Vietnam National University Ho Chi Minh City
Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

*

GRADUATION THESIS

An IoT-based Remote Patient Monitoring System for Tackling the Dengue Fever Outbreak

Major: Computer Science & Engineering

THESIS COMMITTEE	:	**CLC KHMT 4**
SUPERVISOR	:	**Le Lam Son, Ph.D**
REVIEWER	:	**Quan Thanh Tho, Ph.D**

—o0o—

STUDENT 1	:	**Do Nguyen Dat - 1752160**
STUDENT 2	:	**Nguyen Hoang Long - 1852164**

Ho Chi Minh City, November 2022

Declaration of Authenticity

We declare that this research is our own work, conducted under the supervision and guidance of Ph.D. Le Lam Son. The result of our research is legitimate and has not been published in any form prior to this. All materials used within this research are collected by ourselves from various sources and are appropriately listed in the references section.

In addition, within this research, we also used the results of several other authors and organizations. They have all been aptly referenced.

In any case of plagiarism, we stand by our actions and will be responsible for it. Ho Chi Minh City University of Technology, therefore, is not responsible for any copyright infringements conducted within our research.

Ho Chi Minh, Sep 2022
Authors
Do Nguyen Dat
Nguyen Hoang Long

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to our supervisor, Ph.D. Le Lam Son, who read through our multiple modifications and helped us make sense of it all.

We would like to thank all of the teachers in the Faculty of Computer Science and Engineering, in particular, and all of the teachers at Ho Chi Minh City University of Technology in general, for their dedication to teaching and assisting us in grasping the core knowledge and understanding the role of an engineer in life so that we can continue to pursue our passion and achieve our goals.

We would like to express our gratitude to our parents, who always create good conditions for our development and encourage us when we have challenges in both our studies and life. Besides, we want to thank all of our friends for guiding and assisting us during our times at the university.

Ho Chi Minh, Sep 2022
Authors
Do Nguyen Dat
Nguyen Hoang Long

Abstract

The Internet of Things (IoT) refers to the network of physical objects, devices, and appliances that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. This network allows these objects to collect and share data about their surroundings and the actions they take, enabling them to be monitored and controlled remotely. Examples of IoT devices include smart thermostats, smart appliances, and wearable fitness trackers. One of the key features of the IoT is that it allows for the automation of many tasks and processes, which can help to make our lives more efficient and convenient. For example, a smart thermostat can be programmed to automatically adjust the temperature in your home based on the time of day and your usual schedule, and a smart fridge can keep track of the food inside and remind you when you're running low on certain items. In addition to making our lives more convenient, the IoT also has the potential to provide a wealth of data that can be used for a variety of purposes. For instance, data from wearable fitness trackers can be used to monitor and improve our health, and data from smart city sensors can be used to help city planners make more informed decisions about urban infrastructure. Overall, the IoT is a rapidly growing field that has the potential to transform the way we live and work. While there are some concerns about the security and privacy of the data generated by IoT devices, the overall benefits of the technology are likely to outweigh any potential drawbacks.

As the global population grows, so do healthcare costs, which has led to a modernization of the healthcare system based on Internet-of-Things (IoT) technologies, which are composed of intelligent, tiny, low-power, micro and nanotechnology sensor nodes put inside or around the human body, are being developed as a result of advanced electronic device technologies and significant advancements in wireless communication. The data is transferred from these sensors to personal servers where it can be stored, processed, and re-transmitted. Using a wireless medium is substantially less expensive than using a cable connection, which is its alternative. IoT-based solutions are employed in a variety of industries, including the military,

sports, home, and most significantly, medicine. Healthcare is undergoing a silent revolution thanks to IoT medical apps. Value-based medicine becomes a reality when access to patients' vitals is always available, wherever they may be.

One prime example of the potential of IoT in healthcare is its use in combatting dengue fever. Dengue fever is a viral infection transmitted by mosquitoes, and it is a major public health concern in many parts of the world. The disease causes a range of symptoms, including high fever, severe headaches, and muscle and joint pain. In severe cases, it can lead to dengue hemorrhagic fever, which can be fatal. One way IoT sensors can be used is to monitor the presence of mosquitoes, which are the primary vector for the transmission of dengue fever, in a specific area. These sensors can collect data on the population of mosquitoes and alert authorities when the population reaches a level that could lead to an outbreak. In addition to monitoring the mosquito population, IoT can also be used to track the movement of individuals who have been infected with dengue fever. This information can help health officials take timely and effective action to reduce the spread of the disease. However, more importantly, IoT technologies are able to provide timely and accurate information that can help healthcare providers make more informed decisions from monitoring patients.

During a dengue fever outbreak, hospitals can be overwhelmed by the number of patients seeking treatment. This can put a strain on the hospital's resources, such as beds, staff, and medical supplies. In some cases, hospitals may be unable to accommodate all of the patients who are seeking treatment, leading to long waiting times and delays in care. IoT can be used in monitoring patients with dengue fever through the use of wearable sensors. These sensors can be attached to the patient's body and used to collect data on various physiological parameters, such as body temperature, heart rate, and respiratory rate. This data can be transmitted wirelessly to a central hub, where it can be analyzed and used to monitor the patient's health status. For example, if the data collected by the sensors indicates that the patient's body temperature is rising, this could be an early warning sign of a potential dengue fever infection. The healthcare provider can then use this information to take timely and appropriate action, such as providing the patient with medication or other treatment to reduce the risk of complications.

This research presents a development of remote patient's monitoring application with the help of IoT devices. We also focus on learning and proposing a solution for multi-streaming data processing system architecture for IoT to alert and analyze the patient's condition based on data collected from IoT devices, as well as a doctor-patient management system for more efficient and time-saving

procedures during dengue fever outbreaks.

Contents

Declaration of Authenticity

Acknowledgments **i**

Abstract **ii**

1 Introduction **1**

1.1 Background and Motivation 1

1.1.1 The situation with dengue fever 1

1.1.2 The rise of IoMT 4

1.2 Related Works 6

1.2.1 Possible approaches 6

1.3 Proposed solution and Scope 11

1.3.1 Thesis Structure 15

1.3.2 Work Distribution 16

2 Methodologies and Theoretical Background **18**

2.1 IIRA Three-tier Architecture 20

2.1.1 The edge tier 22

2.1.2 The platform tier 24

2.1.3 Backend and the enterprise tier 30

2.1.4 Final architecture 32

2.2 Monolith to Micro-service Architecture 33

2.2.1 The differences 33
2.2.2 Switching to microservices 36
2.3 Others 39

3 Design and Implementation 40
3.1 Requirement Analysis 40
3.1.1 Involved Parties and Users 40
3.1.2 Functional Requirements 42
3.1.3 Non-functional Requirements 43
3.1.4 Use-case view 44
3.1.5 Process view 46
3.2 Database Design 50
3.2.1 Conceptual design 51
3.2.2 Logical design 52
3.3 Hardware and Device Implementation 54
3.3.1 Micro-controller (ESP8266 Wemos D1 Mini) 54
3.3.2 Temperature sensor (MLX90614) 56
3.3.3 Heart rate sensor (MAX30100) 58
3.3.4 OLED screen (SSD1306) 61
3.3.5 Pin Lipo 64
3.3.6 Communication standard between ESP and modules 65
3.3.7 Pin connection between micro-controller and sensors 67
3.3.8 Flow chart of wearable device 69
3.3.9 Implementation 70
3.4 Backend Implementation and Deployment 74
3.4.1 Creating ThingsBoard server 75
3.4.2 ThingsBoard's Actor System 76
3.4.3 Configuring ThingsBoard's Rule Engine 77
3.4.4 Backend and API designs 78

3.4.5 Alarm and Patient's Status 80
3.4.6 User Authentication and Authorization 81
3.4.7 Deployment 82
3.5 Frontend Implementation and Deployment 83
3.5.1 Frameworks 83
3.5.2 Deployment 85
3.6 Scaling 85
3.7 Final look 87

4 Result and Evaluation **95**
4.1 Performance Testing 95
4.2 Device testing 97

5 Conclusion and Future Work **100**
5.1 Achievement 100
5.2 Future Plans & Development 101

List of Figures

1.1 Vismo PVM-2703 6

2.1 Initial model based on IIRA three-tier architecture 22
2.2 Edge tier consisting of wearables 24
2.3 ThingsBoard architecture 28
2.4 Backend and frontend architecture 32
2.5 Architecture of the whole system 32
2.6 Microservices architecture for the whole system 37
2.7 Kafka APIs [12] 38

3.1 Use-case diagram of the system 45
3.2 Sequence diagram of user registration 47
3.3 Sequence diagram of real-time data streaming 49
3.4 Enhanced entity relationship diagram of the database 51
3.5 Logical design of the database 52
3.6 ESP8266 Wemos D1 Mini 54
3.7 Functional Block Diagram [14] 55
3.8 MLX90614 temperature sensor [15] 56
3.9 Pin descriptions [15] 57
3.10 Typical application schematics [15] 58
3.11 MAX30100 heart rate sensor [16] 58
3.12 Pin configurations [16] 59
3.13 Functional diagram [16] 60

3.14 oled SSD1306 [17] 61
3.15 SSD1306 Block Diagram [17] 63
3.16 Oscillator Circuit and Display Time Generator [17] 64
3.17 Pin Lipo 64
3.18 Pin connection between ESP and temperature sensor 67
3.19 Pin connection between ESP and oled screen 67
3.20 Pin connection between ESP and heart rate sensor 68
3.21 Flow chart of whole wearable system 69
3.22 Schematic diagram of whole circuit 71
3.23 PCB layout 72
3.24 3D Simulation of whole circuit 72
3.25 View front and back of device 73
3.26 Interface of Arduino IDE 73
3.27 Result device display 74
3.28 Dashboard of thingsboard 74
3.29 ThingsBoard's Role Hierachy [20] 76
3.30 ThingsBoard's Rule Chain Implementation 78
3.31 Framework comparisons [22] 83
3.32 Scalable components 86
3.33 Home page 87
3.34 Login and Signup pages 88
3.35 Dashboard 89
3.36 Patient Registration 90
3.37 Patient Detail 90
3.38 Patient's real time telemetry line chart 91
3.39 List of nurses 92
3.40 View of facilities (rooms and beds) 93
3.41 List of current devices 94

4.1 Device measure heart rate Microlife-b2-basic 98
4.2 Device measure body temperature Microlife 98

List of Tables

1.1 Summary and comparisons of related works 9

2.1 Open-source IoT platforms comparison 26

3.1 Stakeholders and their roles 42
3.4 Pin description MLX90614 [15] 57
3.5 Pin description MAX30100 [16] 60
3.6 Parameters of components in the circuit 68
3.7 Medical Rules and Dengue Fever Treatment [21] 80

4.1 Device test results 98

1 Introduction

1.1 Background and Motivation

1.1.1 The situation with dengue fever

Dengue fever is a viral infection that is becoming increasingly common around the world. Estimates of the global incidence of dengue infections per year have been increasing drastically in both tropical and subtropical countries [1]. This disease is transmitted when the mosquitoes that are called Aedes mosquitoes, which live in dirty places like unclean water or vases that carry the virus and attack humans [2]. Dengue causes a range of symptoms, including high fever, severe headaches, and muscle and joint pain. In severe cases, it can lead to dengue hemorrhagic fever, which can be fatal. The disease is a major public health concern, and hospitals can struggle to cope during outbreaks due to the large number of patients seeking treatment and the challenges of diagnosing and treating the disease [3].

Dengue fever is a growing concern in Vietnam, with officials reporting 8,248 cases, a 33.3 percent increase in the number of cases in Ho Chi Minh City from January 1 to May 19 compared to the same period last year, according to a Sai Gon Giai Phong (SGGP) report Friday [4]. The southern metropolis is experiencing red alerts due to the high number of dengue fever cases, and the number of severe cases has reached 175, an increase of 500 percent over the same period. The number of deaths from dengue fever in Ho Chi Minh City has also risen, with 6 deaths reported, up 300 percent from last year. Nationally, Vietnam has reported a total of 25,694 cases of dengue fever, including 13 deaths [5]. This represents a significant increase in the number of cases compared to previous years, and is causing concern among public health officials.

Currently, Hanoi records more than 1,000 new dengue fever cases each week, the highest since the beginning of the year on a monthly basis, with 12 deaths related to the disease. Statistics from the Hanoi Center for Disease Control (CDC) show that the number of dengue fever cases has been increasing rapidly in the area since the beginning of October. All districts have reported patients, with more than 1,000 new cases reported each week from 7-13/10 and 14-20/10. The number of cases has increased 3.2 times compared to the same period in 2021 and exceeds the average for the 2019-2021 period. In terms of death cases, the Hanoi CDC reported 12 cases, while no deaths from the disease were reported in 2021. Cases are concentrated in some districts and surrounding areas such as Dan Phuong, Thanh Oai, Thuong Tin, Thanh Tri.

Another report from the Ministry of Health showed that from the beginning of the year to date, the country has recorded more than 270,000 dengue fever cases and 108 deaths. Compared to the same period in 2021, the number of cases has increased 4.8 times and the number of deaths has increased by 87 cases. The forecast is that the number of cases and deaths will continue to develop complexly due to the peak season of the annual epidemic from now until November.

Dengue is caused by one of four closely related viruses, and infection with one of these viruses provides immunity to that specific virus but not the others. This means that a person can be infected with dengue multiple times, increasing the risk of severe illness. Symptoms of dengue typically begin 4-7 days after infection and can range from mild to severe. Mild symptoms include fever, severe headache, joint and muscle pain, rash, and mild bleeding. Because there is no specific treatment for dengue, hospitals often focus on relieving symptoms and preventing complications. This can help improve patient outcomes and reduce the risk of severe illness or death from dengue by providing supportive care to patients. This may involve providing pain relief, fluids, and electrolytes to help relieve symptoms and prevent dehydration. In severe cases, dengue can lead to a potentially fatal complication called dengue hemorrhagic fever, which is characterized by high fever, severe abdominal pain, difficulty breathing, and bleeding. By then, hospitalization may be necessary to monitor the patient's condition and provide more intensive treatment. This still involves providing fluids and electrolytes to prevent dehydration, but accompanied with monitoring vital signs, and providing oxygen and other supportive care as needed. Medications such as pain relievers and blood pressure medications may also be necessary. For these cases, patients may need to be hospitalized for several days to receive supportive care and monitor their condition.

In addition to providing medical care, hospitals may also take steps to prevent

the spread of dengue. This may include implementing infection control measures, such as isolating infected patients and providing personal protective equipment to healthcare workers. Hospitals may also work with local health departments and other organizations to control the mosquito population and prevent the spread of the disease. However, if an outbreak of dengue occurs, hospitals may take a number of steps to manage the influx of patients and provide appropriate care. This may include implementing infection control measures, such as isolating infected patients and providing personal protective equipment to healthcare workers. Hospitals may also need to increase their capacity to treat patients by setting up additional beds, hiring additional staff, and coordinating with other hospitals to ensure that patients receive the care they need. In some cases, hospitals may need to prioritize patients based on their condition and the availability of beds. For example, patients with severe dengue may need to be treated in the hospital, while those with milder symptoms may be able to be treated at home or in an outpatient setting.

Nevertheless, in severe outbreaks of dengue, even prioritizing patients and increasing capacity may fall short in an event of a bed shortage, which poses a challenge for hospitals during an outbreak. On the other hand, it doesn't help with the traditional way medical staff like nurses and doctors typically take vital signs in hospitals. These measurements provide important information about the patient's overall health and can help healthcare workers monitor their condition and identify potential issues, but this comes with a cost of equipments and time. To take a patient's vital signs, a nurse will typically use a thermometer to measure the patient's temperature, a stethoscope to listen to their heartbeat and breathing, and a blood pressure cuff to measure their blood pressure. They may also use a pulse oximeter to measure the patient's oxygen saturation. The specific method for taking each vital sign will vary depending on the type of equipment used and the patient's condition. For example, a nurse may use a digital thermometer to measure a patient's temperature, or a manual thermometer if the patient is unable to cooperate. Similarly, a nurse may use a manual or automatic blood pressure cuff to measure a patient's blood pressure, depending on the patient's condition and the equipment available.

In most cases, a nurse can take a patient's vital signs in just a few minutes. The specific time required to take each vital sign will vary depending on the patient's condition and the type of equipment used. For example, taking a patient's temperature with a digital thermometer may take only a few seconds, while taking their blood pressure with a manual cuff may take a few minutes. However, in addition to these vital signs, nurses have to also monitor specific symptoms of dengue, such as severe headache, joint and muscle pain, rash, and mild bleeding.

They may also assess the patient for signs of severe dengue, such as difficulty breathing, abdominal pain, and bleeding. It becomes clear that this process when multiplied with hundreds to thousands of patients during an outbreak, hospitals do not have enough staff to manage the number of patients and staff are required to work long hours and have little time for rest and recovery. In such situations, medical staff may become fatigued and stressed, which can affect their ability to provide high-quality care to patients. Overwork can also lead to burnout and turnover among medical staff, which can further exacerbate the situation.

Given the concern that such outbreaks are already happening and are becoming increasingly common, there exists the need for systems and solutions that can help medical workers by providing real-time data and insights. These systems and solutions can facilitate remote care and improve the efficiency and effectiveness of the healthcare system, which is particularly important during outbreaks when hospitals may be overwhelmed by the number of patients seeking treatment. By providing timely and accurate information, these systems and solutions can help healthcare providers make more informed decisions and take timely and effective action to prevent and manage outbreaks of dengue fever. In this way, they can play a crucial role in reducing the spread of the disease and saving lives

1.1.2 The rise of IoMT

IoT, or the Internet of Things, refers to the integration of connected devices and technology in various fields, including healthcare. IoMT - the Internet of Medical Medical Things is that integration of connected devices and technology in the field of healthcare. It refers to the use of sensors, wearable devices, and other technologies to collect and transmit data from the human body, as well as the use of this data to improve patient care and medical outcomes. IoMT includes a wide range of technologies and applications, including remote monitoring, telemedicine, and predictive analytics. Rapid advancement in the telecommunication section, such as the introduction of 4G LTE has provided IoMT opportunities to flourish. The global IoMT market size was USD 41.17 billion in 2020 and is project to grow to USD 187.60 billion in 2028 [6] as companies are actively focusing on investing in IoT technology along with government organizations. For example, some examples of companies like Philips, Medtronic, Cisco or Johnson Johnson are leading providers of IoMT solutions, offering a range of products and services designed to help healthcare providers collect, share, and analyze data. The IoT deployments in healthcare industry are expected to grow faster than any other industry and is likely to have a global economic impact of USD 1.6 trillion by 2025.

Moreover, the World Telecommunication Development Conference projected that by the end of 2020, 40% of the IoT technology would be healthcare-driven.

Among the advantages that IoMT could bring, IoMT can facilitate remote monitoring and telemedicine, allowing healthcare providers to monitor and treat patients remotely. This can help reduce the need for hospitalization and free up beds for patients who need them, as well as allowing healthcare providers to provide care to patients in remote or underserved areas. Another benefit is the multi-streaming data processing of a large amount of IoT events. An IoT medical app is more than just a standard mobile app that runs on a smartphone screen like many other types of mobile apps. With sensors or other smart objects, a smartphone app or a web application for service providers, we could create an entire system to combat most medical disease like dengue.

Such system with real-time data collection and analysis enabled, it has the potential to transform the healthcare industry and improve the quality of care for patients. For example, IoT devices and sensors can be used to monitor the availability of beds and other resources in real time, allowing medical workers to quickly identify areas of the hospital that are over capacity and transfer patients as needed. IoT can also be used to monitor patient vital signs and alert healthcare providers to potential issues, allowing them to provide timely care and prevent complications. One example of such technology in action is the use of wearable sensors to monitor patients with chronic conditions, such as diabetes or heart disease. These sensors can be attached to the patient's body and used to collect data on various physiological parameters, such as body temperature, heart rate, and respiratory rate. This data can be transmitted wirelessly to a central hub, where it can be analyzed and used to monitor the patient's health status. If the data collected by the sensors indicates that the patient's blood sugar levels are rising or falling outside of the normal range, this could be an early warning sign of potential complications. The healthcare provider can then use this information to take timely and appropriate action, such as adjusting the patient's medication or providing additional treatment to prevent the development of complications. In the next section, related works, we intend to dive into details regarding existing IoT systems and approaches that facilitate the use of monitoring patients for our own purpose regarding patients with the dengue fever.

1.2 Related Works

1.2.1 Possible approaches

Continuous monitoring systems are a vital tool in the medical field, providing real-time information and insights that can help doctors and nurses to monitor patients' health status and take timely and appropriate action. There are many different continuous monitoring systems available, each with its own unique features and capabilities. However, one system that is particularly widely used in the medical field is the Life Scope Vismo PVM-2703, which is a continuous monitoring system that is designed to provide accurate and reliable information about a patient's health status. It is typically used by doctors and nurses to monitor various physiological parameters, such as body temperature, heart rate, and respiratory rate. The system uses sensors that are attached to the patient's body, and the data collected by these sensors is transmitted wirelessly to a central hub, where it can be analyzed and used to monitor the patient's health status.

Figure 1.1: *Vismo PVM-2703*

One of the key advantages of the Life Scope Vismo PVM-2703 is its accuracy. The system is designed to provide highly accurate information about a patient's health status, which is essential for making timely and appropriate decisions about the patient's care. This accuracy is achieved through the use of advanced sensors and algorithms, which are carefully calibrated to provide accurate and reliable data. However, it does not provide much more efficiency as medical workers still need to manually record the parameters every ten minutes which is susceptible to human errors. Not only that, despite being one of the cheapest options it still costs around $5000 and hence is difficult for developing countries or poor, rural

areas to have multiple in one treatment unit. The alternative would be a wireless monitoring system which is cheap, efficient and can be mass-produced to be easily available for all places to install and use.

Given the recent outbreaks of infectious diseases such as Covid-19 and dengue fever, they have highlighted the need for effective monitoring and control measures to prevent the spread of these diseases. In response to this need, researchers and healthcare providers have developed a range of solutions that use wearable technology to collect physiological data from the human body. This data can be used to monitor the health of individuals and identify potential issues that may indicate the presence of an infectious disease. The table below introduces several papers about wearable designs and applications collecting physiological parameters from the human body.

Authors	Objectives	Design	Limitations, Strengths	Outlook measures
Qureshi, Fayez et al. Sensors. (2020) [7].	1. Use of Internet of medical things to design wearable devices. 2. Front end API for Biomedical wearable	Wearable PPG, Accelerometer, EMG sensors, Edge computing using Raspberry pi, Dashboard as a physician	Sample of Biomedical signals, design factors such as economic cost, human precautions	Arduino, Microphone and cloud space, sensors cost, comparative studies
Petrovic et al. (2020). IcE-TRAN [8]	IoT-based solution to provide indoor safety through social distancing, mask detection and temperature sensing contact less	Arduino Uno, thermal camera, Raspberry pi, Open CV, MQTT, Mask detection algorithm, mobile Application	Limitations in performance due to the number of processed frames per second	Accuracy, frame rate for mask detection, social distance and temperature sensor
Anto Arockia Rosaline R. (2020). Emerald insight [9]	Purpose Geo fencing and tracking of Covid zones to monitor the people and alerting on mobile	A Virtual perimeter monitoring system with wireless infrastructure. Bluetooth, wifi, GPS, Mobile application	Bluetooth option to ensure data security but the short distance and privacy are the concerns	Proximity range accuracy at different range levels. Central monitoring system
S. Nubenthan, C. Shalomy (2017) [10].	A wireless Continuous Patient Monitoring System for Dengue: Wi-mon	Wearable ECG, temperature, SpO2, blood pressure sensors, WBAN, Dashboard	Cheaper compared to existing systems, accuracy needs to be improved	Accuracy, Arduino, sensors cost, comparative studies

Authors	Objectives	Design	Limitations, Strengths	Outlook measures
Quoc H. Nguyen, Quang Dang, Chuan Pham, Tien-Dung Nguyen, Hang Nguyen, Arveity Setty, Trung Quoc Le (2020) [11].	Developing an Architecture for IoT Interoperability in Healthcare: A Case Study of Real-time SpO2 Signal Monitoring and Analysis	SpO2 sensor, Fluke ProSim signal generator, Thingsboard integrated with Spark	Proposed system utilized the IoT gateway, integration and data converter solutions to enhance the IoT interoperability	Data transmission, Spark analysis and evaluations of gateways and data converters

Table 1.1: *Summary and comparisons of related works*

Firstly, papers by Qureshi and Petrovic focused on a topic that is highly relevant and presents a detailed and comprehensive overview of the use of IoMT in the design of wearable devices, including a discussion of the different types of wearable devices that are currently available and the technologies and solutions that are used to collect, share, and analyze data. This information is highly useful and informative, as it provides a clear understanding of the key issues and challenges related to the use of IoMT in healthcare. They also included a discussion of the front-end API for biomedical wearable devices, which is a crucial component of IoMT systems. The API enables wearable devices to communicate with other systems and devices, allowing data to be collected, shared, and analyzed in real-time. This is a key feature of IoMT systems, as it enables healthcare providers to access timely and accurate information about patients' health status, which can help them to make more informed decisions and take timely and appropriate action.

