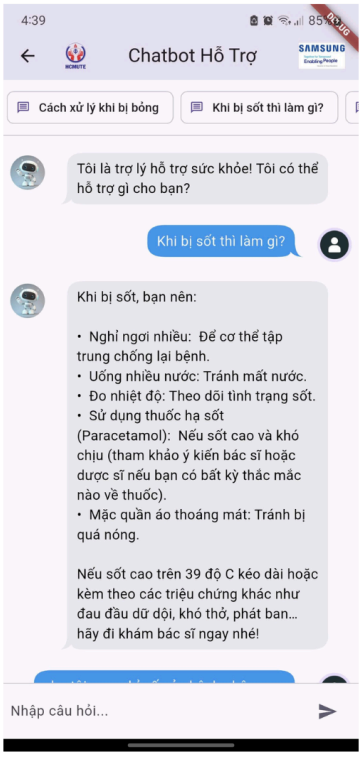
The image above shows that the server is live and connected. We can use it to test whether the FastAPI server is successfully connected to the central database we created earlier. This can be verified through the interface shown.

```
Aug 2 11:17:11 PM  ==> //////////////////////////////////////////////////////
Aug 2 11:17:12 PM  INFO:     35.197.118.178:0 - "GET / HTTP/1.1" 200 OK
Aug 2 11:18:01 PM  INFO:     14.169.236.247:0 - "GET / HTTP/1.1" 200 OK
Aug 2 11:18:02 PM  INFO:     14.169.236.247:0 - "GET /favicon.ico HTTP/1.1" 404 Not Found
Aug 2 11:18:09 PM  INFO:     14.169.236.247:0 - "GET /docs HTTP/1.1" 200 OK
Aug 2 11:18:09 PM  INFO:     14.169.236.247:0 - "GET /openapi.json HTTP/1.1" 200 OK
Aug 2 11:18:34 PM  INFO:     14.169.236.247:0 - "GET /patients/all HTTP/1.1" 200 OK
```

Figure 3.3.2.3: Retrieve patient list, rename patient,...

This is the server terminal output when we interact with FastAPI - for example, retrieving all patients, renaming a patient, and so on.

Mobile app running



We are currently performing tests using a series of simple questions to ensure that the function behaves as expected and fulfills all defined requirements. This step helps validate the logic and reliability of the implementation before proceeding to more complex scenarios.

3.3. Hardware Implementation

Below, we show the hardware product, which consists of a device that collects data from all sensors.

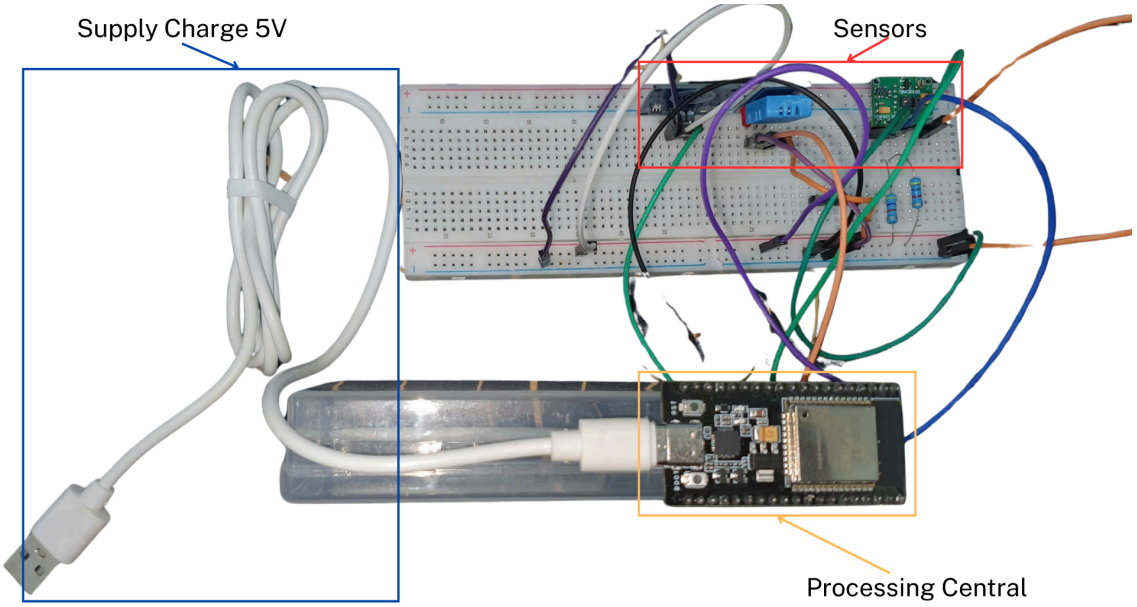


Figure 3.3.3.1: Hardware prototype showing sensor connections, supply charge 5V, and processing central

The devices will be collected from patients.