The work of Rosaline in addition to providing a detailed overview of virtual

perimeter monitoring systems also includes a discussion of the benefits of using these systems in healthcare. One of the key benefits of virtual perimeter monitoring systems is the ability to monitor individuals in real-time and alert them when they approach or cross a virtual perimeter. This is a crucial aspect of virtual perimeter monitoring systems in healthcare, as it enables healthcare providers to monitor and manage the health and safety of individuals in real-time. This can help to prevent the spread of infectious diseases by alerting individuals when they are at risk of coming into contact with an infected individual. This can help to reduce the transmission of the disease and protect the health and safety of individuals.

Other works of Nubethan and Quoc Nguyen's team presented a detailed and comprehensive overview of the development of an architecture for IoT interoperability in healthcare. Quoc Nguyen's included a case study on the use of IoT technology for real-time SpO2 signal monitoring and analysis. This is a valuable and insightful example of how IoT technology can be used in healthcare to improve the quality of care and the health outcomes of patients. On the other side, Nubethan discussed benefits of using a wireless continuous patient monitoring system for dengue, including the ability to monitor patients in real-time and alert healthcare providers when they are at risk of developing complications from the disease.

Of all the approaches above, it seems the best solutions boil down to Wireless Body Sensor Networks (WBSN) or IoT-based remote applications that tackles dengue and helps stop the spreading of the virus. In many papers, sample of biomedical signals, such as heart rate, blood oxygen levels, and muscle activity, can be used to evaluate the performance and accuracy of wearable biomedical devices and sensors. Sensors, such as wearable PPG sensors, would be used to collect data on a person's SpO2 levels in real time. These sensors would be integrated into wearable devices, such as wristbands or smart watches, which the person would wear while going about their daily activities. Edge computing devices like Raspberry Pi or other microcontrollers would be used to process and analyze the data in real time. This would allow the data to be analyzed and interpreted without the need for a powerful and expensive central server, providing faster and more accurate results.

The processed data would then be sent to a cloud-based data storage and analysis system, where it could be accessed and analyzed by healthcare providers. Next, Front-end APIs, or application programming interfaces, provide a way for different software applications to communicate and exchange data. In the context of biomedical wearable devices and sensors, a front-end API could be used to

allow these devices to send data to a central dashboard or other application for analysis and interpretation. For example, a dashboard for physicians can provide a user-friendly interface for viewing and analyzing data from biomedical wearable devices. This can help physicians to quickly and easily understand a patient's health status and identify potential issues, allowing them to provide timely and appropriate care.

After having carefully studied each of these solutions, their strengths as well as their limitations, we were inspired to know that such a system was achievable, although only at limited scale. We decided to use this opportunity to study and self-build a remote monitoring system using IoT, keeping in mind the goal of helping medical workers tackle dengue outbreaks. This project will involve several steps and considerations which we will discuss next.

1.3 Proposed solution and Scope

The development of this application will require the use of several IoT sensors and wireless communication technologies. The use of IoT sensors will enable the application to collect data on the health of patients with dengue fever, including their temperature, pulse, and blood pressure. This data will be transmitted wirelessly to healthcare providers, who can use it to monitor the health of patients in real time. It is important to identify the specific needs and goals of the system. This may involve conducting a needs assessment to understand the type of data that needs to be collected, the type of patients that will be monitored, and the specific goals of the system. As for patients with dengue fever, our solution will satisfy three main goals:

- **Remote monitoring using IoT devices**: By creating a wearable consisting of sensors, each collecting data remotely using IoT devices, can help to combat dengue fever by providing real-time data and insights, facilitating remote care, and improving the efficiency and effectiveness of the healthcare system. IoT devices, such as wearable sensors and smart phones, can be used to collect data on a person's vital signs and symptoms, such as fever, headache, and muscle pain. This data can be transmitted to a central dashboard or other application, where it can be analyzed and interpreted by healthcare providers.
- **Patient management system for medical workers**: Through our website or mobile application, medical workers can monitor registered patients under

their care in real time. The dashboard displays all patients, allowing medical workers to quickly and easily identify patients who may need additional care or attention. Patients with worsened conditions are prioritized and highlighted, allowing medical workers to focus on providing care to those who need it most.

- **Data Analysis and Alerts**: In addition, an alert system is built into the website or mobile application to warn medical workers about critical cases. If a patient's condition deteriorates or they experience a medical emergency, the alert system will notify medical workers, allowing them to provide timely and appropriate care. This can help to prevent complications and improve patient outcomes.

Advantages to existing systems:

- Improved patient care: By providing real-time data and insights, the website or mobile application can help medical workers to identify patients who may need additional care or attention, and provide timely and appropriate intervention. This can help to prevent complications and improve patient outcomes.
- Increased efficiency: By automating the process of monitoring and managing patients, the website or mobile application can help medical workers to save time and effort. This can allow medical workers to focus on providing care to patients, rather than spending time on administrative tasks.
- Better coordination: The website or mobile application can help to facilitate communication and coordination among medical workers. For example, if a patient's condition deteriorates, the alert system can notify all relevant medical workers, allowing them to quickly respond and provide care.
- Improved data analysis: The website or mobile application can help to collect and analyze data from registered patients, allowing medical workers to identify trends and patterns that may indicate the onset of a disease or other health issue. This can help medical workers to provide early intervention and support to patients, and improve the overall effectiveness of the healthcare system.

The next step is to select the appropriate sensors and devices for the remote monitoring system. This may involve choosing wearable sensors that can collect

data on a person's vital signs and other health metrics, as well as devices such as smart watches or smart phones that can receive and transmit data from the sensors. It is important to select sensors and devices that are accurate, reliable, and easy to use, as well as being appropriate for the specific needs and goals of the system.

Once the sensors and devices have been selected, the next step is to implement the infrastructure for the remote monitoring system. This may involve setting up edge computing devices, such as Raspberry Pi or other microcontrollers, to process and analyze the data from the sensors in real time. It may also involve setting up cloud-based data storage and analysis systems, as well as user-friendly interfaces, which is responsible for providing an intuitive and user-friendly experience for healthcare providers and other users, allowing them to access and interpret the data from the sensors and devices in a convenient and efficient manner. These are the screens and controls that users interact with to access and interpret the data from the sensors and devices. For example, a dashboard may provide an overview of a patient's health status, while a detailed view may allow users to see more specific data and trends.

Finally, as an afterthought, we can consider the economic and human factors involved in building a remote monitoring system using IoMT. This may involve evaluating the cost of the sensors and devices, as well as the cost of implementing and maintaining the infrastructure for the system. It may also involve considering the potential risks and benefits to patients and healthcare providers, and implementing appropriate safeguards and protocols to protect patient privacy and security.

Since IoT sensors are the key technology that will be used in this project, as they will be used to collect data on the health of patients with dengue fever, we would need to learn about the different types of IoT sensors that are available, as well as how to integrate these sensors into the application. For the backend, we would need to learn about a server-side programming language, such as Java or Python or Nodejs with the aid of frameworks such as Express.js, Spring or Flask. Additionally, we would need to learn about a database management system, either SQL (PostgreSQL, MySQL) or noSQL (MongoDB), to store and manage the data collected by the IoT sensors. For the frontend, aside from the basics of HTML, CSS, and JavaScript, frontend frameworks such as React or Angular are necessary for developing the user interface of the application. Additionally, data visualization library, such as D3.js or Chart.js, can be used to present the data collected by the IoT sensors in a clear and easily understandable format.

The end result of this project will be the development of an application that

can monitor patients with dengue fever in real time, providing valuable information and insights that can help healthcare providers to manage and prevent the spread of the disease. This application has the potential to greatly improve the efficiency and effectiveness of the healthcare system, and it could potentially save many lives by preventing the spread of dengue fever. Before we move on to the next chapter, there are in total five chapters in this thesis proposal that for each we would like to discuss the content of.

1.3.1 Thesis Structure

Chapter	Content
1. Introduction	Literature survey and a brief introduction about the motivation and goal of the thesis
2. Methodologies and Theoretical Background	Architecture of the system
3. Design and Implementation	Architecture design of the proposed approach for the problem statement of the project
4. Evaluation	Conclusion of the thesis and further development
5. Conclusion	Conclusion of the thesis and further development

Chapter 2 is where the architecture and design of the remote monitoring system are described in detail, including how the hardware and software components should interact, as well as how the data analysis and alarm system are implemented after data is collected from the sensors and devices.

In Chapter 3, the system design and implementation will be discussed in depth. This chapter provides a step-by-step guide to the development of the remote monitoring system, including requirement analysis and database designs. This chapter will also include detailed descriptions of the hardware and software components that were used, as well as the algorithms and models that were developed to analyze the data from the sensors and devices.

Chapter 4 is about how tests are done and evaluated. The results and performance of the remote monitoring system will be presented and discussed. This chapter will also include any feedback and comments from healthcare providers (if possible) and other users of the system, as well as any suggestions for improvements or enhancements that could be made in the future.

Finally, in Chapter 5, we conclude the thesis status and future work. This chapter will summarize the key findings and contributions of the project, and discuss the potential impact and implications of the remote monitoring system for healthcare and other areas. Additionally, this chapter will outline potential future work that could be pursued based on the results of the project, including further research and development that could be carried out to improve the system.

1.3.2 Work Distribution

Writing report: both team members are responsible for writing the final report for the project. This may involve collecting and organizing data and information from other team members, as well as writing the report itself and ensuring that it meets the required format and standards.

Business logic and Database design/implementation: both team members are responsible for designing and implementing the business logic as well as database for the application. This involves working with the application backend to develop a database schema that meets the needs of the application, as well as implementing the database and ensuring its performance and reliability.

Individual contributions - **Do Nguyen Dat:**

- Sensors: responsible for selecting and implementing the sensors that will be used in the system. This may involve researching and evaluating different sensors, as well as determining which sensors are appropriate for the specific needs and goals of the system.
- IoT wearable and IoT gateways: responsible for creating the IoT wearable and IoT gateways that will be used in the system, as well as implementing the necessary hardware and software components for the wearable devices and gateways.
- Thingsboard server: This person or team would be responsible for implementing and maintaining the thingsboard server that will be used in the system. This may involve setting up the server and configuring it to collect and store data from the sensors and wearable devices, as well as maintaining the server and ensuring its performance and reliability.

Individual contributions - **Nguyen Hoang Long:**

- Application backend: responsible for developing and maintaining the backend components of the application, including the server-side code and the database. This may involve working with other team members to develop and implement the overall architecture of the application, as well as writing code and managing the database.
- UI head: responsible for designing and implementing the user interface of the application, including the visual design and the user experience. This

may involve working with other team members to develop wireframes and prototypes of the user interface, as well as creating and implementing the final design of the application.

- Web server: responsible for implementing and maintaining the web server that will be used in the system. This may involve setting up the web server, configuring it to serve web pages and other content, and ensuring its performance and reliability.

2 Methodologies and Theoretical Background

Unlike traditionally, where you would analyze business requirements first before the design step, we'd decided to put the architecture design before any analysis step. One reason is that the architecture provides the foundation on which the rest of the system will be built and by defining the high-level structure and components of the system, the architecture helps to ensure that the system will be able to meet the functional and non-functional requirements of the business or organization. In real world situations it also provides a common reference point for all stakeholders, helping to ensure that everyone involved in the project has a shared understanding of the system's capabilities and limitations. Additionally, by establishing the architecture up front, the requirements analysis phase can focus on how the specific requirements of the business or organization will be mapped to the components and capabilities of the architecture, rather than trying to define the architecture and requirements simultaneously. This can help to improve the efficiency and effectiveness of the requirements analysis process.

Architecture provides a blueprint for how the different components of the application will interact with each other. A well-designed architecture will ensure that the application is scalable, maintainable, and efficient. As the application grows and becomes more complex, a well-designed architecture will allow for the easy addition of new features and capabilities without negatively impacting the overall performance of the application. This can be especially important for applications that are expected to handle a large amount of traffic or data, such as any healthcare system that continuously take in real time data from patients. After investing time and resources into researching and developing a good architecture, we found many different options available when it comes to choosing an architecture for the internet of things (IoT). Some of the most commonly used architectures include the following:

- Client-server architecture: In this architecture, devices and sensors connected to the IoT network are organized into a hierarchical structure, with a central server or "cloud" at the top and individual devices and sensors at the bottom. This architecture is commonly used in IoT systems that require high levels of reliability and scalability, as it enables the central server to manage and control the devices and sensors on the network.

- Publish-subscribe architecture: In this architecture, devices and sensors connected to the IoT network communicate with each other and with other parts of the system using a publish-subscribe model. This means that devices and sensors can publish data and other information to the network, and other devices and sensors can subscribe to this information and use it as needed. This architecture is commonly used in IoT systems that require high levels of flexibility and interoperability, as it enables devices and sensors to communicate with each other without the need for a central server or control point.

- Peer-to-peer architecture: In this architecture, devices and sensors connected to the IoT network are organized into a decentralized network, with no central server or control point. Instead, devices and sensors communicate directly with each other and share information and resources as needed. This architecture is commonly used in IoT systems that require low-latency communication and high levels of resilience, as it enables devices and sensors to continue functioning even if parts of the network are lost or damaged.

From these three, we decided that the client-server architecture is most suitable for our use-case. Designed to provide a high degree of reliability and scalability in the IoT system, the central server is responsible for managing and controlling the devices and sensors on the network, ensuring that they are able to communicate and exchange data with each other and with other parts of the system. This enables the system to handle large volumes of data and maintain high levels of performance, even as the number of devices and sensors on the network grows. It's also designed to provide a high degree of security in the IoT system. The central server acts as the gatekeeper for the network, controlling access to the devices and sensors and ensuring that only authorized users and devices are able to communicate with the network. This helps to protect against unauthorized access and other security threats.

The publish-subscribe architecture, on the other hand, has devices and sensors on the network are organized in a decentralized manner, with no central

server or control point. Instead, devices and sensors communicate directly with each other, using the publish-subscribe protocol or a similar protocol to exchange data and other information. Even though this architecture is designed to provide high levels of flexibility and interoperability, the reason we did not choose it is because it may require more complex algorithms and protocols to manage the network, without have the benefits of the client-server architecture. It's also worth noting that, the publish-subscribe protocol and the publish-subscribe architecture are related, but are not the same thing. The publish-subscribe protocol is a specific protocol that can be used to enable communication and data exchange among devices and sensors on an IoT network, whereas the publish-subscribe architecture is a broader concept that refers to the overall design and organization of the IoT system. Which means even with the client-server architecture, we can still implement publish-subscribe protocols.

When it comes to interoperability, which is the ability of different devices and sensors on an IoT network to communicate and exchange information with each other, regardless of their specific make, model, or manufacturer, while it is possible in the client-server architecture, it may require some additional effort and resources to achieve. To achieve interoperability in the client-server architecture, the central server must be able to communicate with devices and sensors from different manufacturers, using a range of different technologies and protocols. This may require the use of special adapters or gateways that can translate between the different technologies and protocols used by the devices and sensors. Alternatively, the central server may need to be designed to support multiple technologies and protocols, so that it can communicate directly with devices and sensors from different manufacturers. For this we found ThingsBoard opensource to be best at, which we will discuss later in this chapter, in the dedicated Thingboard section.

2.1 IIRA Three-tier Architecture

It's also helpful to look at an IoT architecture as layers, with each layer representing a different aspect of the system. The Industrial Internet Reference Architecture (IIRA), is a framework and set of guidelines for the design and development of industrial internet of things (IIoT) systems. The IIRA is designed to provide a common set of principles and standards that can be used to guide the development of IIoT systems, and to enable interoperability and collaboration among different IIoT systems. According to IIRA, we can divide a system into three tiers:

- The edge tier: The edge tier is the outermost layer of the three-tier architecture pattern, and is responsible for collecting data from the sensors and devices that are located at the edge of the network. The Edge Tier is responsible for managing the communication with the devices and sensors, and for collecting and preprocessing the data that is sent to the other tiers of the IIRA. The edge tier typically includes a range of microcontrollers, single-board computers, and embedded devices that are used to collect, process, and transmit the data from the sensors. The edge tier also includes any edge computing algorithms and models that are used to perform lightweight processing and analysis of the data from the sensors.

- The platform tier: The platform tier is the middle layer of the three-tier architecture pattern, and is responsible for providing the infrastructure and support services that are needed to deploy, operate, and manage the system. The platform tier typically includes a range of cloud-based services and tools that are used to host and operate the system, as well as the tools and APIs that are used to integrate the system with other systems and applications. The platform tier also includes any security and privacy mechanisms that are used to protect the system and the data that is being processed and stored.

- The enterprise tier: Or the front-end tier architecture for the IoT system would consist of the user interface that is used to interact with the system. This tier could include a web-based interface, a mobile app, or a desktop application that allows users to view and control the sensors, as well as view and analyze the data that is collected by the sensors. The front-end tier could also include any natural language processing (NLP) algorithms and models that are used to process and analyze the user's queries and feedback.

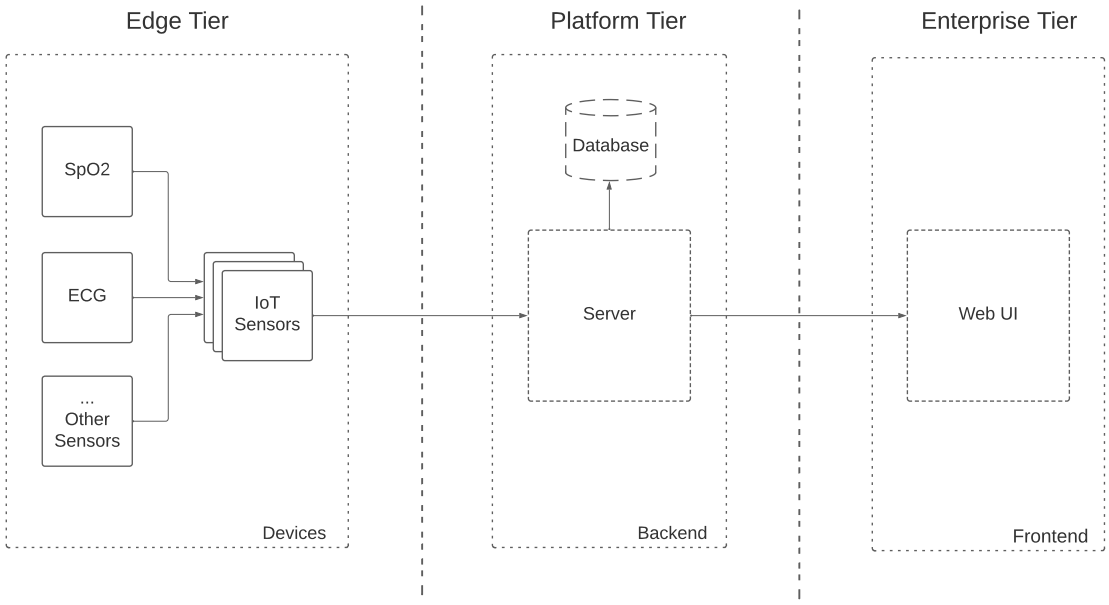

Figure 2.1: *Initial model based on IIRA three-tier architecture*

We'll continue to discuss how we designed our system based on this three-tier model in the following sections.

2.1.1 The edge tier

One of the main benefits of the edge tier is that it allows the system to operate more efficiently and reliably. By performing some of the processing and analysis of the data at the edge of the network, the edge tier reduces the amount of data that needs to be transmitted over the network to the middle tier. This can reduce the amount of bandwidth and storage that is required by the system, and can improve the performance and responsiveness of the system. Another benefit of the edge tier is that it allows the system to operate more autonomously and independently. By performing some of the processing and analysis of the data at the edge of the network, the edge tier enables the sensors and devices to make decisions and take actions without the need for constant communication with the middle tier. This can enable the system to operate more effectively in situations where the network connectivity is limited or unreliable.

The components of the edge tier can vary depending on the specific needs and requirements of the system, but typically include the following:

- Sensors and devices: The edge tier of the IIRA architecture includes a range of sensors and devices that are used to collect data from the physical world. These sensors and devices may include sensors that measure temperature, pressure, humidity, light, sound, or other physical phenomena, as well as devices that control or actuate physical systems, such as motors, valves, or switches.

- IoT gateways: IoT gateways are devices that are used to connect the sensors and devices that are located at the edge of the network to the rest of the system. IoT gateways typically include a range of connectivity options, such as WiFi, Bluetooth, Zigbee, or cellular, that are used to connect the sensors and devices to the network. IoT gateways also typically include some processing and storage capabilities, such as microcontrollers, single-board computers, or embedded systems, that are used to collect, process, and store the data from the sensors. For example, these microcontrollers and single-board computers such as Arduino or Raspberry Pi are used to collect and process the data from the sensors, as well as single-board computers, such as Nvidia Jetson or Google Coral, that are used to perform more sophisticated processing and analysis of the data.

In our IoMT system, to collect vital signs data related to dengue fever, temperature sensors can be used to detect changes in body temperature, which can be indicative of various health conditions, such as fever, infection, or inflammation. SpO2 sensors can be used to detect changes in oxygen saturation, which can be indicative of various health conditions and heart rate sensors are used to detect changes in heart rate. In order to integrate the sensors and devices into a single, compact, and portable wearable. This would involve designing the circuit boards and components, as well as the firmware and software that are required to control the sensors and devices, and to transmit the data using an microcontroller acts as an IoT gateway through a Wifi network.

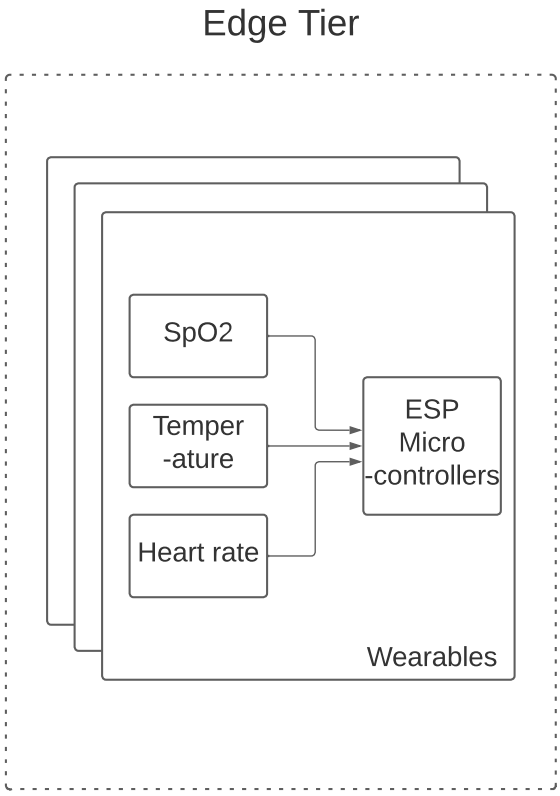

Figure 2.2: *Edge tier consisting of wearables*

2.1.2 The platform tier

The edge tier is very similar to the perception layer as they both are responsible for converting the raw data from these sensors and devices into meaningful information that can be used by other parts of the IoT system, such as the processing layer. The processing layer refers to the hardware and software components that are responsible for analyzing and making sense of the data collected by the perception layer. This layer typically consists of a range of different technologies and algorithms, including machine learning and data mining, that are used to extract useful insights and knowledge from the data.

In IIRA architecture, the platform tier is also responsible for the data processing layer, which collects the data from the IoT sensors and gateways located at the edge tier, and for storing the data in a central repository. This may involve using technologies, such as more IoT gateways, cloud computing platforms, and data storage systems, to collect, process, and store the data from the sensors. This

layer is also responsible for performing some initial processing and analysis of the data from the IoT sensors. This may involve using techniques, such as filtering, aggregation, and transformation, to clean, filter, and transform the raw data from the sensors, as well as using algorithms, such as machine learning, to perform classification, clustering, or regression on the data.

Choosing an open-source IoT platform can be an effective way to reduce costs and time at this step. However, choosing the right open-source IoT platform is not a simple task when the appropriate platform depends on the specific applications and context, and on the platform provider. To be effective, an IoT platform must have certain characteristics, such as the ability to manage and connect IoT endpoints, to collect and process data, and to provide data visualization and application development tools.

There are many surveys that review existing IoT platforms and evaluate their performance and capabilities. Some of these surveys focus on specific domains, such as smart buildings or agriculture, while others consider a broader range of factors. In this work, we focus on engineering aspects that are relevant to interoperability and interface development, using the Kafka and Spark platforms for big data processing. We also consider other factors such as ease of customization and configuration, device management, and security. Based on these requirements, we have identified and summarized the main features of four open-source IoT platforms in the table 2.1: ThingsBoard, Zetta, DeviceHive, and SiteWhere. These platforms are among the most appropriate for our purposes, based on their capabilities and features.

Platform	Architecture	Protocols	Security Management	Analysis	Database
ThingsBoard	Monolithic, Microservices	HTTP(s), REST, MQTT, CoAP, ModBus, OPC-UA, BLE, CAN	MQTT/HTTP(s) encryption, Device authentication, Credentials management	Connection with data analysis platforms and tools	Built-in SQL, NoSQL, Hybrid
DeviceHive	Monolithic	HTTP, MQTT, REST	Device authentication	No	NoSQL Plugin
Zetta	Monolithic	HTTP, REST	No	Yes	Need plug-ins
SiteWhere	Monolithic, microservices	HTTP, MQTT, AMQP, Stomp, CoAp	HTTP(s), Spring Security	Security	SQL, NoSQL plugin

Table 2.1: *Open-source IoT platforms comparison*

Each of the open-source IoT platforms that we have reviewed has its own strengths and weaknesses. The appropriate platform for a given scenario will depend on the specific requirements and goals of the system, as well as on the capabilities and features of the platform. In a healthcare environment, for example, it may be necessary to integrate a heterogeneous network of sensors and to support bidirectional communication between the platform and the devices. In this case, the platform must support multiple protocols and must be able to be customized to interact with the devices and external systems in an effective and efficient manner.

Support from developer communities is also an important factor to consider, as this can help ensure the continued development and improvement of the platform. Based on these criteria, ThingsBoard and SiteWhere are among the best platforms available. However, we have decided to use ThingsBoard as the core IoT platform for our system, due to its ability to be re-engineered for multiple applications.

ThingsBoard - An overview

ThingsBoard is an open-source platform designed to support the development and management of IoT systems. As a cloud-based IoT platform that provides a range of tools and services for managing and analyzing the data from the sensors and devices, ThingsBoard includes a range of features, such as device management, data visualization, data storage, data analysis, and alerting, that are used to collect, process, and analyze the data from the sensors and devices. ThingsBoard also provides a range of APIs and integration tools that are used to connect the system to other systems and applications.

Overall, ThingsBoard can act as the server component in an IoT system that follows a client-server architecture. It can also act as the processing layer by managing the communication between the devices and sensors on the network, and routing data and other information from the devices and sensors to the appropriate applications and services. Some benefits of using ThingsBoard cloud server are:

- ThingsBoard cloud is designed to be highly scalable, and can support a large number of devices and sensors, as well as a large number of users and applications. This means that ThingsBoard cloud is well-suited for applications and systems that require a high degree of scalability, such as large-scale IoT deployments or applications that are expected to grow and evolve over time.
- ThingsBoard cloud is highly flexible and customizable, and can be configured and customized to meet the specific requirements and needs of different applications and systems. This means that ThingsBoard cloud can be used to support a wide range of different IoT applications and systems, and can be adapted and modified to support new and emerging technologies and requirements.
- ThingsBoard cloud is designed to be highly secure, and includes a range of security features and mechanisms that are designed to protect the data and devices that are connected to the platform. This means that ThingsBoard cloud can be used to support applications and systems that require a high

level of security, such as applications that involve sensitive or confidential data, or applications that operate in highly regulated environments.

In addition, ThingsBoard was also designed to be a microservices architecture, which is a software design pattern that is used to build scalable and resilient systems by decomposing the system into a collection of small, independent, and loosely-coupled services that can be developed, deployed, and maintained independently. We will discuss ThingsBoard as a monolith and as a microservices near the end of this chapter.

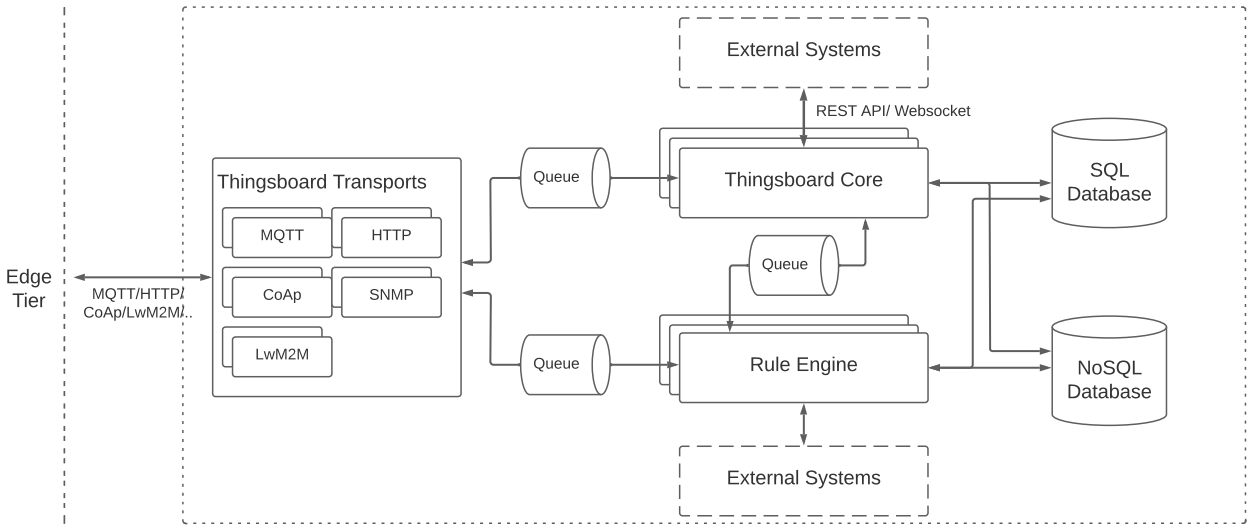

Figure 2.3: *ThingsBoard architecture*

The transport component of ThingsBoard is a software module that is responsible for connecting the sensors and devices to the platform, and for transmitting the data from the sensors and devices to the platform. The transport component of ThingsBoard includes support for various transports, such as MQTT, CoAP, HTTP, and WebSockets, which are used to connect the sensors and devices to the platform, and to transmit the data from the sensors and devices to the platform. MQTT (Message Queuing Telemetry Transport) is a lightweight and open-source messaging protocol that is designed for low-power, low-bandwidth, and low-latency devices. MQTT is widely used in the IoT and M2M (Machine-to-Machine) domains, and is supported by a range of devices, platforms, and tools. ThingsBoard supports MQTT as a transport, and includes a range of features, such as device provisioning, message routing, and data storage, that are designed to support MQTT-based devices and applications. Once the Transport receives

the message from device, it is parsed and pushed to durable Message Queue. The message delivery is acknowledged to device only after corresponding message is acknowledged by the message queue.

In addition to supporting various transports, the transport component of ThingsBoard also includes a range of features, such as device registration, message routing, and data storage, that are designed to support the specific requirements of the sensors, devices, and applications that are connected to the platform. For example, the transport component of ThingsBoard may include features, such as device provisioning, message handling, and data visualization, that are designed to support MQTT-based devices and applications.

The core component of ThingsBoard is a software module that is responsible for providing the core functionality and services of the platform. The core component of ThingsBoard includes a range of tools and services, such as device management, data visualization, data storage, data analysis, and alerting, that are used to collect, process, and analyze the data from the sensors and devices that are connected to the platform, enabling the platform to be used for a variety of purposes, such as remote monitoring, asset tracking, energy management, and supply chain optimization. ThingsBoard Core is responsible for handling REST API calls and WebSocket subscriptions.

Connected with the ThingsBoard Core is the rule engine - a software module that is responsible for executing rules and actions on the data that is collected from the sensors and devices that are connected to the platform. The rule engine of ThingsBoard enables users to define rules, using a simple and intuitive syntax, that can be used to process and analyze the data from the sensors and devices, and to trigger actions based on the data. Users can define rules that can be used to process and analyze the data from the sensors and devices, and to trigger actions based on the data. Some features of the rule engine are rule chaining, data filtering, and data transformation, that are designed to support the specific requirements.

One example of using the rule engine in our case is to define a rule that sends an alert to the system if the temperature of a device exceeds a certain threshold. We may define a rule that specifies that the rule should be executed whenever the temperature of a patient exceeds a certain threshold, such as 40 degrees Celsius. The rule may also specify that, when the rule is executed, an alert should be sent to the user or medical workers using a email or SMS to notify of the high temperature. In this case, the rule engine of ThingsBoard would monitor the data from the sensors and devices, and would detect when the temperature of the device exceeds

the specified threshold. When this occurs, it would execute the rule, and send the alert.

In general, ThingsBoard can be compared to a broker can be used in any architecture for IoT, including the client-server architecture. In the client-server architecture, the broker would typically be located on the central server or "cloud" at the top of the hierarchical structure. The broker would be responsible for receiving data and other information from the devices and sensors on the network, and for routing this data to the appropriate destination within the system. In this way, the broker plays a crucial role in enabling communication and data exchange within the client-server architecture.

2.1.3 Backend and the enterprise tier

WebSocket is a protocol that enables full-duplex, bi-directional communication between a web server and a web browser. WebSocket allows the web server and the web browser to exchange data in real-time, without the need for the web browser to constantly poll the web server for updates. This makes WebSocket an effective way of implementing real-time communication and other interactive features in web applications. In order to use WebSocket on the server-side, we need to ensure that our web server supports incoming WebSocket connections, and that this support is enabled and configured properly. This is due to the fact that a WebSocket connection is a continuous, long-running connection, it does not follow the Common Gateway Interface (CGI) server model. Instead, there must be a long-running process that is responsible for handling live WebSocket connections. In server models like CGI, this requires the use of a web server add-on that supports long-running processes for WebSocket connections. However, in a server environment like Node.js, which is already a long-running process, the addition of WebSocket support does not require any changes to the architecture, but rather just the use of a library that supports the WebSocket protocol. Node.js provides a powerful and flexible platform for building WebSocket-based applications, so we chose it for this purpose.

The backend built upon the Express.js framework, in a Node.js environment can connect to ThingsBoard either through REST API or through Websockets. Once the components and modules of the backend have been implemented, we choose a database management system (DBMS) that will be used to store and manage the data that is collected and processed by the backend. In this case, the DBMS is PostgreSQL, and the design and implementation of the DBMS will

involve creating the appropriate database schema, tables, and indexes, as well as configuring and optimizing the DBMS to support the data storage and management requirements of the backend.

Relational databases, such as MySQL and PostgreSQL, provide a wide range of features and capabilities that can be used to manage and analyze time-series data (IoT sensors are often used to collect time series data, as they provide a convenient and cost-effective way of collecting data from a wide range of physical systems and environments). However, PostgreSQL is widely considered to be more feature-rich than MySQL, and it provides support for a wider range of advanced SQL features, such as subqueries, triggers, and user-defined functions. MySQL, on the other hand, is more focused on providing basic SQL functionality, and it may not support all of the advanced features that are available in PostgreSQL. Another key difference between the two systems is the level of support for different data types. PostgreSQL provides support for a wide range of data types, including geometric, network, and text data types, as well as support for user-defined data types. MySQL, on the other hand, provides support for a more limited set of data types, and it may not be as well-suited to applications that require support for specialized data types.

Even though the ThingsBoard's web UI also provides the user-friendly interface for accessing and interacting with the platform. We won't be using the ThingsBoard's web UI but instead develop our own front-end using the React.js framework and TailwindCSS. REST API will be used to enable the front-end to interact with the backend, and to access and use the data that is collected and processed by the backend to stream through Websockets in real time. This would involve designing the various endpoints and resources of the API, and implementing the code and logic that is required to support the key features and functions of the API, including authentication, authorization, and validation. This would also involve designing the layout and appearance of the dashboard, as well as the various charts, graphs, and other visualizations that will be used to represent the data, and implementing the code and logic that is required to support the real-time creation, updating and deletion of the dashboard, and the interactions and controls that are available to manage users.

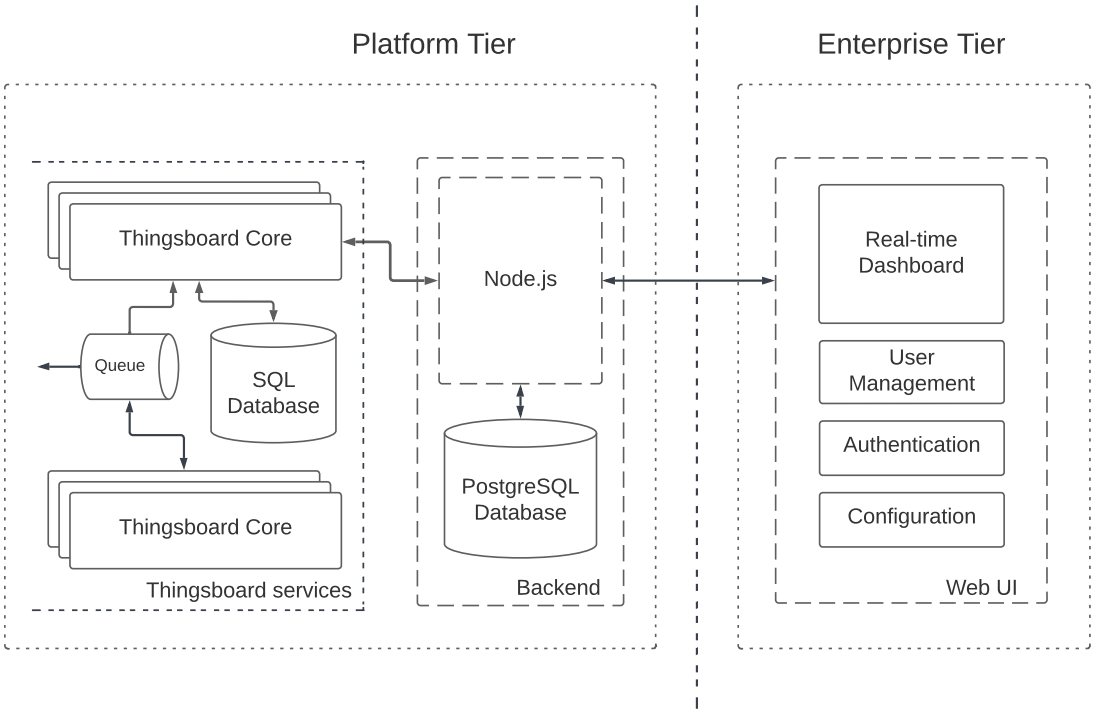

Figure 2.4: *Backend and frontend architecture*

2.1.4 Final architecture

Overall, the final architecture model is the result of a careful consideration of the various components of architecture, and it will be tailored to our specific requirements and goals of the system further down the road.

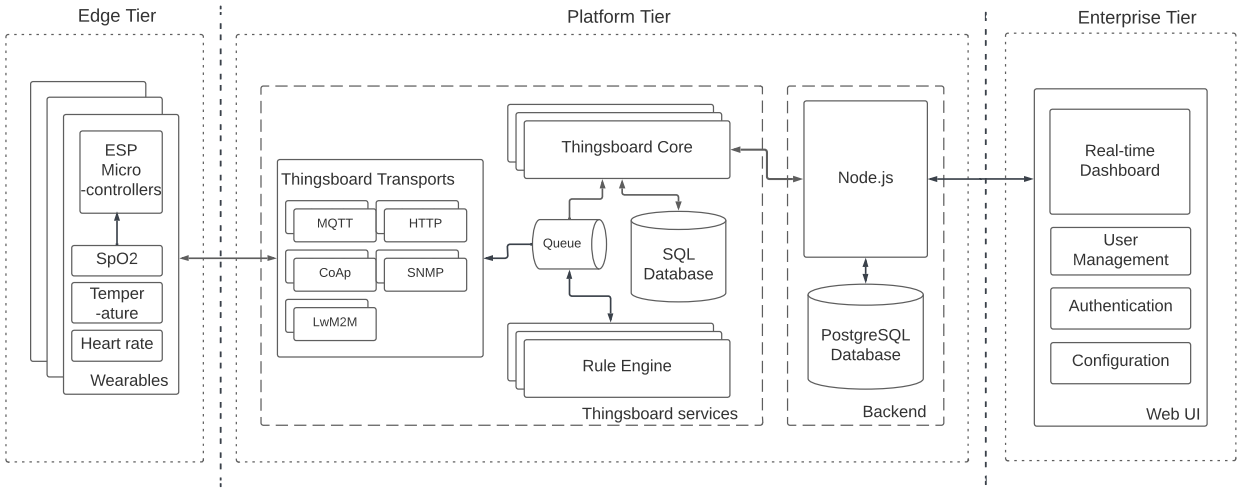

Figure 2.5: *Architecture of the whole system*

2.2 Monolith to Micro-service Architecture

2.2.1 The differences

Monolith and microservices architecture are two different approaches to designing and implementing the architecture of a software application or system. Monolith architecture refers to a traditional, single-tier architecture, where the entire application or system is built as a single, monolithic codebase, with all of the components, modules, and functions of the application or system being tightly coupled and integrated. In contrast, microservices architecture refers to a more modern and modular approach, where the application or system is built as a collection of independent, self-contained services, with each service being responsible for a specific domain or functionality, and with the services being loosely coupled and independently deployable.

One of the key differences between monolith and microservices architecture is the complexity and scalability of the application or system. In a monolithic architecture, the entire application or system is built as a single, complex codebase, which can make it difficult to understand, maintain, and evolve over time. This can lead to significant challenges and limitations in terms of scalability, as the entire application or system must be deployed, scaled, and managed as a single unit, which can be difficult and costly to do.

In contrast, in a microservices architecture, the application or system is built as a collection of independent, modular services, which are designed to be small, simple, and focused on a specific domain or functionality. This makes the architecture of the application or system much more modular, flexible, and scalable, as each service can be deployed, scaled, and managed independently, based on the specific requirements and needs of the application or system. This enables the application or system to grow and evolve over time, without requiring significant refactoring or redesign of the architecture, and without incurring significant costs or challenges in terms of scalability.

Another key difference between monolith and microservices architecture is the level of coordination and communication between the various components and modules of the application or system. In a monolithic architecture, all of the components and modules of the application or system are tightly coupled and integrated, which means that they must coordinate and communicate with each other in order to function properly. This can lead to significant challenges and limitations in terms of flexibility and adaptability, as changes to one component

or module can have significant impacts on other components or modules, and can require significant refactoring and redesign of the architecture.

In contrast, in a microservices architecture, the various services of the application or system are loosely coupled and independently deployable, which means that they can coordinate and communicate with each other in a more flexible and adaptive manner. This enables the services to evolve and adapt over time, without requiring significant refactoring or redesign of the architecture, and without incurring significant impacts on other services. This makes the architecture of the application or system much more flexible and adaptable, and enables it to respond to changing requirements and needs more effectively.

An example of where using a microservices architecture has more advantages than using a monolith one is in the development of a healthcare management system. In this example, a healthcare organization is looking to develop a new system that will be used to manage the various clinical, administrative, and financial aspects of its operations, including patient records, appointments, billing, and other types of data and functionality. In a monolith architecture, the healthcare organization would typically build the entire system as a single, complex codebase, with all of the components, modules, and functions of the system being tightly coupled and integrated. This approach would have several disadvantages and limitations, including the following:

- Complexity: Building the entire system as a single, complex codebase would make the architecture of the system difficult to understand, maintain, and evolve over time. This would require significant effort and resources to develop, test, and maintain the system, and would make it difficult to respond to changing requirements and needs, such as new clinical, administrative, or financial functionality, or new regulatory or compliance requirements.
- Scalability: Building the entire system as a single, monolithic codebase would also make it difficult to scale the system to meet the changing demands and requirements of the healthcare organization, such as an increase in the number of patients, or a new clinical or administrative function. This would require significant effort and resources to deploy, manage, and maintain the system, and would make it difficult to respond to sudden spikes or changes in demand or usage, such as during a pandemic or other emergency situation.
- Flexibility: Building the entire system as a single, monolithic codebase would also make it difficult to adapt the system to the specific requirements and

needs of the healthcare organization, such as the use of different clinical or administrative protocols, or the integration with other systems or applications. This would require significant effort and resources to refactor and redesign the system, and would make it difficult to respond to changing requirements and needs in a timely and efficient manner.

- Interoperability: Building the entire system as a single, monolithic codebase would also make it difficult to integrate the system with other systems or applications, such as electronic health records, insurance systems, or patient portals. This would require significant effort and resources to develop, test, and maintain the integration between the systems, and would make it difficult to exchange data and functionality between the systems in a seamless and secure manner.

In contrast, in a microservices architecture, the healthcare organization would build the system as a collection of independent, self-contained services, with each service being responsible for a specific domain or functionality, and with the services being loosely coupled and independently deployable. This approach would have several advantages and benefits, including the following:

- Simplicity: Building the system as a collection of independent, self-contained services would make the architecture of the system much simpler and more modular, and would make it easier to understand, maintain, and evolve over time. This would require less effort and resources to develop, test, and maintain the system, and would make it easier to respond to changing requirements and needs.
- Scalability: Building the system as a collection of independent, self-contained services would also make it easier to scale the system to meet the changing demands and requirements of the healthcare organization. This would require less effort and resources to deploy, manage, and maintain the system, and would make it easier to respond to sudden spikes or changes in demand or usage, such as during a pandemic or other emergency situation. Each service could be scaled independently, according to its specific requirements and usage patterns, and the services could be deployed on different physical or virtual infrastructure, depending on their performance, availability, and security needs.
- Flexibility: Building the system as a collection of independent, self-contained services would also make it easier to adapt the system to the specific requirements and needs of the healthcare organization. This would require less

effort and resources to refactor and redesign the system, and would make it easier to respond to changing requirements and needs in a timely and efficient manner. Each service could be developed and maintained by a different team or vendor, according to its specific domain or functionality, and the services could be updated, modified, or replaced without affecting the overall functionality or performance of the system.