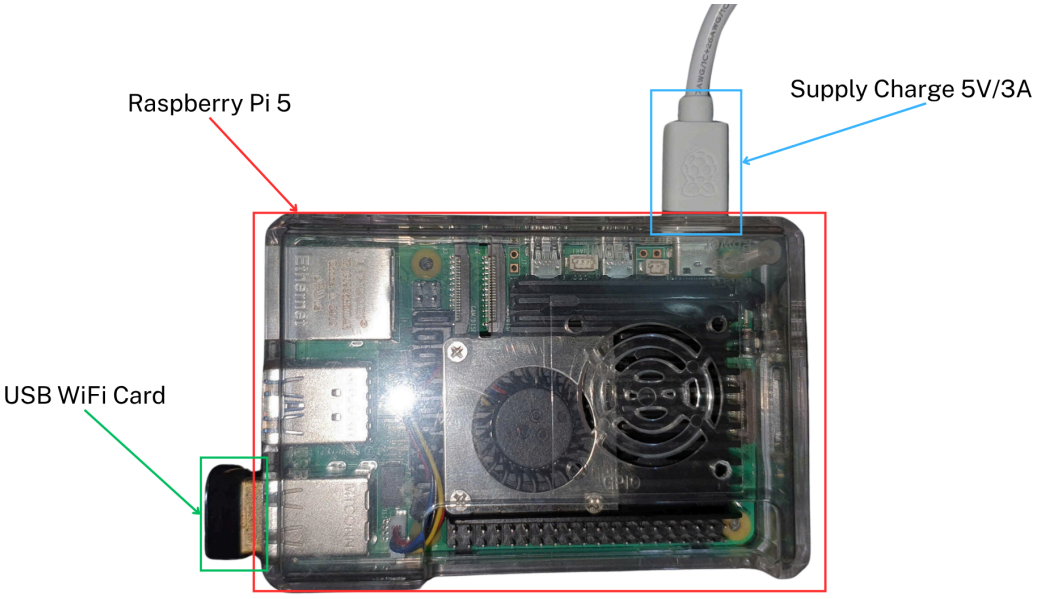


Figure 3.3.3.2: Raspberry Pi stores and sends data to the central server.

The Raspberry Pi hardware stores all the data and sends it to the central server, allowing the hospital side to manage it.

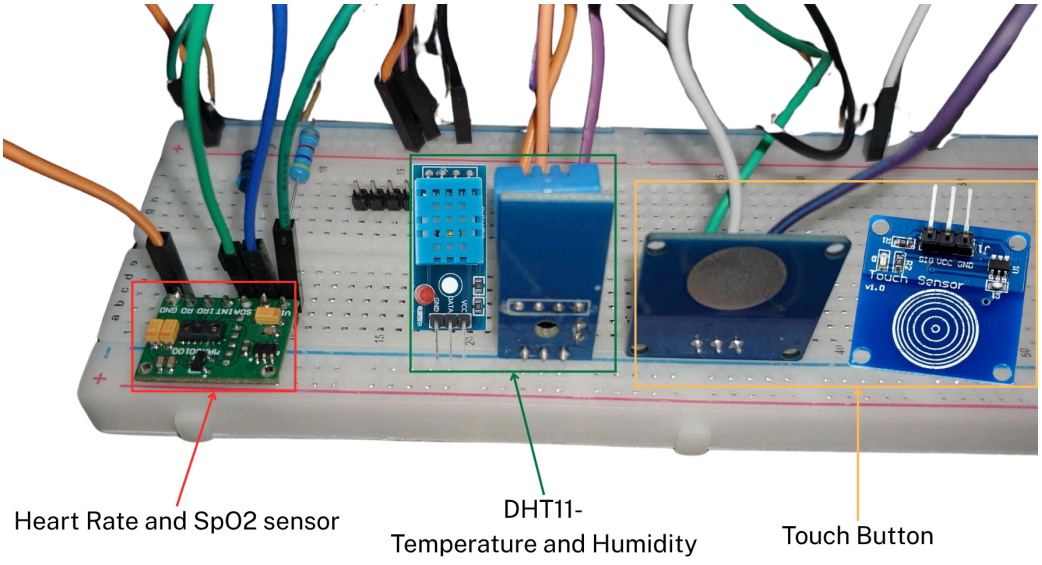


Figure 3.3.3.3: Sensors collect and send patient data to the Raspberry Pi.

The sensors collect data from the patient, store it, and send it to the Raspberry Pi.