- Interoperability: Building the system as a collection of independent, self-contained services would also make it easier to integrate the system with other systems or applications. This would require less effort and resources to develop, test, and maintain the integration between the systems, and would make it easier to exchange data and functionality between the systems in a seamless and secure manner. The services could use different communication protocols, data formats, and security mechanisms, depending on their specific requirements and capabilities, and the services could be accessed and used by other systems or applications, through well-defined, standardized, and documented interfaces and APIs.

In summary, using a microservices architecture to develop a healthcare management system would provide several advantages and benefits over using a monolith architecture. The microservices architecture would be simpler, more modular, more scalable, more flexible, and more interoperable than the monolith architecture, and would require less effort and resources to develop, test, maintain, and evolve over time. The microservices architecture would also provide greater resilience and availability, greater performance and scalability, and greater agility and adaptability than the monolith architecture, and would enable the healthcare organization to respond to changing requirements and needs, and to leverage the latest technologies and innovations in the field of healthcare and information technology.

2.2.2 Switching to microservices

Migrating from a monolithic architecture to microservices involves breaking a large, monolithic application into smaller, independent microservices, designing the interfaces between these microservices, and carefully migrating the components of the monolithic application to the corresponding microservices. This process resulted in the final microservices architecture we have below.

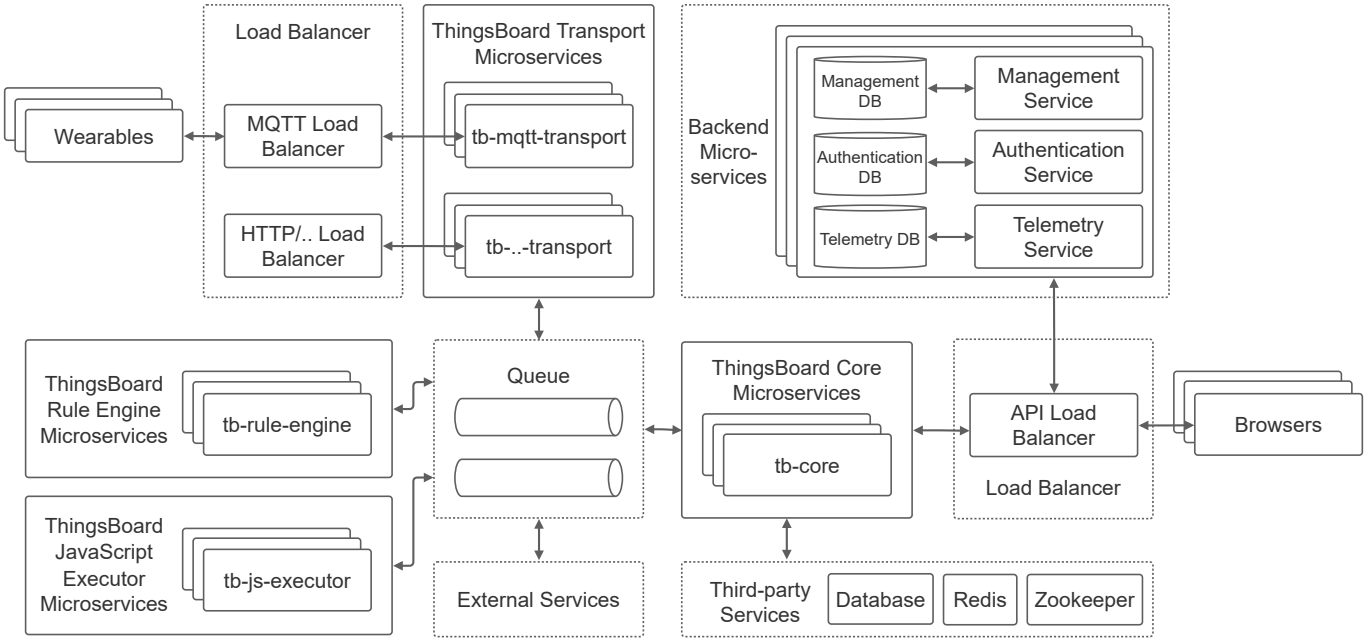

Figure 2.6: *Microservices architecture for the whole system*

Aside from having a monolith model, ThingsBoard is also based on and designed following the microservices pattern, where each service is responsible for a specific domain or functionality. The transport, core, rule engine components now become the transport, core, rule engine microservices, responsible for similar tasks in a monolith architecture. Additionally, there is the new JavaScript Executor Microservices. ThingsBoard rule engine allows users to specify custom javascript functions to parse, filter and transform messages. Since those functions are user defined, we need to execute them in an isolated context to avoid impact on main processing. ThingsBoard provides a lightweight component written using Node.js to execute user defined JavaScript functions remotely to isolate them from the core rule engine components.

Each of the servers of services listed above communicates with the main ThingsBoard Node (with ThingsBoard Core components) microservices using Kafka. Apache Kafka is an open-source stream-processing software platform that is widely used for building real-time data pipelines and streaming applications. It allows us to publish and subscribe to streams of data, process streams of data in real-time, and store streams of data in a fault-tolerant way. In addition to using Kafka for data persistence and processing, ThingsBoard also uses it for some API calls between microservices. This allows the different microservices that make up the ThingsBoard platform to communicate with each other in a efficient and scalable way.

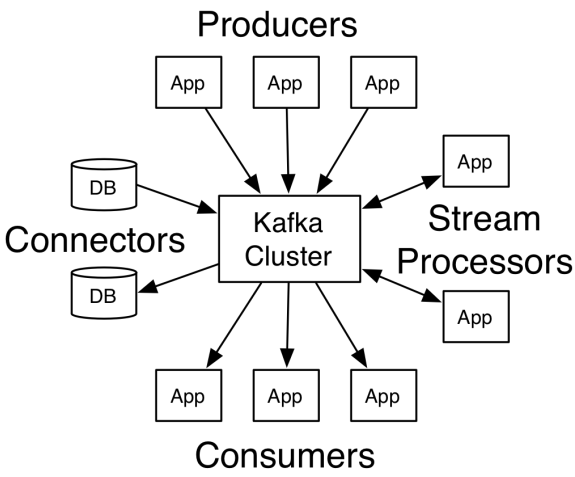

Figure 2.7: *Kafka APIs [12]*

There are four main APIs of Kafka included:

- **Producers API**: allows applications to send streams of data to topics in the Kafka cluster.
- **Consumers API**: allows applications to read streams of data from topics in the Kafka cluster.
- **Connectors API**: allows implementing connectors that continually pull from some source data system into Kafka or push from Kafka into some sink data system.
- **Stream Processors API**: allows transforming streams of data from input topics to output topics.

Kafka cluster consists of brokers which are actually the servers. The topics are partitioned among different nodes of a cluster. Different processes called producers publish the data on a topic. Data is distributed to the different partitions of a topic. The messages are recognized by their offsets in the partition. The data from the topic is read by the processes called consumers for further processing. By using Kafka, ThingsBoard is able to provide developers with a highly performant and reliable platform for building IoT applications.

We have also divided the backend into a microservices-based application that consists of three services: a telemetry service, a management service, and an

authentication service. Each service has its own PostgreSQL database for storing data, and they are all connected to the frontend through an API load balancer.

The telemetry service is responsible for receiving data output from a ThingsBoard server and displaying it on the screen in real-time. This service uses the PostgreSQL database to store time series data, and it exposes a RESTful API that allows the frontend to access the data and open Websocket connections to stream real time data on dashboards. The management service is responsible for managing users and their roles that allows the frontend to create, update, and delete users, as well as manage their roles and permissions stored also in a PostgreSQL database. Finally the authentication service stores user data and authorizes users according to their roles and permissions.

All of these services are connected to the frontend through an API load balancer. An API load balancer is a type of load balancer that routes requests from clients to the appropriate application server based on the URL and the HTTP method of the request. This allows a single load balancer to distribute requests to multiple application servers, providing improved scalability and availability for the application. API load balancers are commonly used in microservices-based architectures, where multiple independent services are used to implement different aspects of the application. By using an API load balancer, clients can access the functionality provided by the different services in a seamless and consistent way, without having to worry about the details of how the services are implemented or where they are running. They typically use a round-robin algorithm to distribute requests evenly across the available application servers. They may also use other algorithms, such as least connections or least response time, to distribute requests in a way that optimizes performance or availability. An API load balancer here allows the frontend to access the functionality provided by the different services in a seamless and consistent way, without having to worry about the details of how the services are implemented.

Now that we have discussed the architecture we will use for this project, we can move on to the design and implementation of the system. In the next chapter, we will describe our design process at every step, as well as the tools and technologies we use, and see how they fit into the overall architecture.

2.3 Others

fhir security

3 Design and Implementation

Software Development Life Cycle (SDLC) is a framework that is used to plan, develop, test, and deploy software applications. The SDLC is typically divided into several distinct phases, each of which serves a specific purpose and produces specific deliverables. These phases may include planning, requirements gathering and analysis, design, implementation, testing, deployment, and maintenance. The specific steps and activities involved in each phase will vary depending on the specific SDLC methodology being used and the needs of the project. We choose to follow SDLC to provide a systematic, disciplined approach to software development that helps ensure the delivery of high-quality, cost-effective software applications that meet the needs of the business or organization. The next section that follows the SDLC is the requirement analysis step which sets the foundation for the entire development process. This step involves gathering and analyzing information about the needs and goals of the project, as well as the stakeholders involved.

3.1 Requirement Analysis

3.1.1 Involved Parties and Users

Let's imagine a scenario where a patient having the dengue fever needs to get taken care of. The entire process of entering a hospital can take anywhere from a few hours to several days, depending on the severity of the patient's condition and the treatments they require. Throughout this process, the medical team will work to provide the patient with the best possible care and help them recover from their illness or injury. When a patient enters a hospital, the first person they typically encounter is a triage nurse. This nurse is responsible for assessing the patient's condition and determining the level of medical attention they need. The

triage nurse will ask the patient about their symptoms and medical history, and may also perform a physical examination. Based on this information, the triage nurse will decide whether the patient needs immediate medical attention or can wait to be seen by a doctor.

If the patient needs immediate attention, they will be taken to an exam room where they will be seen by a doctor or other medical professional. In the exam room, the patient will receive the necessary medical care to address their condition, which could include treatments such as medication, IV fluids, or other interventions. On the other hand, if the patient does not need immediate attention, they will be asked to wait in the hospital's waiting area until a doctor is available to see them. While they are waiting, they may be asked to fill out paperwork or undergo further assessments by the triage nurse.

Once the patient is in an exam room, they will meet a fill-in nurse who will help them fill out any necessary paperwork and collect information about their medical history and current condition. This information is important because it will be used by the on-call specialist who will be treating the patient. The fill-in nurse may also perform additional assessments, such as taking the patient's vital signs or checking for any other medical issues. They may also provide the patient with information about their condition and answer any questions they may have. The exam room step may be repeated multiple times, depending on the patient's condition and the treatment they require. After each exam, the patient will be reassessed by the triage nurse to determine if they need further medical attention or can be discharged from the hospital.

If we could call the above scenario the assessment phase, the next phase would be called the treatment phase, if the patient's condition is not severe, they may be prescribed medication or other treatment by the on-call specialist and sent home. However, if the patient's condition is severe, they may need to be admitted to the hospital for an extended period of time in a care unit where they will be under the supervision of a doctor and a team of nurses. The care unit is a specialized area of the hospital where patients with serious or chronic conditions can receive the intensive medical care they need. The supervising doctor will develop a treatment plan that is tailored to their specific needs that may include a combination of medications, therapies, and other interventions. The nursing staff will be responsible for carrying out the doctor's orders and providing the patient with the necessary care. Once the patient's condition has improved to the point where they are no longer in need of intensive medical care, they may be discharged from the hospital and sent home. However, they may still need to follow up with

their doctor or other medical professionals to ensure that their condition continues to improve and to monitor for any potential complications.

In the scenario described above, there were several actors involved. The patient was the primary actor in this scenario, as they were the individual seeking medical treatment. The doctor and the nurses also played important roles in providing the patient with the necessary medical care and ensuring that their condition improves. Suppose we were a company that provided an IoT-based application for hospitals to monitor and manage patients under the dengue fever, these actors - doctor, nurse and patient would be our three primary users under the hospital as the customer. As for the devices, hospitals could choose to purchase the wearable devices directly from our company, or they could use devices from a third-party provider.

Table 3.1: *Stakeholders and their roles*

Stakeholder	Role
Doctors	Responsible for assessing, monitoring and treating patients.
Nurses	Monitor patients, coordinate the consultations with doctors and administer treatments.
Patients	Use the system for self-diagnosing and treatment

Aside from the involved stakeholders above, it's assumed that the admin staff would be responsible for managing and maintaining the app in the shadow, including ensuring that it is up to date and functioning properly by installing updates, and troubleshooting any technical issues, and providing support to users of the app.

3.1.2 Functional Requirements

The functional requirements should outline several key features and capabilities that the app should have. These requirements are as follows:

Application aspect

- The app should allow users (doctors) to manually control the data collection process, or set it to run automatically based on their preferences. This could include tasks such as setting the frequency of data collection or specifying the types of data to be collected.

- The app should be able to monitor the user (patient) remotely by collecting data from a wearable device. The app should simultaneously run these processes for large amount of users.
- The app should include a patient management system that allows users to effectively manage and prioritize their patients based on their individual conditions. This could include tasks such as organizing patients by condition, setting reminders for follow-up care, and tracking the progress of each patient over time.
- The app should be able to identify patients with critical conditions and alert doctors to their status. This feature should help doctors prioritize and respond to patients who are in urgent need of care.

Internet-of-Things aspect

- Wearable devcice should automatically collect temperature, blood pressure, heart rate and SpO_2 data from IoT sensors on patients at regular intervals.
- The system should be able to report issues that require manual intervention in the event that a measuring device fails to function properly.
- The system should provide the user with the option to set the time between data collection events or to manually collect data at any time.
- Provide live reports to the user via some form of display (graph view and log view) and share with others.
- The system should be able to identify patients with critical conditions and alert doctors to their status.

3.1.3 Non-functional Requirements

Application aspect

- The app should be designed to be available and usable for people of all ages, with a particular focus on children and elderly users. To ensure that it is accessible to users of all ages and abilities, the user interface (UI) should be legible and easy to understand. Additionally, for mobile screens, the UI should resize accordingly.

- In terms of performance, the system should be able to establish a connection with the server within 2 seconds, and basic interface interactions should take no more than 500 milliseconds. This will ensure that the system is responsive and easy to use, even in situations where speed is of the essence.
- Finally, the system should provide secure authentication options for users. This should include the ability to authenticate using a securely-stored password and/or patterns. This will help to protect the system and the data it contains from unauthorized access.
- The website should be accessed and used by a wide range of users, regardless of their preferred browser or platform. By ensuring compatibility with all major browsers and platforms, the website will be accessible to the widest possible audience.

Internet-of-Things aspect

- The system should be designed to run 24/7 with minimal scheduled downtime, and ideally no unscheduled downtime. This will ensure that the system is always available to users and can support mission-critical operations.
- The system should be designed to handle devices using one or more MQTT servers. This will provide a scalable and efficient means of communicating with and controlling devices.
- The system should be able to persistently log historical readings for diagnostic purposes. This will allow the system to keep track of its status over time and provide diagnostic information when needed. The system should be able to maintain log records for at least 180 days.
- The system should use a scalable and efficient database system with a capacity of at least 500MB. This will ensure that the system can store and manage large amounts of data without performance degradation.

3.1.4 Use-case view

Here we used a use-case diagram to identify and document the functional requirements of a system, and to show how the system will be used in practice. A use-case diagram typically consists of a series of connected shapes, with each shape representing a different actor or component of the system. The relationships

between these shapes illustrate the ways in which the actors and components will interact with each other to achieve the system's goals.

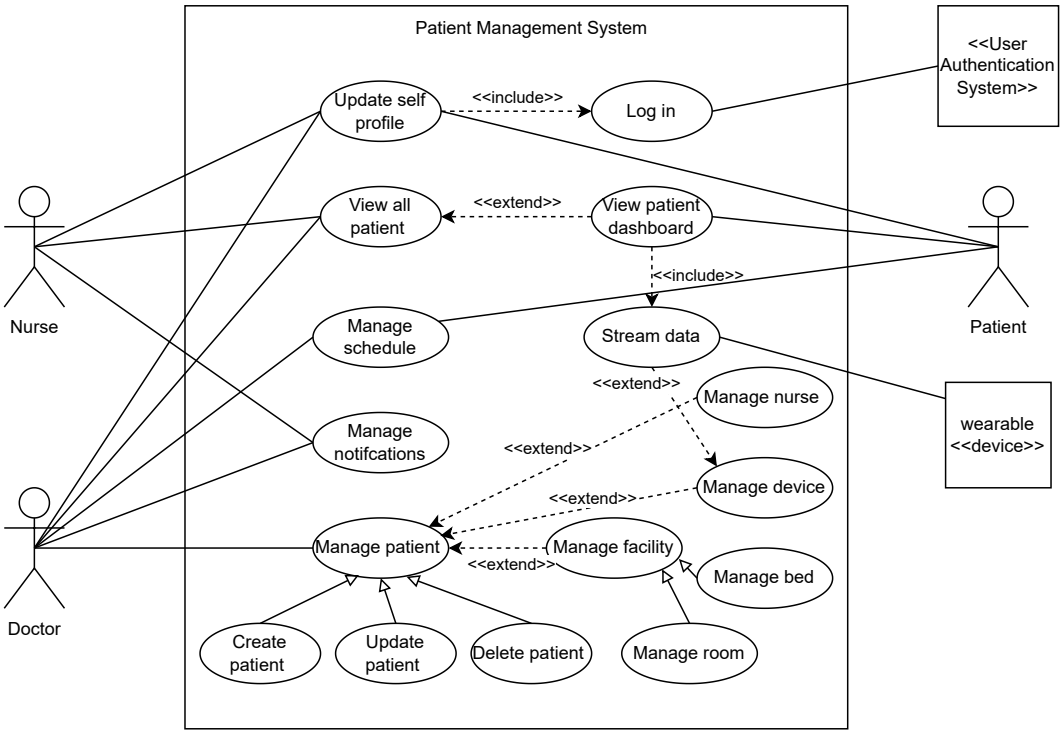

Figure 3.1: *Use-case diagram of the system*

As mentioned before, in the case of our healthcare system, the main actors will be nurse, doctor, and patient. In a use-case diagram, each actor is represented by a rounded rectangle, and each use-case is represented by an oval shape. The relationships between the actors and use-cases are shown using lines or arrows. In this example, the nurse, doctor, and patient would be represented by rounded rectangles, and the use-cases would be represented by ovals. The primary use-cases for such a system might include managing self profiles, viewing all patients, managing schedules, managing notifications, managing patients, and stream data.

For example, the "update self profile" use-case is a subset of the broader "manage profile" use-case that applies to all actors in the system. Each actor has their own individual profile that is created when they first log in to the system. The "log-in" use-case has a relationship with the "update self profile" use-case, as well as with almost every other use-case in the system. This is because only authorized users are able to read or write data, so the "log-in" use-case is a prerequisite for most other use-cases. This design choice was made in order to avoid cluttered and hard-to-read diagrams.

The "view all patients" use-case is only linked to the doctor and nurse actors, as they are the only ones who have the ability to view information about all patients in the system. Patients, on the other hand, have access to real-time information about their own health and vital signs, as monitored by wearable devices that are assigned to them by their doctor. In addition, doctors and nurses are able to communicate with each other through the notification system, and doctors can schedule appointments with patients. These interactions are all represented in the use-case diagram by the relationships between the actors and use-cases.

The "manage user" use-case is only linked to doctors, as they have the highest level of authority in the system. This use-case is a combination of several other use-cases, including "manage patient", "manage nurse", "manage facility", and "manage device". Each of these use-cases has three actions: create, update, and delete. The "manage facility" use-case has two additional sub-use-cases, "manage room" and "manage bed", which are used to manage the primary assets in hospitals. This hierarchy of use-cases and their relationships is also represented in the use-case diagram.

3.1.5 Process view

A sequence diagram is a type of diagram that shows the interactions between objects or components in a system, over time. It is used to illustrate the order in which these interactions take place, and to show how the different objects or components interact with each other to accomplish a specific goal. In the case of user registration process, a sequence diagram can help us to understand the steps involved and the interactions between the different actors and sub-use-cases in the system.

For example, the user registration sequence diagram might show the interactions between the user (in this case the doctor), the frontend, backend, and the database. It would show the steps involved in creating a user account, such as entering personal information, choosing a username and password, storing the credentials and activate the account by email.

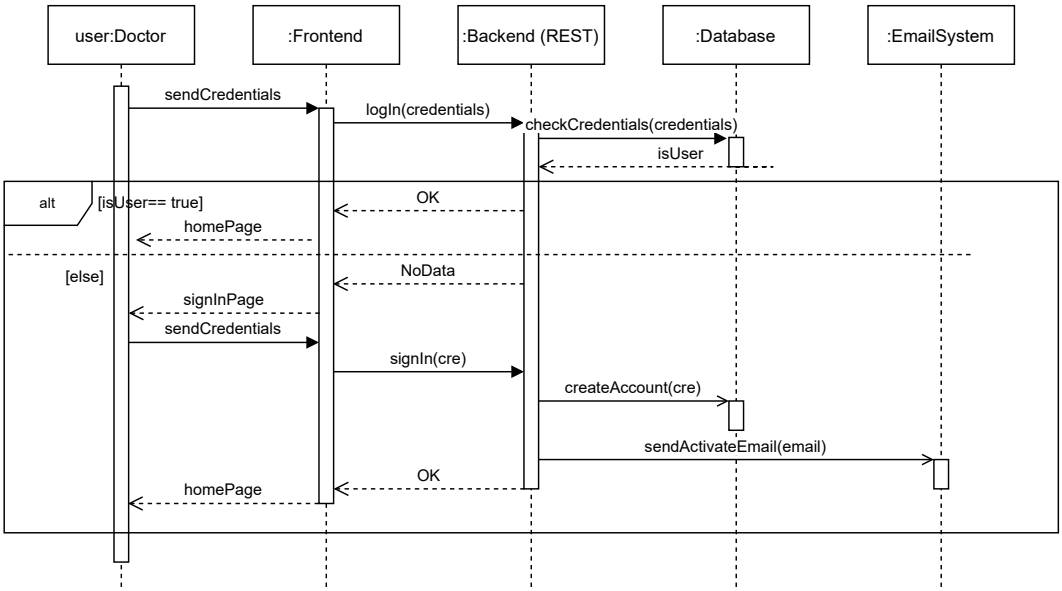

Figure 3.2: *Sequence diagram of user registration*

Further descriptions

Use-case name	User Registration
Actors	User, server, and database
Description	Interactions between the user and the system when it comes to account authentication and authorization.
Preconditions	The system is running normally. The website is connected to the database and user may or may not already have an account.
Normal flow	The expected course of actions would be for the user to log in. 1. User opens/logs in the website. 2. System sent a log in page to user. User fills in their credentials such as username, password. 3. System sends the credentials to the backend over REST API. The backend verifies this information with the authentication database that stores all users' credentials. 4. User's credentials are valid, the frontend sends back the home page.

Exceptions	*At step 2, if user inputs invalid information, such as wrong email syntax:* 1. System informs user to revalidate their inputs".
Alternative flow	*At step 4, if no user exists in the database:* 1. System informs user that they need to create an account and redirects user to the sign up page. 2. User sends their information, including new passwords. 3. System validates that the email is unique and stores user's information in the database. System sends an activation email to user and alert user to activate their account. 4. User activates their account by clicking the link sent in their mail. 5. System redirects user to the home page.

After creating a sequence diagram for the user registration process, we can use this diagram as a reference when creating another sequence diagram that illustrates the process of data streaming. This will allow us to see how the user registration process fits into the broader context of data streaming, and how the different actors and components interact with each other to accomplish this goal.

The diagram would show how the user (doctor), the frontend, backend, the database, and thingsboard server come to play. It would also show how the user's wearable devices connect to the data streaming system and transmit data over MQTT, and how the system processes this data and provides it to the user in real-time.

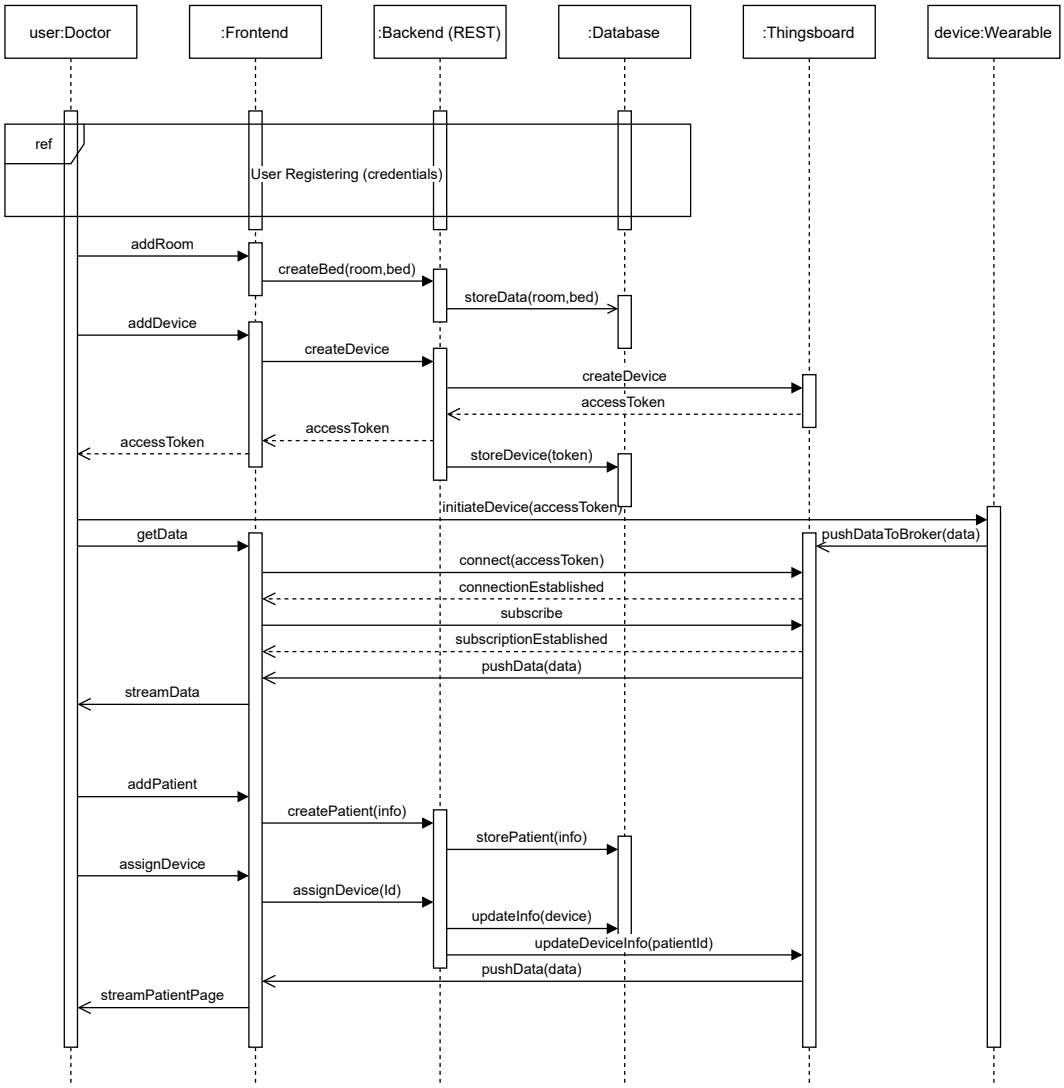

Figure 3.3: *Sequence diagram of real-time data streaming*

Further descriptions

Use-case name	Data Streaming
Actors	User, server, database, ThingsBoard and devices
Description	Interactions between the user and the system when it comes to assigning a device to registered patients and enabling real time streaming.
Preconditions	The system is running normally. The website is connected to the database. ThingsBoard server is ready as well as devices .

Normal flow	The expected course of actions would be for the user, assuming a doctor, to enable data streaming. 1. User either logs in or signs up in the log in page (User Registering step). 2. User adds facilities (room and bed). Required information are preferably the room and bed's IDs. System stores new information in the database, linked to the ID of the user. 3. User adds device. The server send a create request to ThingsBoard under the new device topic. If no access token is provided, an access token will be automatically generated and sent back to the server to be stored alongside the device's details in the database. 4. User configures device to stream data to ThingsBoard. 5. User sends request to server to open Websocket connection with ThingsBoard. After subscription, ThingsBoard now can stream data directly to the frontend. Telemetry data can be stored in a separate database if configured so. 6. User adds patient by filling in their details, especially their email address so that server can send the activation link from the Email System. 7. User assigns the just created to the patient. Now user and the patient can view device's real time telemetry data on the dashboard.
Optional flow	*If user wishes to add assisting medical staff or nurses, the steps are similar to patient registration except that instead of assigning a device to patient, user needs to assign a room to the nurse to monitor. All assigned nurses can view the beds, and their according patients, available in the specific room assigned to them*

3.2 Database Design

A well-designed database is critical for the performance and reliability of a system. A poorly-designed database can result in slow performance, data

inconsistencies, and other issues that can have a negative impact on the system's usability and effectiveness. In order to design an effective database, it is important to carefully consider the requirements and goals of the system, and to choose an appropriate database model and design patterns. The database design should be based on a thorough understanding of the data that will be stored, the relationships between different data elements, and the ways in which the data will be accessed and manipulated.

To provide a broad overview of the database and help to ensure that all relevant aspects of the system have been considered, we use a conceptual design to identify the entities, attributes, and relationships that are relevant to the system, and so forth define the high-level structure of the database.

3.2.1 Conceptual design

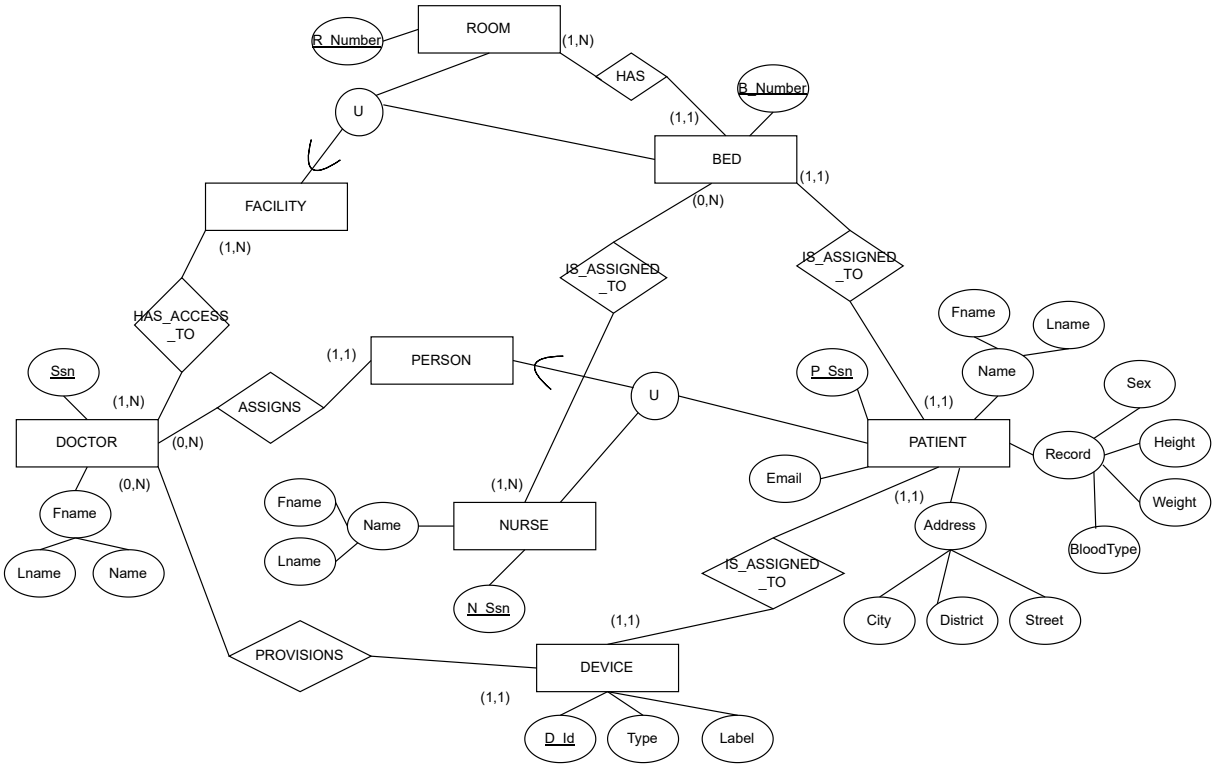

Figure 3.4: *Enhanced entity relationship diagram of the database*

After completing the conceptual design of a database, the next step for us is to move on to the logical design phase. In this phase, the focus is on mapping the entities, attributes, and relationships identified in the conceptual design to

a specific database model, such as the relational model. This process typically involves creating a detailed map of the database, including tables, columns, and relationships, and defining the constraints and rules that govern the data. We accomplished this by creating a set of tables that represent the entities in the system, and defining the columns in each table to represent the attributes of those entities. The relationships between the entities can then be defined using foreign keys, which link the tables together. This process can help to ensure that the database is able to accurately and efficiently store and retrieve data, and that it is able to maintain the relationships between different data elements.

3.2.2 Logical design

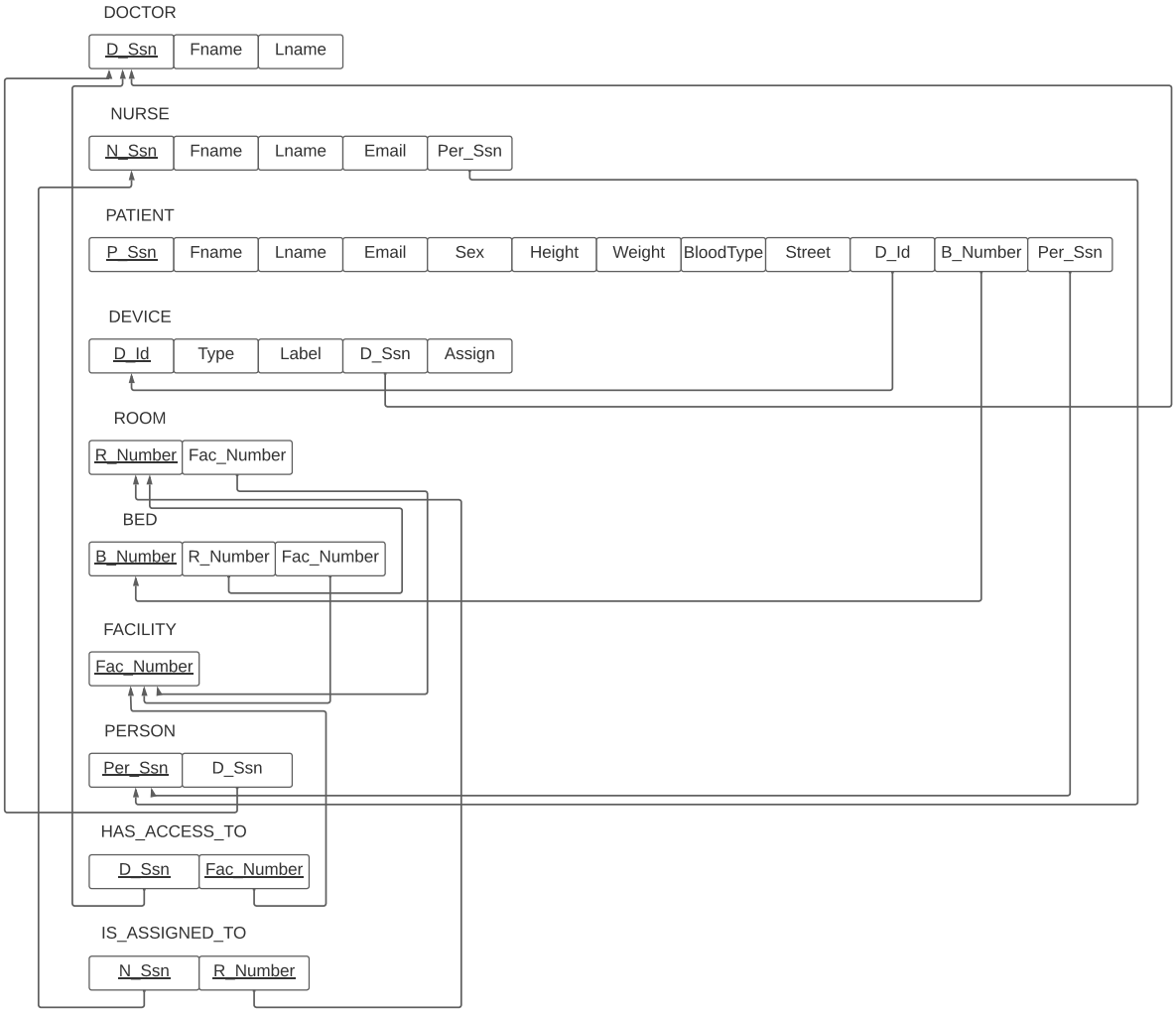

Figure 3.5: *Logical design of the database*

For the physical design and in actual implementation, we decided to use PostgreSQL - a powerful and popular open-source relational database management system (RDBMS) in our project. It is well-suited for handling patient management and real-time telemetry data from IoT devices because of its strong support for SQL to manage and analyze complex data sets, such as those that are common in healthcare and IoT applications. PostgreSQL is also highly scalable, meaning that it can easily handle large volumes of data without sacrificing performance. It also offers robust support for transactions, which is essential for maintaining the integrity of patient records and other sensitive data. Finally, PostgreSQL is highly extensible, which means that it can be customized to meet the unique needs of a healthcare or IoT application.

In contrast, NoSQL database management systems and MySQL are not as well-suited for managing patient management and real-time telemetry data. NoSQL databases are designed for handling large volumes of unstructured data, not really the structured and complex data sets that are common in healthcare and IoT applications. MySQL, on the other hand, is a popular relational database management system, but it is not as feature-rich or robust as PostgreSQL.

Once the database design for an IoT system has been completed, the next step is to begin the implementation phase. The first step in this phase is to implement the hardware and IoT devices that will be used to collect and store data. This involves choosing the hardware components that will be used to build the IoT system, such as sensors, microcontrollers, and networking equipment, and configuring those components to work together. In addition, it may also involve deploying software and firmware updates to the IoT devices in order to enable them to communicate with the database and other components of the system.

After the hardware and IoT device implementation step is complete, the next steps in the implementation phase are backend implementation and frontend implementation. The backend implementation step involves building the server-side components of the system, such as the database server and any APIs or other services that will be used to access the data. The frontend implementation step involves building the user-facing components of the system, in this case the website that will be used to view and interact with the data.

3.3 Hardware and Device Implementation

3.3.1 Micro-controller (ESP8266 Wemos D1 Mini)

The ESP8266 Wemos D1 Mini is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif Systems [13]. ESP8266 Wemos D1 Mini is a microcontroller with powerful processing speed on board. The storage space of this module is also high allowing it to integrate with other devices like sensors. Besides, ESP8266 Wemos D1 Mini module integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow the processing power to be available for user application programming and development. This module is cost-effective and thus used widely in many applications like the Internet of things.

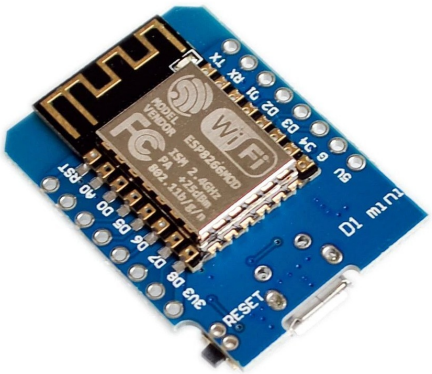

Figure 3.6: *ESP8266 Wemos D1 Mini*

Specifications:

- **Model:** Node MCU Lua D1 Mini
- **Main IC:** ESP8266 WiFi SoC version ESP12F
- **Firmware version:** NodeMCU
- **Chip and UART communication:** CH340
- **Power supply:** 5VDC MicroUSB or 5V, 3V3 pin
- **GPIO voltage communication:** 3.3VDC

- **WiFi:** 2.4GHz

ESP8266 Wemos D1 Mini Architecture

The functional diagram of ESP8266 Wemos D1 Mini is shown in Figure below:

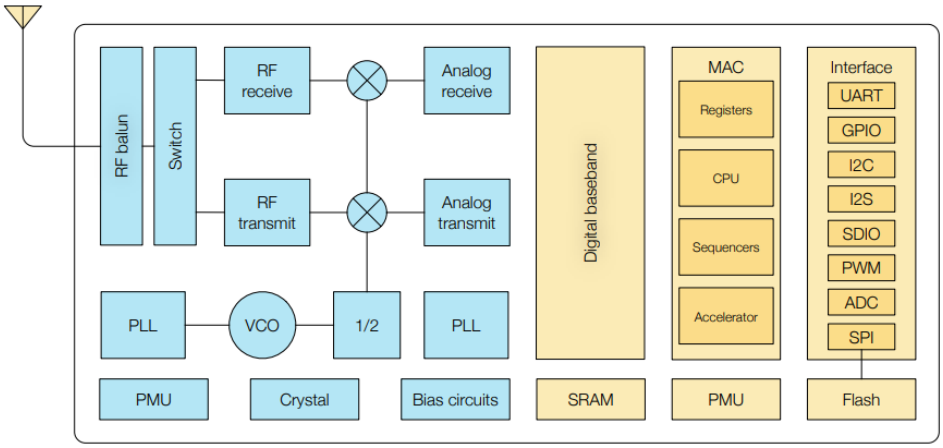

Figure 3.7: *Functional Block Diagram [14]*

- **CPU**: The ESP8266 Wemos D1 Mini integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow 80% of the processing power to be available for user application programming and development.

- **Memory**: ESP8266 Wemos D1 Mini Wi-Fi SoC integrates a memory controller and memory units including SRAM and ROM. MCU can access the memory units through iBus, dBus, and AHB interfaces. All memory units can be accessed upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

- **External Flash**: ESP8266 Wemos D1 Mini uses external SPI flash to store user programs, and supports up to 16 MB memory capacity theoretically.

- **Clock**: The high-frequency clock on ESP8266 Wemos D1 Mini is used to drive both transmit and receive mixers. This clock is generated from an internal crystal oscillator and an external crystal. The crystal frequency

ranges from 24 MHz to 52 MHz. On the other hand, an externally generated clock is available with a frequency ranging from 24 MHz to 52 MHz.

- **Wi-Fi**: ESP8266 Wemos D1 Mini implements TCP/IP and full 802.11 b/g/n WLAN MAC protocol. It supports Basic Service Set (BSS) STA and SoftAP operations under the Distributed Control Function (DCF). Power management is handled with minimum host interaction to minimize active-duty period.

3.3.2 Temperature sensor (MLX90614)

The MLX90614 is an infrared thermometer for non-contact temperature measurements. Both the IR-sensitive thermopile detector chip and the signal conditioning ASIC are integrated into the same TO-39 can. The MLX90614 is factory calibrated in wide temperature ranges: $-40\,°\mathrm{C}$... $125\,°\mathrm{C}$ for the ambient temperature and $-70\,°\mathrm{C}$... $380\,°\mathrm{C}$ for the object temperature. The measured value is the average temperature of all objects in the Field Of View of the sensor. The MLX90614 offers a standard accuracy of $\pm 0.5\,°\mathrm{C}$ around room temperatures.

Figure 3.8: *MLX90614 temperature sensor [15]*

Specifications:

- Power supply: 3∼5VDC
- Accuracy: $\pm 0.5\,°\mathrm{C}$
- Sensor temperature range: $-40\,°\mathrm{C}$... $125\,°\mathrm{C}$

- Communication protocol: I2C
- Size: 11.5 x 16.5mm

Pin definitions and descriptions

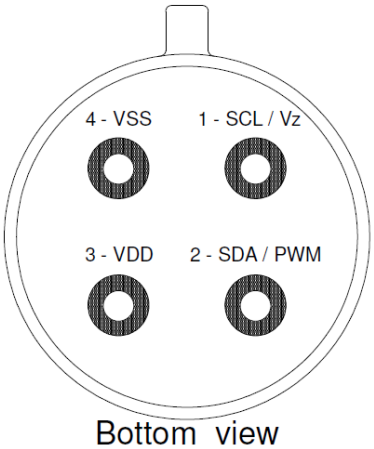

Figure 3.9: *Pin descriptions [15]*

Table 3.4: *Pin description MLX90614 [15]*

Pin Name	Function
SCL / Vz	Serial clock input for 2 wire communications protocol. 5.7V zener is available at this pin for connection of external bipolar transistor to MLX90614Axx to supply the device from an external 8. . . 16V source.
SDA / PWM	Digital input / output. In normal mode the measured object temperature is available at this pin Pulse Width Modulated. In SMBus compatible mode the pin is automatically configured as open drain NMOS.
VDD	External supply voltage.
VSS	Ground. The metal can is also connected to this pin.

MLX90614 Functional Diagram

The MLX90614 is built from 2 chips developed and manufactured by Melexis:

- The infra-red thermopile detector MLX81101.

- The signal conditioning ASSP MLX90302, specially designed to process the output of IR sensor.

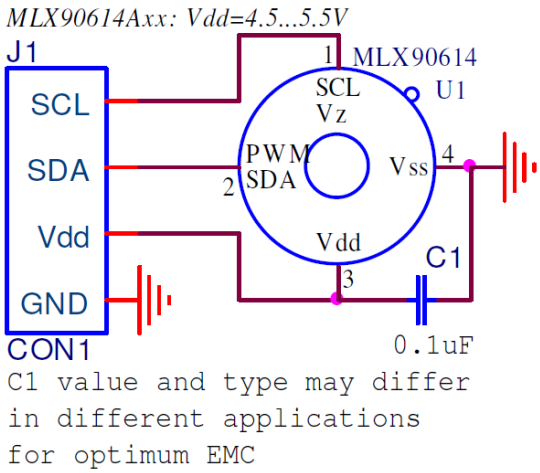

Figure 3.10: *Typical application schematics [15]*

3.3.3 Heart rate sensor (MAX30100)

The MAX30100 is an integrated pulse oximetry and heart rate monitor sensor solution. It combines two LEDs, a photodetector, optimized optics, and low-noise analog signal processing to detect pulse oximetry and heart-rate signals. The MAX30100 operates from 1.8V and 3.3V power supplies and can be powered down through software with negligible standby current, permitting the power supply to remain connected at all times.