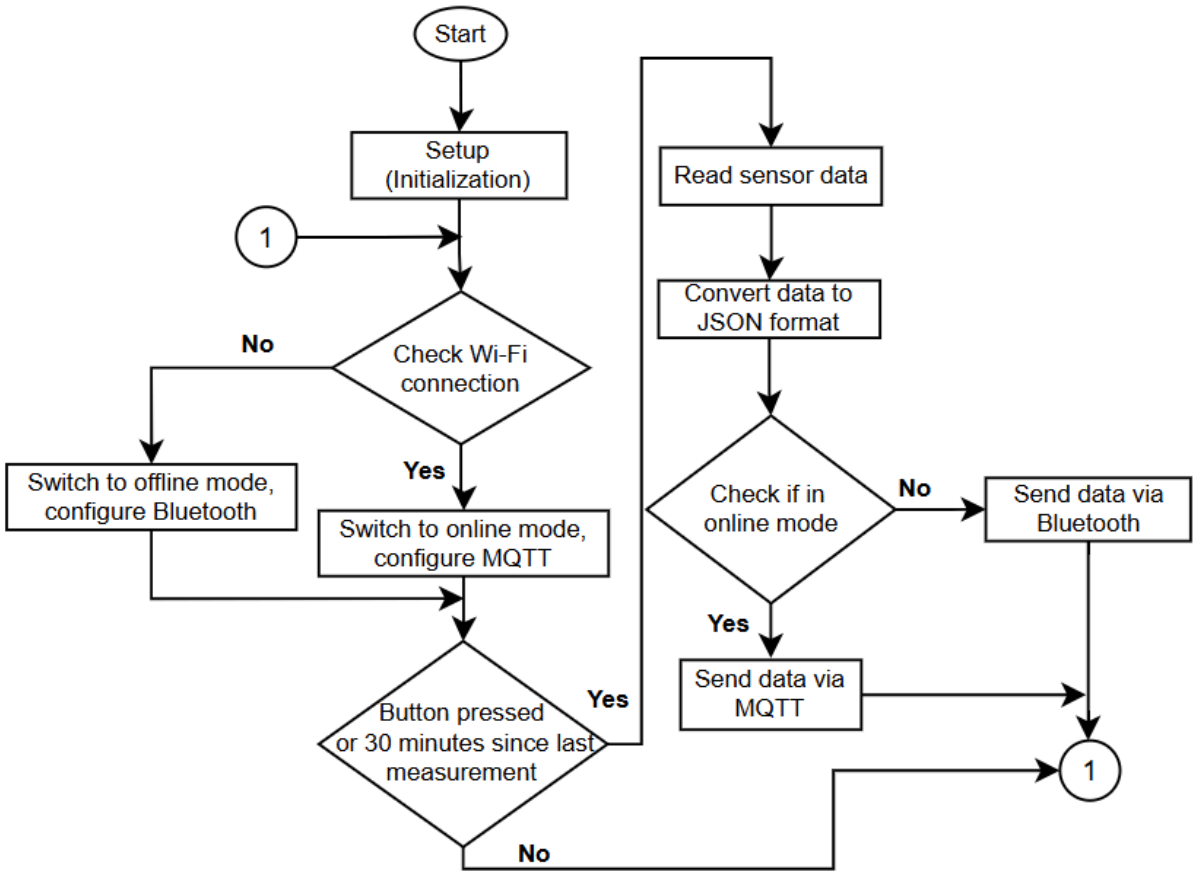


Figure 3.3.3.4: Hardware algorithm flowchart

When the system starts up, it performs the necessary initialization procedures such as configuring the sensors, button, and communication interfaces. Immediately afterward, the device checks for a Wi-Fi connection. If the connection is successful, the device switches to online mode and sets up the MQTT configuration to prepare for sending sensor data to the server. If it fails to connect to Wi-Fi, the device switches to offline mode, enables Bluetooth, and waits for an external device to connect in order to transmit data later.

During operation, the system continuously checks whether the button has been pressed or if 30 minutes have passed since the last measurement. If either condition is true, the device proceeds to read data from sensors such as temperature, heart rate, and SpO2. After collecting the data, the device processes and packages the information into a JSON string.

Next, the device checks its current operating mode. If it is in online mode, the data is sent to the server via the MQTT protocol. If it is in offline mode, the data is sent via Bluetooth to a mobile device if connected. Finally, once the data handling and transmission process is complete, the system returns to the main loop to continue operation, waiting for the next trigger.

3.4. Testing and Improvements

- Tested with 3 ESP32 devices + 1 Raspberry Pi (1 family)
- Verified stable operation in both Online and Offline modes

4. Projected Impact

4.1. Accomplishments and Benefits

- Enables remote health monitoring for families
- Reduces primary healthcare costs.
- Provides App for multiple households.

4.2. Future Improvements

- Integrate AI for advanced health analytics
- Expand sensor set (blood pressure, ECG).
- Integrate voice chat for easier for older people.

5. Team Member Review and Comment

NAME	REVIEW and COMMENT
Phan Duy Hoàng	
Lý Tấn Lộc	
Ngô Thành Đạt	
Nguyễn Việt Hoàng	
Lưu Nguyễn Thảo Nguyên	

6. Instructor Review and Comment

CATEGORY	SCORE	REVIEW and COMMENT
IDEA	__/10	
APPLICATION	__/30	

RESULT	__/30	
PROJECT MANAGEMENT	__/10	
PRESENTATIO N & REPORT	__/20	
TOTAL	__/100	