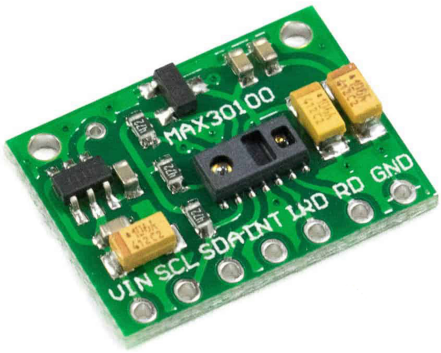

Figure 3.11: *MAX30100 heart rate sensor [16]*

Specifications:

- Power supply: 1.8∼5.5VDC
- Main IC: MAX30100
- Communication protocol: I2C
- Size: 1.9 cm x 1.4 cm x 0.3 cm

Pin definitions and descriptions

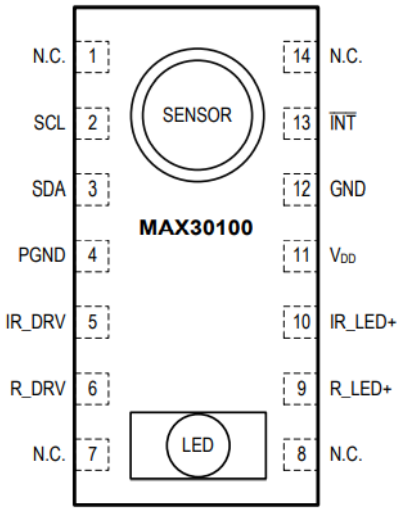

Figure 3.12: *Pin configurations [16]*

Table 3.5: *Pin description MAX30100 [16]*

PIN	NAME	FUNCTION
1, 7, 8, 14	N.C	No Connection. Connect to PCB Pad for Mechanical Stability.
2	SCL	I2C Clock Input
3	SDA	I2C Clock Data, Bidirectional (Open-Drain)
4	PGND	Power Ground of the LED Driver Blocks
5	IR_DRV	IR LED Cathode and LED Driver Connection Point. Leave floating in circuit.
6	R_DRV	Red LED Cathode and LED Driver Connection Point. Leave floating in circuit
9	R_LED+	Power Supply (Anode Connection) for Red LED. Bypass to PGND for best performance. Connected to IR_LED+ internally.
10	IR_LED+	Power Supply (Anode Connection) for IR LED. Bypass to PGND for best performance. Connected to R_LED+ internally
11	VDD	Analog Power Supply Input. Bypass to GND for best performance
12	GND	Analog Ground
13	INT	Active-Low Interrupt (Open-Drain)

MAX30100 Functional Diagram

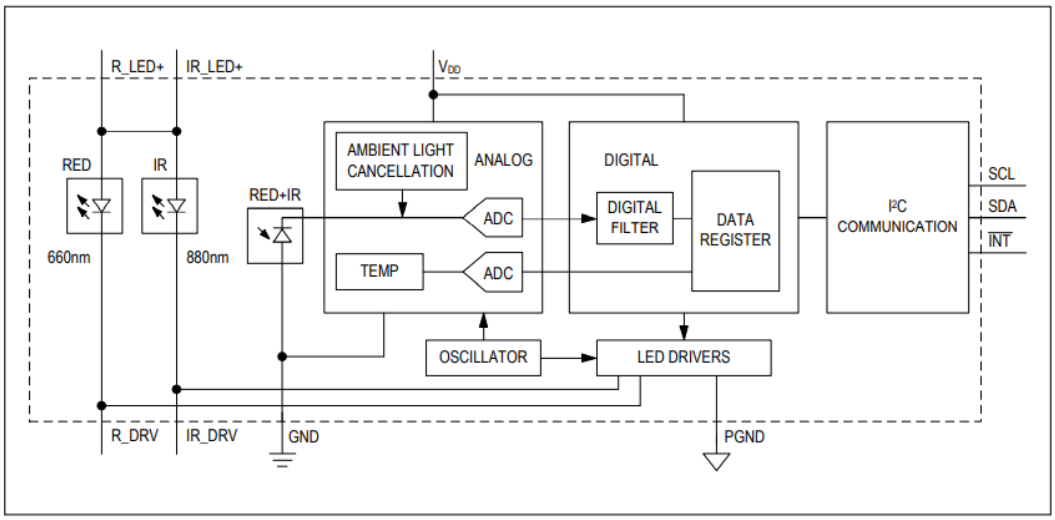

Figure 3.13: *Functional diagram [16]*

- **SpO2 Subsystem:**
  The SpO2 subsystem in the MAX30100 is composed of ambient light cancellation (ALC), 16-bit sigma delta ADC, and proprietary discrete time filter. The SpO2 ADC is a continuous time oversampling sigma delta converter with up to 16-bit resolution. The ADC output data rate can be programmed from 50Hz to 1kHz. The MAX30100 includes a proprietary discrete time filter to reject 50Hz/60Hz interference and low-frequency residual ambient noise.

- **Temperature Sensor:**
  The MAX30100 has an on-chip temperature sensor for (optionally) calibrating the temperature dependence of the SpO2 subsystem. The SpO2 algorithm is relatively insensitive to the wave length of the IR LED, but the red LED's wavelength is critical to correct interpretation of the data. The temperature sensor data can be used to compensate the SpO2 error with ambient temperature changes

- **LED Driver:**
  The MAX30100 integrates red and IR LED drivers to drive LED pulses for SpO2 and HR measurements. The LED current can be programmed from 0mA to 50mA (typical only) with proper supply voltage. The LED pulse width can be programmed from 200µs to 1.6ms to optimize measurement accuracy and power consumption based on use-cases.

3.3.4 OLED screen (SSD1306)

Figure 3.14: *oled SSD1306 [17]*

Specifications:

- Power supply: 2.2∼5.5VDC
- Power consumption: 0.04W
- Display angle: Greater than 160 degrees
- Number of points displayed: 128x64 points
- Display color: White / Blue
- Driver: SSD1306
- Communication protocol: I2C
- Size: Screen width 0.96 inches

SSD1306 is a single-chip CMOS OLED/PLED driver with controller for organic / polymer light emitting diode dot-matrix graphic display system. It consists of 128 segments and 64commons. This IC is designed for Common Cathode type OLED panel.

Block diagram of OLED

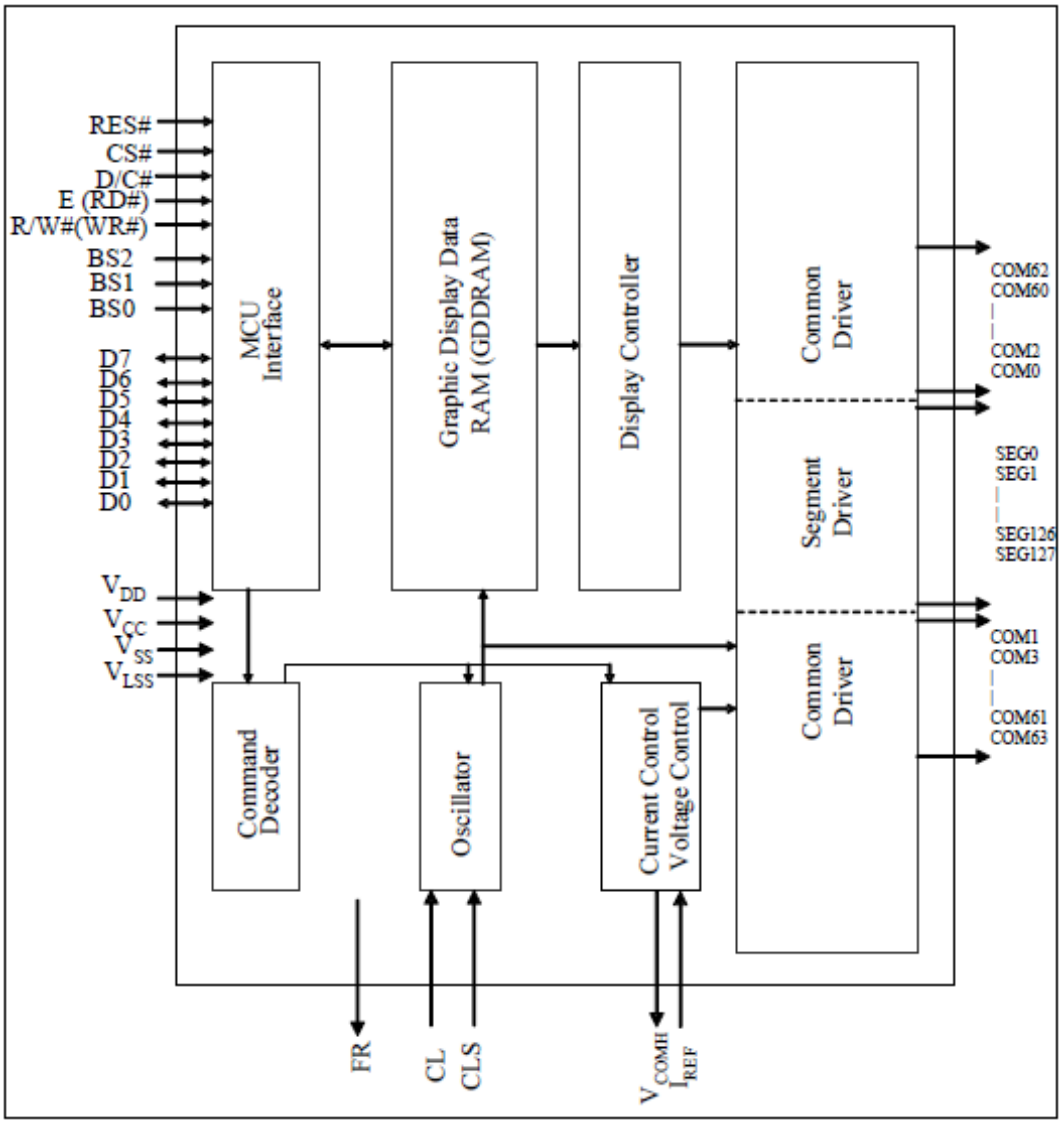

Figure 3.15: *SSD1306 Block Diagram [17]*

- **MCU Interface selection:**
  SSD1306 MCU interface consist of 8 data pins and 5 control pins. Different MCU mode can be set by hardware selection on BS[2:0] pins.

- **Command Decoder:**
  This module determines the input data is interpreted as data or command. Data is interpreted based upon the input of the D/C# pin.

- **Oscillator Circuit and Display Time Generator:**

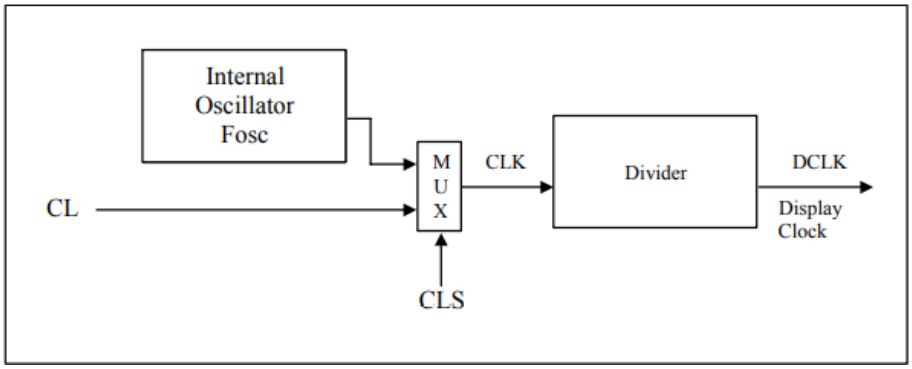

Figure 3.16: *Oscillator Circuit and Display Time Generator [17]*

This module is an on-chip LOW power RC oscillator circuitry. The operation clock (CLK) can be generated either from internal oscillator or external source CL pin. When the internal oscillator is selected, its output frequency Fosc can be changed by command D5h A[7:4].

- **Segment Drivers / Common Drivers:**
  Segment drivers deliver 128 current sources to drive the OLED panel. The driving current can be adjusted from 0 to 100uA with 256 steps. Common drivers generate voltage-scanning pulses.

- **Graphic Display Data RAM (GDDRAM):**
  The GDDRAM is a bit mapped static RAM holding the bit pattern to be displayed. The size of the RAM is 128 x 64 bits and the RAM is divided into eight pages from PAGE0 to PAGE7, which are used for monochrome 128x64 dot matrix display.

3.3.5 Pin Lipo

Figure 3.17: *Pin Lipo*

Specifications:

- Pin type: Lithium Polimer
- Number of cell: 1 cell
- Average voltage: 3.7∼5VDC
- Capacity: 1000mAh
- Size: 53 x 33 x 5.5mm

One of the notes when using lipo batteries is the charging process. Due to the strict requirements on voltage when charging batteries as well as ensuring charging quality. Currently, TP4056 battery charging module is being widely used to charge 1-cell Lipo batteries. Features integrated on the TP4056 IC make charging more convenient.

3.3.6 Communication standard between ESP and modules

In order to ESP8266 communicate with other modules, we must choose right communication protocol to understanding how system work and well-programming. In this project, we mention 2 communication standards.

- **I2C communication protocol**

  I2C stands for the Inter-Integrated Circuit protocol. I2C was invented in 1982 by Philips Semiconductor (now NXP Semiconductor) as a low speed communication protocol for connecting microprocessor master devices with lower-speed peripheral slave devices[18].

  I2C is a widely used protocol for many reasons. It requires only two lines for communications. Like other serial communication protocols, there is a serial data line and a serial clock line. I2C can connect to multiple devices on the bus with only the two lines. The master device can communicate with any slave device through a I2C address sent through the serial data line.

  There are only two lines used for I2C communications. The first line is SCL, which is a serial clock primarily controlled by the master. SCL is used to synchronously clock data in and out of the slave device. The second line is SDA, which is the serial data line. SDA is used to transmit data between the master devices and slave devices. The master device controls the serial

clock SCL, the SDA is used to send data in both directions. The SDA is bi-directional, which means that the master devices and slave devices can both send data on the line. An I2C master device starts and stops communication, which removes the potential problem of bus contention. Also, communication with a slave device is sent through a unique address on the bus. This allows for both multiple master and multiple slave devices on the I2C bus.

I2C communication is initiated from the master device with an I2C start condition. If the bus is open, an I2C master may claim the bus for communication by sending an I2C START condition. Firstly, the master device pulls the SDA low and then pulls the SCL low. This sequence indicates that the master device is claiming the I2C bus for communication, forcing other master devices on the bus to hold their communication. When the master device has completed it's communication, it releases the SCL high and then releases the SDA high. This indicates an I2C STOP condition. This releases the bus to allow other masters to communicate or to allow for the same master to communicate with another device.

- **SPI communication protocol**

  The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The interface was developed by Motorola in the mid-1980s and has become a standard. Typical applications include Secure Digital cards and liquid crystal displays[19].

  SPI devices communicate in full duplex mode using a master-slave architecture usually with a single master. The master (controller) device originates the frame for reading and writing. Multiple slave-devices may be supported through selection with individual chip select (CS). Sometimes SPI is called a four-wire serial bus, because it has 4 point-to-point unidirectional wires in this communication standard.

  - **SCK (Serial Clock):** Generated by the Master. Synchronizes data transmission between the controller and peripheral SCK originates from controller and shared with all slaves.
  - **MOSI (Master Out Slave In):** Output from the Controller used to send data to the peripheral deviceCan be shared between peripheral devices.
  - **MISO (Master In Slave Out):** Output from the peripheral device used to send data to the controller. It can be shared between all

peripheral devices. Peripheral output becomes high impedance when SS is not selected.

– **SS (Slave Select):** Selects the peripheral device for communication. It is often used as active low, which is often represented by an overbar.

3.3.7 Pin connection between micro-controller and sensors

Pin connection between ESP and temperature sensor

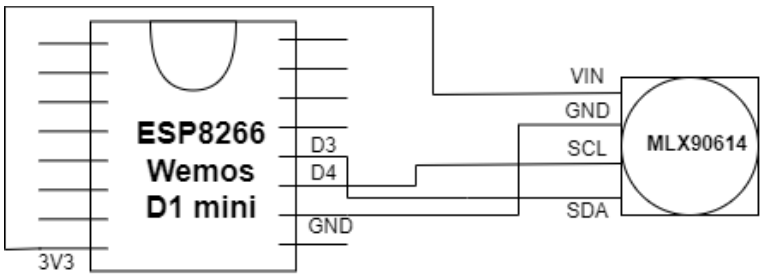

Figure 3.18: *Pin connection between ESP and temperature sensor*

We provide the power supply for temperature sensor first. 3V3 pin and ground (GND) of ESP8266 would connect to VIN and ground (GND) of temperature sensor respectively. Next, We choose D3 pin (GPIO 0) of ESP8266 to connect SDA pin of temperature sensor and D4 pin (GPIO 2) of ESP8266 to connect SCL pin of temperature sensor.

Pin connection between ESP and oled screen

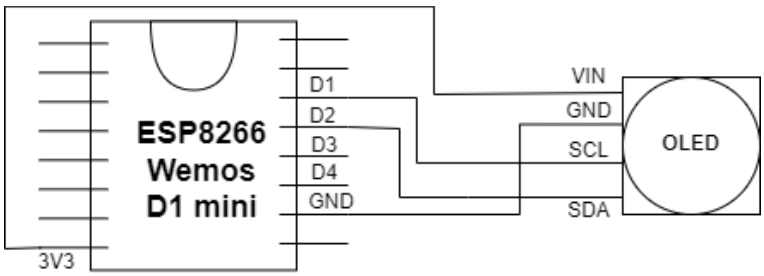

Figure 3.19: *Pin connection between ESP and oled screen*

We also provide the power supply for oled screen. 3V3 pin and ground (GND) of ESP8266 would connect to VIN and ground (GND) of oled screen

respectively. Next, We choose D2 pin (GPIO 4) and D1 pin (GPIO 5) of ESP8266 to connect SDA and SCL pin of oled screen respectively to display data in realtime.

Pin connection between ESP and heart rate sensor

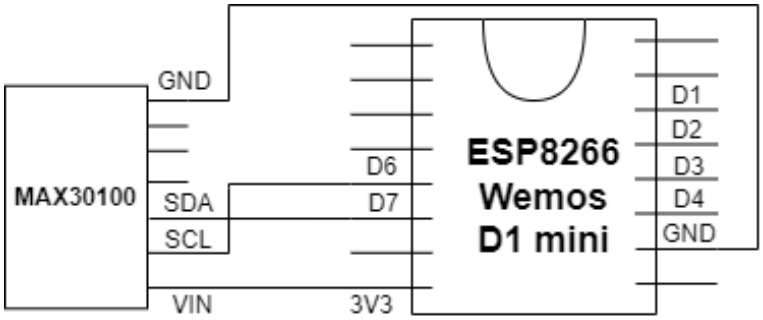

Figure 3.20: *Pin connection between ESP and heart rate sensor*

We also provide the power supply for heart rate sensor. 3V3 pin and ground (GND) of ESP8266 would connect to VIN and ground (GND) of heart rate sensor respectively. Next, We choose D7 pin (GPIO 13) of ESP8266 to connect SDA pin of heart rate sensor and D6 pin (GPIO 12) of ESP8266 to connect SCL pin of heart rate sensor.

We would use pin lipo for power supply of ESP8266 as power of the whole circuit. We summarize the rated voltage and amperage that each device use in the table below:

Table 3.6: *Parameters of components in the circuit*

Hardware device	Rated voltage	Amperage
ESP8266 Wemos D1 Mini	5V DC	40mA
MLX90614 sensor	5V DC	1.5mA
OLED screen	5V DC	8mA
MAX30100 sensor	5V DC	20mA

3.3.8 Flow chart of wearable device

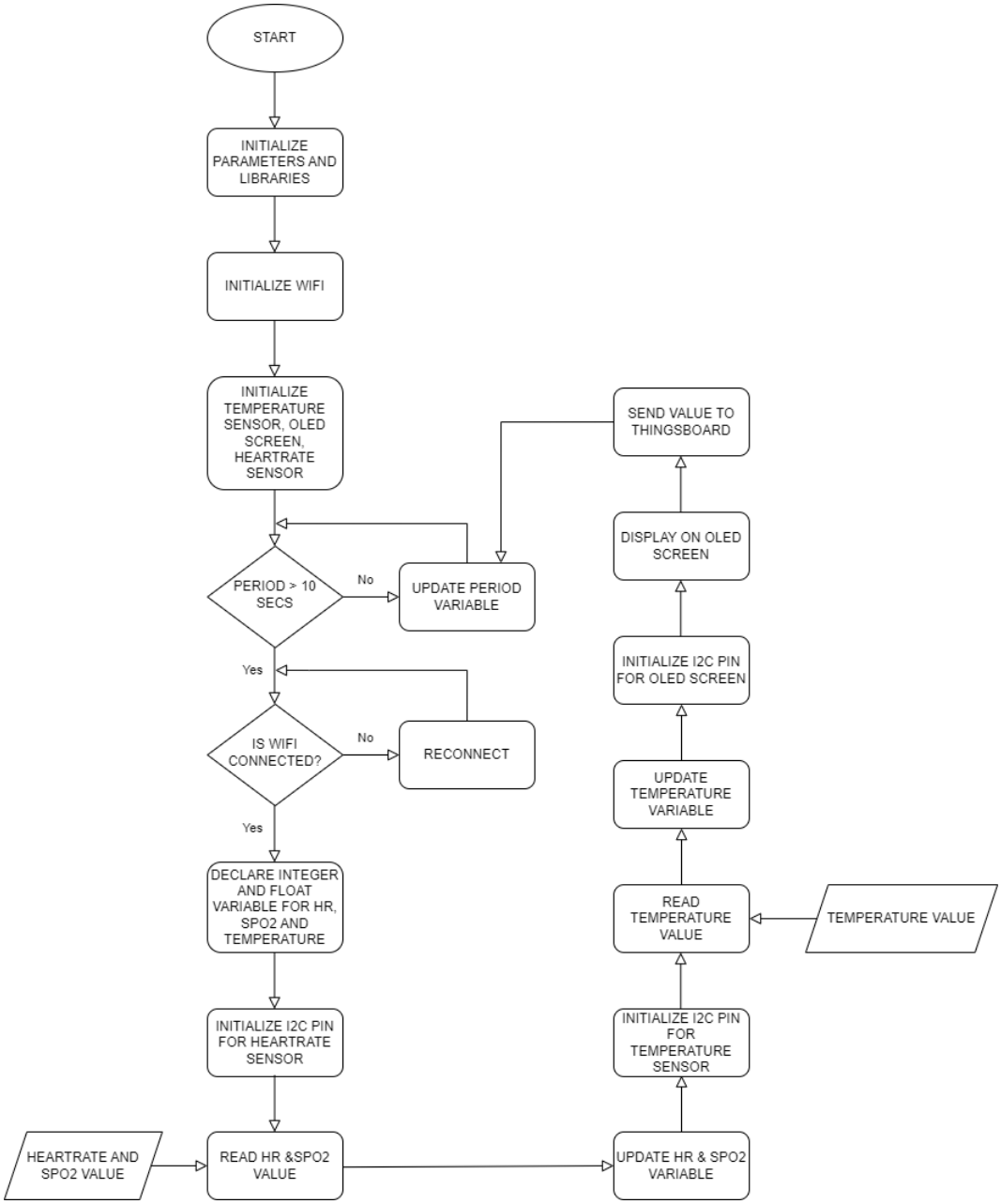

Figure 3.21: *Flow chart of whole wearable system*

We would initialize parameters and libraries that we would use of the whole program. After that, we also initialize WiFi. Then, we initialize temperature sensor, oled screen and heart rate sensor before checking the period variable. If this value is smaller than 10, we would update this value and re-check as previous step. Otherwise, we would check if the connected or not. If it is not connected, we

would reconnect and turn to previous step. If it is connected, we would declare integer and float variable that used for store heart rate, spo2 and temperature value. Next, we would initialize I2C pin of micro-controller to read heart rate sensor value. Micro-controller would read the value from the heart rate sensor via I2C communication standard and then update the heart rate and spo2 variable. We would do the same as temperature sensor and update the temperature variable. We also initialize I2C pin for oled screen to display value in real time. The reason why we must initialize pin of micro-controller before reading value of each sensor is that all sensors and oled screen used mutually I2C communication protocol. Next, we would send these values to thingsboard cloud thanks to MQTT protocol. Finally, we update the period variable and back to loop.

3.3.9 Implementation

We would perform the implementation of PCB schematic, programming system, hardware assembly and test circuit. The implementation process includes schematic design, simulation 3D of the whole circuit and result of the real system.

Implement printed circuit

We use the Altium 16 Designer, which is a unified design environment, empowering engineers with a single view of every aspect of the PCB design process from schematic, to PCB layout, to design documentation. After that, we conduct circuit printing and ironing. At the end of process, we would use multi-meter(VOM) in order to test the error of input and output circuit while we implement.

Initially, we design schematic diagram. We connect pins between micro-controller and sensors as 3.3.7 section describe.

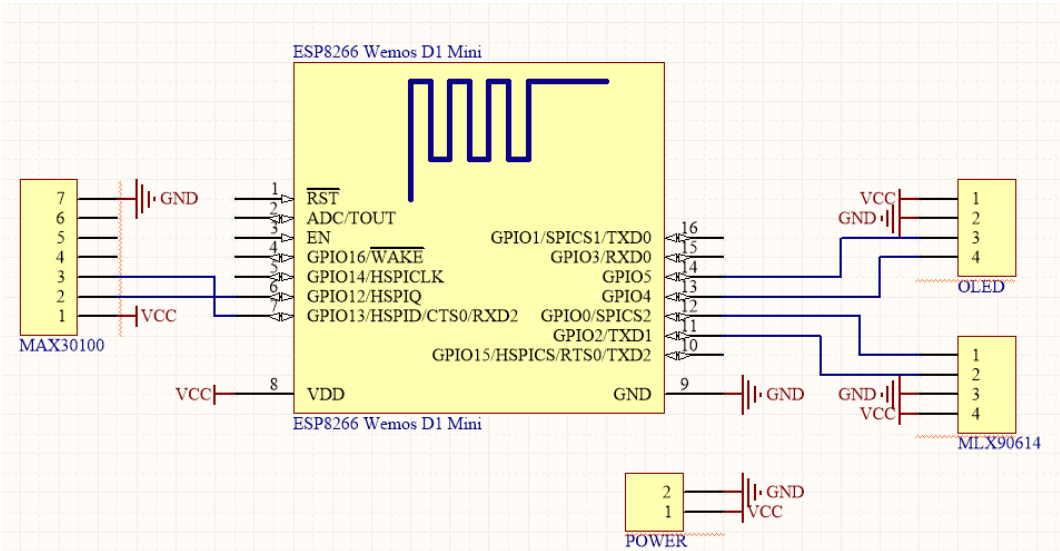

Figure 3.22: *Schematic diagram of whole circuit*

Next, we update all components to PCB. We also need to rearrange the position of all components accordingly. Then, we choose size of the board and cut it. Before wiring, we should add rules to control the width and distance between wires. Finally, we process copper pour for circuit.

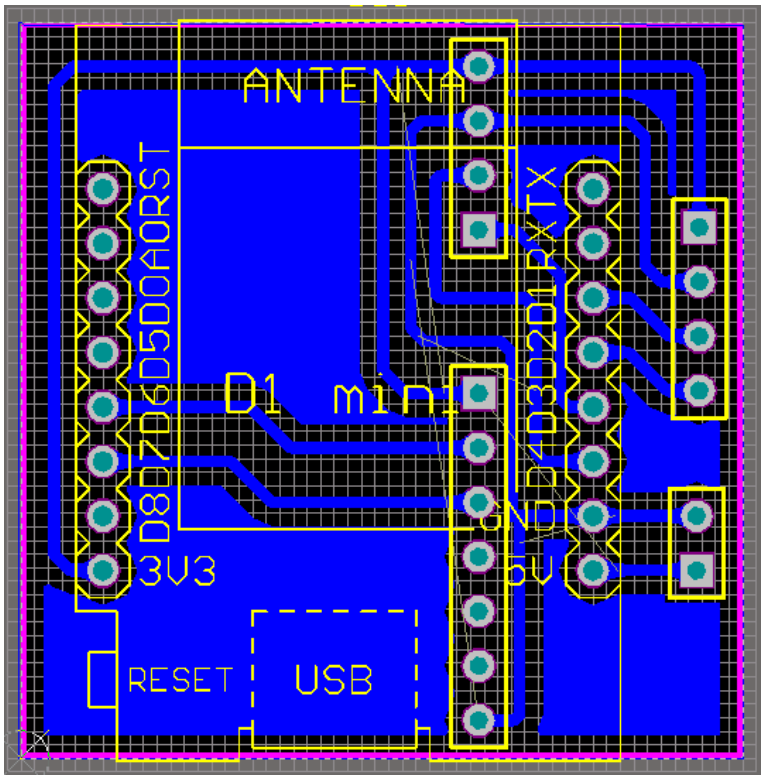

Figure 3.23: *PCB layout*

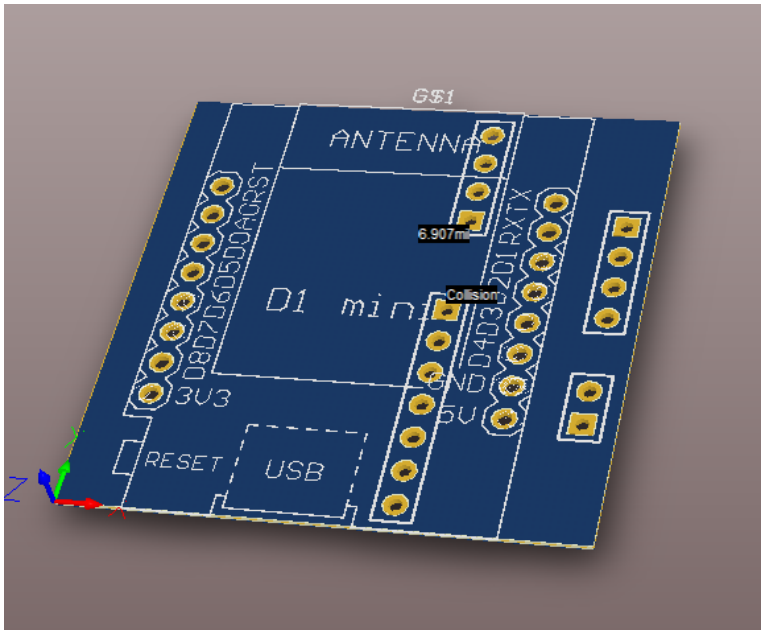

Figure 3.24: *3D Simulation of whole circuit*

Assembling and testing

After finished drawing the printed circuit. We do the assembly of all components and solder the components to the board.

Figure 3.25: *View front and back of device*

Installing Arduino IDE

In this project, we would use Arduino IDE 2.0.1 version because it is convenient and good support for users. After we have already installed it, we must add ESP8266 board manager URL to install ESP8266 platform. Finally, we choose ESP8266 Wemos D1 Mini as our board we use in this project. We also choose right COM port that computer use and serial debug baud to install code.

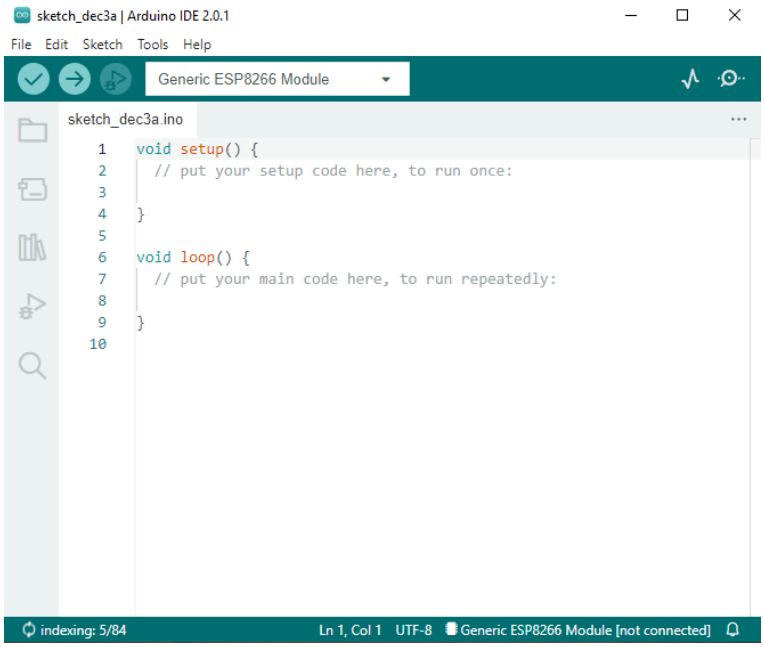

Figure 3.26: *Interface of Arduino IDE*

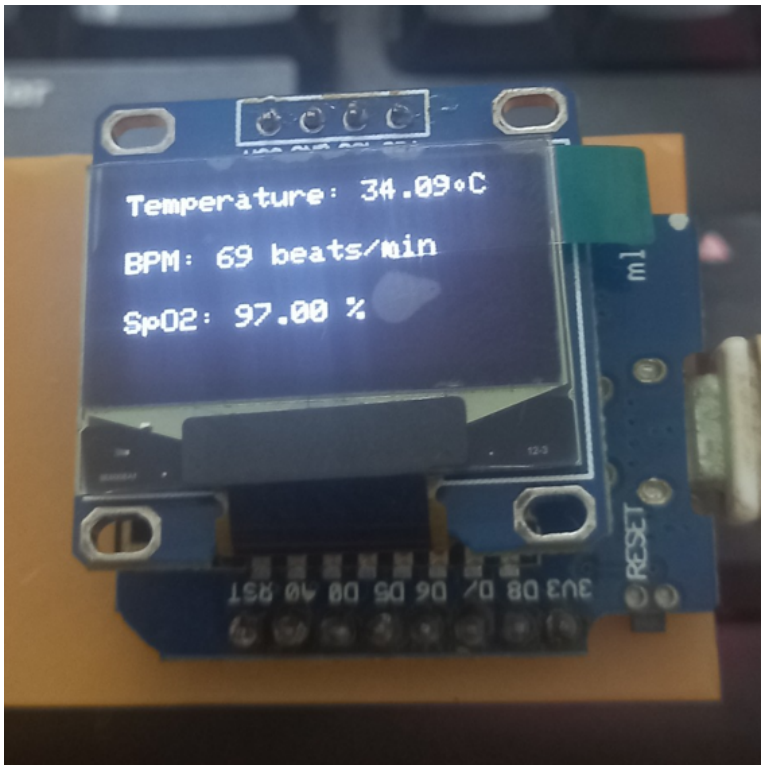

Figure 3.27: *Result device display*

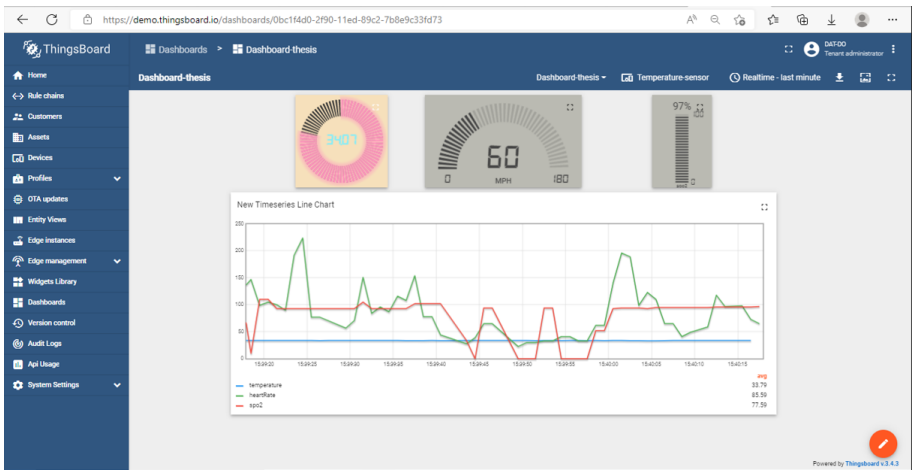

Figure 3.28: *Dashboard of thingsboard*

3.4 Backend Implementation and Deployment

After the hardware and IoT device implementation step has been completed, the next step in the implementation phase is backend implementation. This step

involves building the server-side components of the system, such as the database server and any APIs or other services that will be used to access the data.

3.4.1 Creating ThingsBoard server

Two most common ways of installing open-source ThingsBoard server is by using linux operating system or via docker on Windows Subsystem for Linux. One advantage of installing ThingsBoard on Linux is that it gives you more control over the installation process and the configuration of your ThingsBoard instance. By installing ThingsBoard directly on a Linux system, you can customize the installation and configuration to meet your specific needs and requirements. This can be useful if you want to use custom plugins or integrations, or if you need to fine-tune the performance or security of your ThingsBoard instance.

On the other hand, installing ThingsBoard via Docker on Windows has the advantage of being easier and more convenient. Docker is a popular open-source platform for building, deploying, and running applications in containers. Containers are a type of lightweight, standalone, and portable package that contains everything an application needs to run, including the code, libraries, dependencies, and runtime. This makes it easy to run an application on any system that supports Docker, without worrying about compatibility or dependencies. One of the key benefits of Docker is that it enables developers and system administrators to easily package and distribute applications, and to run them in a consistent and predictable environment. This can help to improve the reliability and portability of applications, and can make it easier to manage and update them. Additionally, Docker provides a number of features and tools for managing and orchestrating containers, such as networking, storage, and security, which can make it easier to build and run complex and distributed applications.

This is why choosing the "thingsboard/tb-postgres" image - a single instance of ThingsBoard with PostgreSQL database, can save time and effort, and can make it easier to get started with ThingsBoard. Additionally, Docker allows you to easily create and manage multiple ThingsBoard instances like this, which can be useful if you need to test or develop different configurations or scenarios. One thing to note is that ThingsBoard is able to use various messaging systems/brokers for storing the messages and communication between ThingsBoard services. For this we chose Kafka, which is also suitable for microservices architecture development.

3.4.2 ThingsBoard's Actor System

In the ThingsBoard platform, an Actor is a user or system that is capable of performing actions, such as sending or receiving data, or accessing resources. Actors can be associated with different roles, which determine the types of actions they are allowed to perform, and the resources they are allowed to access. ThingsBoard supports a hierarchical structure of roles, with different levels of permissions and privileges. At the top of the hierarchy is the SYS_ADMIN role, which has the highest level of privileges, and is able to perform all actions and access all resources within a ThingsBoard instance. Below the SYS_ADMIN role are other roles, such as TENANT_ADMIN, CUSTOMER_USER, and DEVICE, which have varying levels of permissions and privileges.

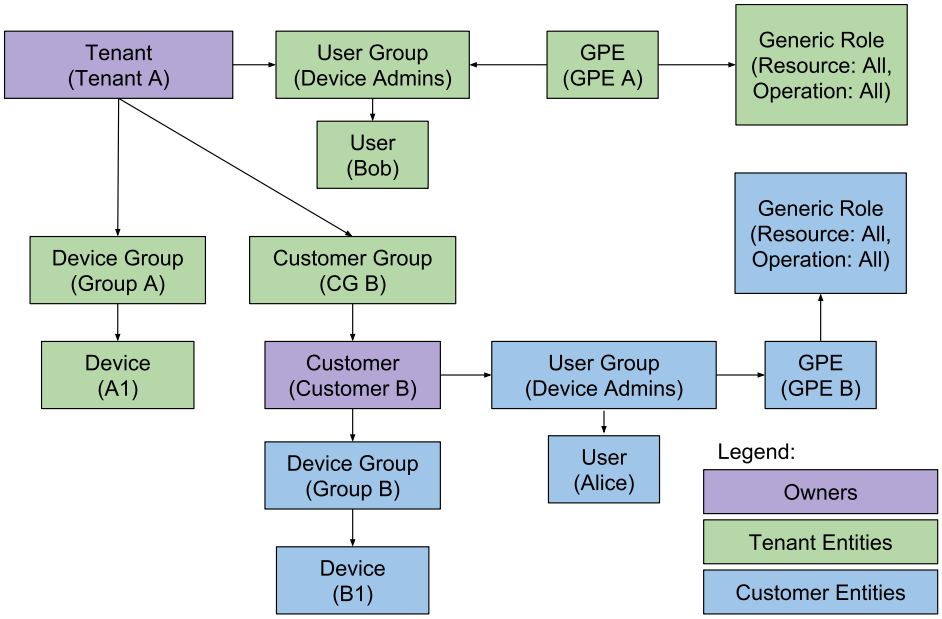

Figure 3.29: *ThingsBoard's Role Hierachy [20]*

A DEVICE actor might represent a particular IoT device, and would be assigned the DEVICE role to enable it to communicate with the ThingsBoard platform. We can establish ThingsBoard as the SYS_ADMIN and give it the authority to add DEVICE and create device profiles for IoT wearables. This will enable us to manage and monitor your wearable devices using the ThingsBoard platform, and to access the data and other resources they provide.

One of the ways to use the ThingsBoard API is to use the REST API. We will need to know the base URL for our ThingsBoard instance, which is typically

in the form of http://<host>:<port>/api. We will also need to authenticate with the ThingsBoard instance in order to access the API. This can be done using an access token, which can be obtained from the ThingsBoard web interface, or using a username and password. Once we have the base URL and an access token, we can use the REST API to perform a variety of tasks, such as creating or updating devices, retrieving or storing data. The REST API uses a standard set of HTTP methods, such as GET, POST, PUT, and DELETE, to perform different actions, and it uses JSON or XML to encode and decode the data that is exchanged between the client and the server, which in this case is between our server and ThingsBoard instance.

3.4.3 Configuring ThingsBoard's Rule Engine

ThingsBoard Rule Engine is a highly customizable and configurable system for complex event processing. With rule engine we are able to filter, enrich and transform incoming messages originated by IoT devices and related assets. We are also able to trigger various actions, for example, notifications or communication with external systems. There are 3 main components:

- **Message:** Any incoming event. It can be an incoming data from devices, device life-cycle event, REST API event, RPC request, etc.
- **Rule Node:** A function that is executed on an incoming message. There are many different Node types that can filter, transform or execute some action on incoming Message.
- **Rule Chain:** Nodes are connected with each other with relations, so the outbound message from rule node is sent to next connected rule nodes.

Because we need to validate data and modify for incoming telemetry or attributes before saving to the database. we will modify default rule chain and will add filter rule node with temperature, heart rate and spo2 validation script. We will place these rule node between default "message type switch" and "save timeseries" rule nodes. We must have removed irrelevant rule nodes from the root rule chain as well.

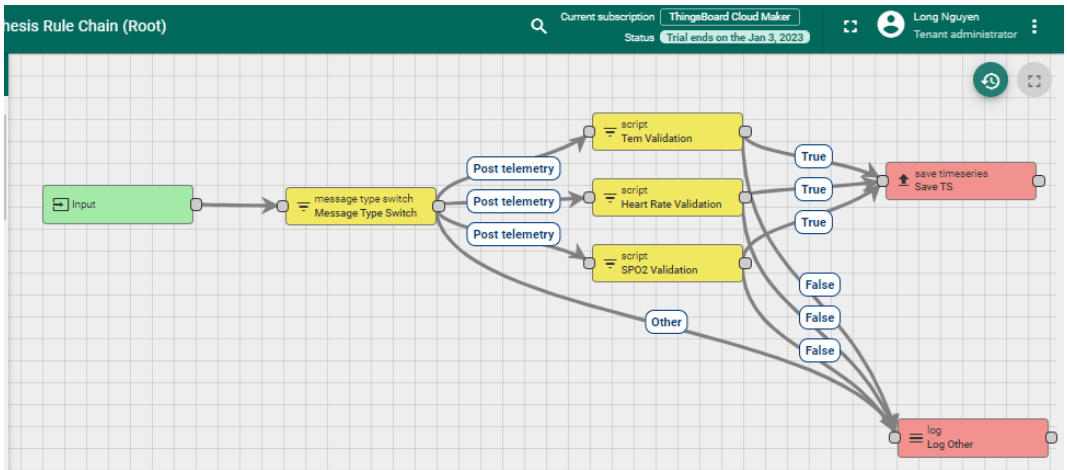

Figure 3.30: *ThingsBoard's Rule Chain Implementation*

We will add condition for script to constraint the range of temperature, heart rate and spo2 that we would collect. If temperature, heart rate and spo2 property not defined or valid - script will return True, otherwise it will return False. If script returns True incoming message will be routed to the next nodes that are connected with True relation. We want that all telemetry requests pass through this validation script. We connect Message Type Switch node with Script Filter node using Post Telemetry relation. We need to connect Script Filter node with Save Telemetry node using True relation. Therefore, all valid telemetry will be saved. We also will connect Script Filter node with Log Other node using False relation. So that all not valid telemetry will be logged in the system log.

3.4.4 Backend and API designs

WebSocket is a protocol that enables full-duplex, bi-directional communication between a web server and a web browser, or a server to another server. In order to use WebSocket on the server-side, we need to ensure that our web server supports incoming WebSocket connections, and that this support is enabled and configured properly. This is because a WebSocket connection is a continuous, long-running connection, and requires a long-running process to handle it. Node.js is a powerful and flexible platform for building WebSocket-based applications. This is because Node.js is already a long-running process, and so it does not require any changes to the architecture to support WebSocket connections. Instead, it just requires the use of a library that supports the WebSocket protocol. This makes Node.js a convenient and effective choice for implementing WebSocket-based applications, and is why we chose it for this purpose. In addition to supporting WebSocket, the

backend built upon the Express.js framework in a Node.js environment can also connect to ThingsBoard through the REST API and CRUD (Create, Read, Update and Delete) operations. This allows the backend to access and manipulate data and resources within a ThingsBoard instance, and to integrate with the ThingsBoard platform in various ways.

Another advantage of using Node.js for WebSocket implementation is that it uses JavaScript as its programming language. This means that the same language can be used for both the frontend and backend of the application, which can improve compatibility and consistency between the different components of the application. By using JavaScript on both the frontend and backend, you can share code, libraries, and other resources between the two sides of the application. This can make it easier to develop and maintain the application, and can reduce the complexity and duplication of code. Additionally, using the same language on both the frontend and backend can make it easier for developers to switch between the different parts of the application, and to collaborate and work together more effectively.

Once the components and modules of the backend have been implemented, we connect it to PostgreSQL, and create the appropriate database schema, tables, and indexes, as well as configuring and optimizing the DBMS to support the data storage and management requirements of the backend.

3.4.5 Alarm and Patient's Status

Condition	Status Sign	Treatment
37.5 <= Temp <= 38.5 & 0.15 <= Time <= 48.0	Yellow	Incubation phase (chest x-ray and abdominal ultrasound)
39.0 <= Temp <= 40.0 & 0.15 <= Time <= 23.99	Orange	Febrile phase (monitoring for warning signs and other clinical parameters)
37.5 <= Temp <= 38.5 & 24.0 <= Time <= 48.0	Red	Emergency phase (measure hematocrit every 1-2 hours, once every 6 hours)
37.5 <= Temp <= 38.5 & 0.15 <= Time <= 48.0	Green	Recovery phase (close monitoring is necessary to recognize heart failure or pulmonary edema)

Table 3.7: *Medical Rules and Dengue Fever Treatment [21]*

The table above lists the rules for medical actions that need to be taken if a patient's body temperature, measured by their wearable device, reaches a critical level. The wearable device constantly sends the wearer's real-time body temperature to the coordinating server. Many of these patients are in serious health conditions and may not be able to cooperate voluntarily. The server keeps track of the patient's body temperature and decides what to do next in a rule-based manner. For example, as listed in the first row, if the patient's body temperature falls between 37.5 and 38.5 degrees Celsius for over 10 minutes and less than 48 hours, the patient's wearable will turn yellow to signify an incubation phase. At the same time, the coordinating server activates the medical rule to notify the doctor or nurse responsible for the patient. The doctor or nurse is then supposed to perform an X-ray and watch for other symptoms.

This update of the patient's medical condition is illuminated in colors that indicate the severity of their condition. For example, yellow may indicate an incubation phase, while red may indicate a more serious condition. By using this system, doctors and nurses can quickly and easily monitor the health of their

patients and take appropriate actions to ensure their well-being. These color-indicating statuses are also implementation in our dashboard of real application.

3.4.6 User Authentication and Authorization

When it comes to user authentication on the website, there are several different options and approaches that can be used, each with their own benefits and drawbacks. In this case, we have chosen to use JavaScript Web Tokens (JWTs) for authentication, rather than cookies. One reason for this choice is that JWTs are more secure than cookies. Cookies are typically stored in the browser, which means that they are vulnerable to tampering, theft, or other forms of attack. In contrast, JWTs are cryptographically signed, which means that they cannot be altered or forged without the private key that was used to sign them. This makes JWTs much more resistant to tampering and other security threats, and provides a higher level of security for the authentication system.

Another reason for choosing JWTs is that they are more portable and flexible than cookies. JWTs are self-contained, which means that they can be easily passed around between different systems and applications. This makes it easier to integrate the authentication system with other components and services, and to support different types of devices and platforms. Additionally, JWTs can be used to encode additional information and claims about the authenticated user, which can be useful for implementing advanced features and functionality.

Firebase Authentication is a cloud-based authentication platform that provides a range of tools and features for implementing authentication in web and mobile applications. To use Firebase Authentication, we first need to create a Firebase project, and enable the Authentication service for the project. This can be done using the Firebase Console, which is a web-based interface for managing and configuring Firebase projects. Once the Authentication service is enabled, we can use the Firebase SDK to integrate the authentication system with our application. The Firebase SDK provides a set of libraries and tools that can be used to implement authentication features, such as user registration, login, password management, and two-factor authentication. The Firebase SDK also provides support for generating and validating JWTs, which can be used to authenticate users and secure the application. To implement authentication using JWTs, we can use the Firebase SDK to create a custom authentication flow that generates and signs JWTs for authenticated users. This can be done by implementing the necessary functions and logic for handling user registration, login, and other authentication-related

tasks. Once the authentication system is in place, we can use the Firebase SDK to validate JWTs on incoming requests, and to enforce access controls and permissions based on the information encoded in the JWTs.

3.4.7 Deployment

When it comes to hosting a fullstack project on a budget, there are several different options to consider. The first option is to use a cloud provider, such as AWS, Azure, or Google Cloud. This can be a cost-effective way to host a fullstack project, as many cloud providers offer free tiers or other pricing options that are suitable for small or medium-sized projects. However, one of the drawbacks of using a cloud provider is that they can be complex and difficult to use. Many cloud providers offer a wide range of services and tools, which can be overwhelming for developers who are new to the platform. For example, AWS itself currently has around 200 services, many created for similar use-cases. Additionally, cloud providers often require a steep learning curve, and can be challenging to configure and manage, especially for large or complex projects.

Another option for hosting a fullstack project on a budget is to self-host the project. This involves hosting the different components of the project on different platforms or services, such as a server hosting provider, a database hosting provider, a frontend hosting provider, and an image storage provider. For the server and database hosting, we have chosen to use Vultr virtual private server (VPS) as our hosting provider. This is because Vultr VPS offers a range of features and benefits that make it a good choice for hosting a fullstack project on a budget.

First, Vultr VPS is affordable and cost-effective. It offers a range of pricing options that are suitable for different budgets and requirements, and includes a free trial period to allow you to try out the service before committing to a subscription. Second, Vultr VPS is easy to use and manage. It provides a simple and intuitive control panel that allows us to easily set up and configure our server and database, and to manage and monitor our resources. Third, Vultr VPS is scalable and flexible. It allows us to easily add and remove resources, such as storage, CPU, or memory, as our project grows and changes. This makes it a good choice for projects that are expected to scale up or are themselves microservices, and need a hosting provider that can support those changes.

To push ThingsBoard image to Vultr VPS via Docker, we login to our Vultr VPS account, and create a new virtual machine (VM) instance. Once the VM is created, we will need to configure it to support Docker, then we can push the

ThingsBoard image to the VM using the Docker command-line interface (CLI). From then on, we use the Docker CLI to run the image. The process is the same for the PostgreSQL DBMS.

3.5 Frontend Implementation and Deployment

3.5.1 Frameworks

The next and final step in the implementation phase is frontend implementation. This step involves building the user-facing components of the system - the frontend, which plays a crucial role in enabling users to access and interact with data in a user-friendly and intuitive way. For this reason, it is important to choose a frontend development framework that is well-suited for building interactive and user-friendly applications. One framework that is particularly well-suited for this purpose is React.js.

	Portfolio	Content	Storefront	Social Network	Immersive
Holotype	Personal Blog	CNN	Amazon	Facebook	Figma
Interactivity	Minimal	Linked Articles	Purchase	Multi-Point, Real-time	Everything
Session Depth	Shallow	Shallow	Shallow - Medium	Extended	Deep
Values	Simplicity	Discover-ability	Load Performance	Dynamicism	Immersiveness
Routing	Server	Server, Hybrid	Hybrid, Transitional	Transitional, Client	Client
Rendering	Static	Static, SSR	Static, SSR	SSR	CSR
Hydration	None	Progressive, Partial	Partial, Resumable	Any	None (CSR)
Example Framework	11ty	Astro, Elder	Marko, Qwik, Hydrogen	Next, Remix	Create React App

Figure 3.31: *Framework comparisons [22]*

There are several reasons why React.js is a good choice for a patient management application. One of the key advantages of React is that it is a component-based framework, which means that it allows developers to build applications by composing small, reusable pieces of code, known as components. This makes it easy to build complex and scalable applications without sacrificing maintainability. Additionally, React is designed to support the development of Single Page Applications (SPAs), which are web applications that load all of their content on a single page, and that use JavaScript to dynamically update the page content in response to user actions. This is well-suited for patient management applications, where interactivity is important for enabling users to quickly and easily access and manipulate data. Another advantage of React is that it is fast and efficient. React uses a virtual DOM (Document Object Model) to update the page content, which means that it only updates the parts of the page that have changed, rather than re-rendering the entire page. This makes React applications highly performant, and enables them to provide a smooth and responsive user experience.

In comparison to other frontend development frameworks, such as Angular.js and Vue.js, React has a number of advantages. For example, React is generally easier to learn and use than Angular, which can be complex and difficult to work with. Additionally, React is more lightweight and flexible than Vue, which can make it easier to integrate with other technologies and tools. Finally, React has a large and active community, which provides a wealth of resources, libraries, and other support for developers.

In order to build a robust and feature-rich application, aside from React.js we can use a number of additional tools, libraries, and packages. One such tool is react-query, which is a library for fetching, caching, and updating asynchronous data in React applications. react-query provides a set of hooks and APIs that can be used to manage and manipulate data in a declarative and reactive way, and to automatically handle common data-related tasks, such as pagination, caching, and error handling. Another tool that can be useful in a React.js application is react-router, which is a routing library for SPAs, by which we can create a consistent and intuitive navigation experience for our users, and enable them to move easily and smoothly between different pages, components, and views. Additionally, Chart.js provides a simple and intuitive API for creating a variety of different chart types, including bar charts, line charts, pie charts, and scatter plots. It also offers a range of customization options and features, such as labels, legends, tooltips, and animations, which can be used to create professional-looking and interactive charts.

For styling we use Material-UI - a React.js component library that provides a set of pre-built, customizable, and themeable components that can be used to create a modern and professional-looking user interface. By combining TailwindCSS and Material-UI, we can quickly and easily create a beautiful and functional design for our application.

As a bonus point, we prefer Vite over traditional Webpack, which is a build tool and development server for modern JavaScript applications that provides a fast and efficient way to build, serve, and develop React.js applications, and can improve the speed and performance of our development workflow.

3.5.2 Deployment

For our frontend, we use Vercel which is a popular and widely used Content Delivery Network (CDN) that offers a range of features and benefits for hosting websites and other web-based applications. One of the main advantages of Vercel is that it is easy to use and configure, and provides a simple and intuitive interface for managing and deploying your website. Additionally, Vercel is fast and scalable, and can handle a large volume of traffic and requests without sacrificing performance or reliability. After our project is configured, we push our website files and assets to the Vercel servers, using a version control system (VCS) like Git. Once our website is deployed, Vercel will automatically build and publish it, and make it available to users through the Vercel CDN.

For hosting user-uploaded images, we use a specialized image hosting service such as imgix. imgix is a cloud-based image hosting and processing platform that provides a range of tools and features for managing and delivering images, such as various of image formats and transformations, for example, resizing, cropping, and color correction, and can be easily integrated with other web-based applications and services.

3.6 Scaling

To devise a scaling plan for a microservices-based system, we considered several approaches. One approach to horizontal scaling would be to use a load balancer to distribute the workload across multiple instances of the microservices. This would involve adding additional instances of the microservices to the system, and configuring the load balancer to distribute the incoming traffic across these instances. This would allow the system to handle a larger workload, and would

improve the overall performance and reliability of the system.

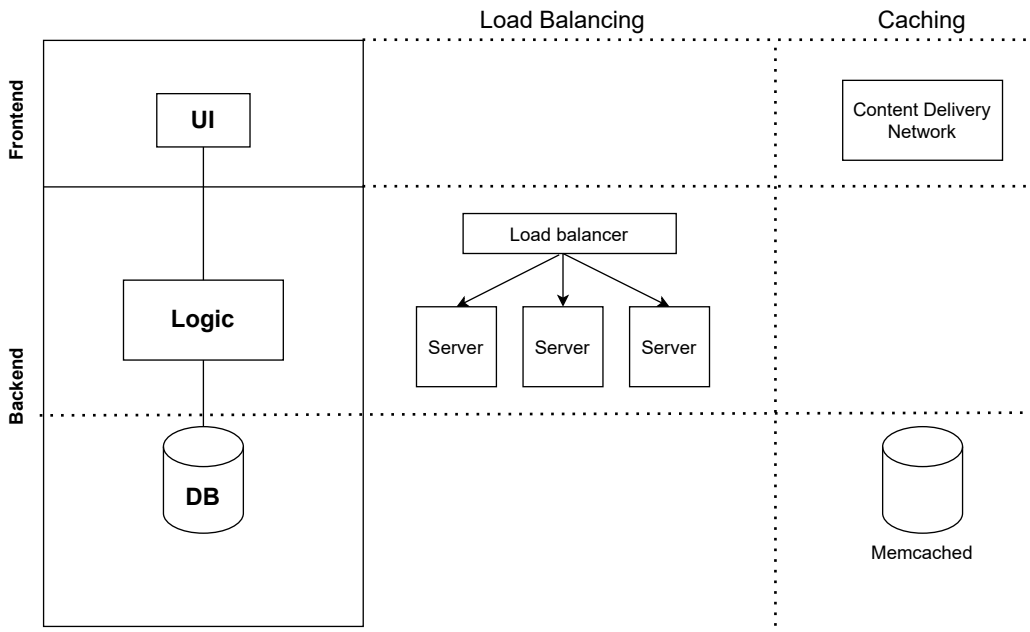

Figure 3.32: *Scalable components*

Another approach to horizontal scaling would be to use a distributed cache to improve the performance of the system. This would involve implementing a distributed cache, such as Memcached or Redis, and configuring the microservices to use the cache for data storage and retrieval. This would improve the performance of the system by reducing the amount of time and resources required to access data, and would help to ensure that the system can handle a larger workload.

A third approach to horizontal scaling would be to use a distributed database to support the data storage and management requirements of the system. This would involve implementing a distributed database, such as Cassandra or MongoDB, and configuring the microservices to use the database for data storage and retrieval. This would improve the performance and reliability of the system, and would help to ensure that the system can handle a large and growing amount of data.

3.7 Final look

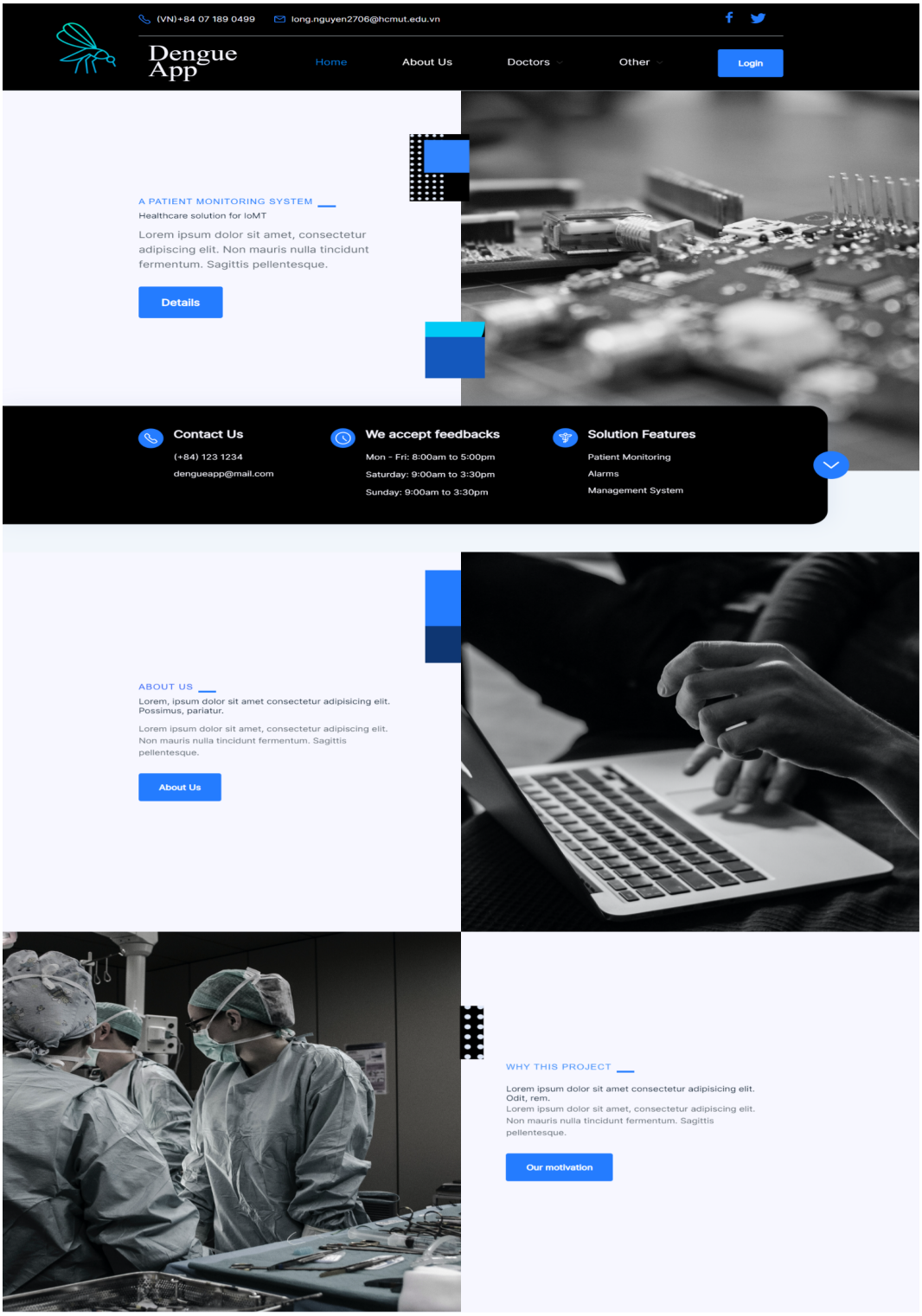

Figure 3.33: *Home page*

Taking inspiration from other healthcare websites and applications, The home page of DengueApp, our healthcare solution, is designed to look like a landing page, with a clean and modern design and cool colors that help to create a professional and trustworthy aesthetic. The layout of the page is carefully crafted to guide the user's attention to key features and information, and to encourage them to explore the site and learn more about the application. The logo, a cyan-colored mosquito, is prominently displayed at the top of the page, along with the website's name, DengueApp, on the navigation bar.

Hard and soft features are carefully balanced on the page to create a user-friendly and engaging experience. Hard features, such as image sections and well-defined buttons, help to guide the user and provide a clear path for interacting with the site. Soft features, such as pleasant colors and engaging images, help to create a visually appealing and inviting atmosphere.

Figure 3.34: *Login and Signup pages*

The login and sign up pages allow users to create an account and log in to the site. In order to create an account, users typically need to provide their email address and choose a password. The page may also ask users to select their role, such as doctor, nurse, or patient. This information is used to create a unique account for each user and help ensure that they have access to the appropriate features and functionality within the site or app. The site will send an activation link to the user's email and through that link user can confirm their account creation. Once users have created an account, they can log in to the site or app using their email address and password. This allows them to access their personal account and any features or resources that are available to them based on their role.

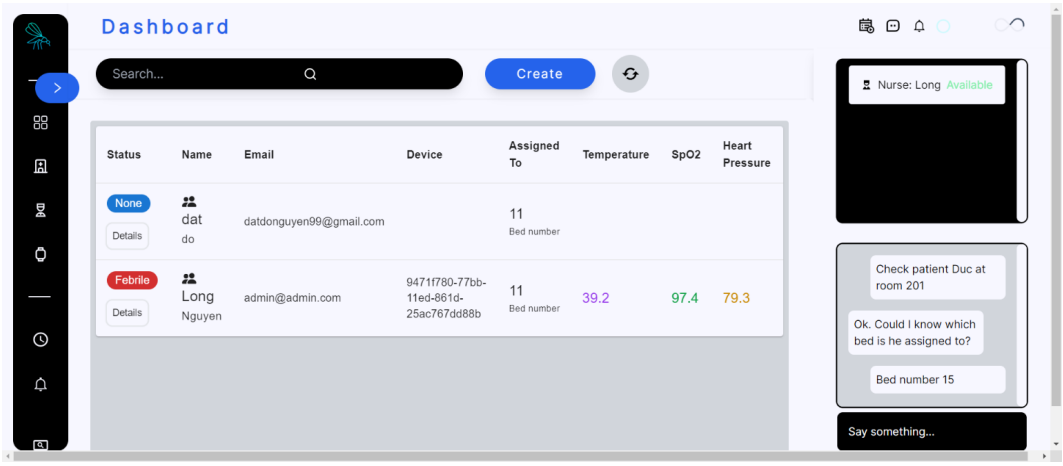

Figure 3.35: *Dashboard*

After logging in, user as a doctor is met with a dashboard with a sidebar on the left, a search bar, and a container displaying a list of patients available in the system is a type of user interface that is commonly used in healthcare applications. The sidebar on the left typically contains a navigation menu that allows users to access different sections of the dashboard. The search bar at the top of the page allows users to quickly find specific patients or information within the system. The container in the middle displays a list of patients available in the system, along with their relevant information such as their name, email, and current telemetries. On the right side of the dashboard, there is a chatbox that allows doctors to send messages to nurses. This allows doctors and nurses to communicate and collaborate on patient care in real-time. The account icons are typically located on the top right corner of the dashboard, allowing users to access their personal settings and account information.

This status of the patient's medical condition is illuminated in colors that indicate the severity of their condition. For example, yellow may indicate an incubation phase, while orange means febrile phase and red indicates a more serious condition. By having this status, doctors and nurses can quickly and easily monitor the health of their patients and take appropriate actions to ensure their well-being.

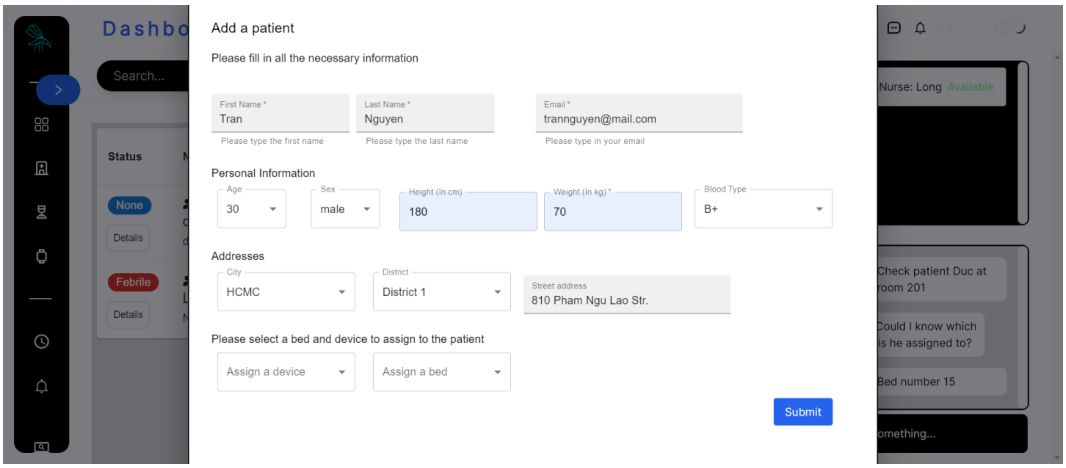

Figure 3.36: *Patient Registration*

A modal will pop up after the user clicks the "create" button next to the search bar. The modal asks the user to register a new patient by providing the patient's first and last name, email address, age, sex, height, weight, blood type, city, district, and street address. Optionally, the user can also assign a device and a bed for the patient now, or leave them blank and assign them later. These information will be used to create a new patient record within the system, allowing doctors and nurses to access the patient's information and provide care. Once the user has entered all of the required information and clicked the "create" button within the modal, the new patient record is added to the system and appears in the list of patients on the dashboard.

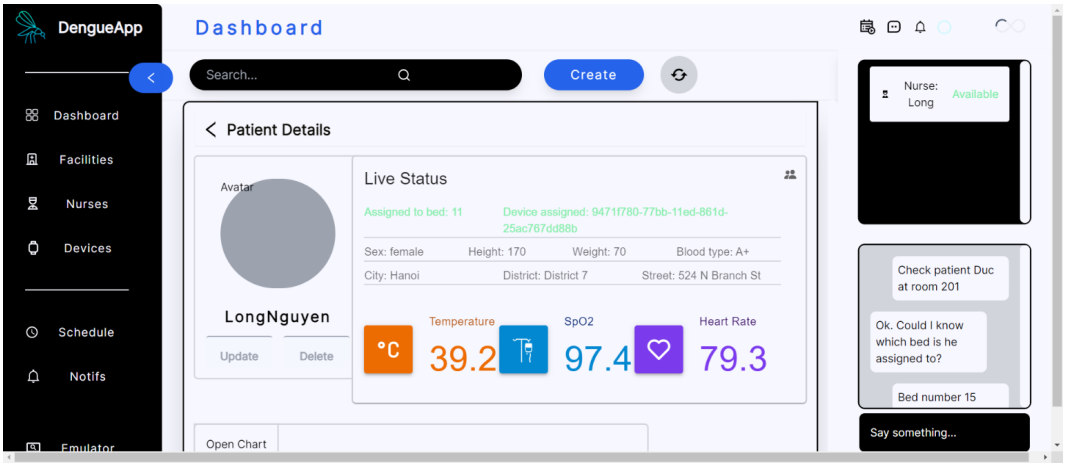

Figure 3.37: *Patient Detail*

The patient details page provides detailed information about a specific

patient. This is among the few pages that can be viewed by both the patient and doctor. On the left side of the page, there is the user avatar, which is a small image or icon representing the patient. Next to the avatar, there are options to update or delete the patient's information. Next to the avatar and options is the live status section, which displays previously entered information about the patient such as their name, age, sex, and medical history. This section also includes real-time telemetry data collected from IoT devices, such as the patient's temperature, blood oxygen level (SPO2), and heart rate. This information is updated in real-time, allowing doctors and nurses to monitor the patient's health and make informed decisions about their care.

Figure 3.38: *Patient's real time telemetry line chart*

When the user clicks the "open chart" button under the live status section, a multi-axis real-time line chart will pop up. The chart displays three datasets: temperature, blood oxygen level (SPO2), and heart rate. Each dataset is represented by a different color and plotted on its own axis, allowing users to easily compare and contrast the different data points. The chart updates in real-time, allowing users to see how the patient's health is changing over time. This can be useful for identifying trends or potential issues, and can help doctors and nurses make informed decisions about the patient's care. The chart can be closed by clicking the "arrow down" button in the top left corner.

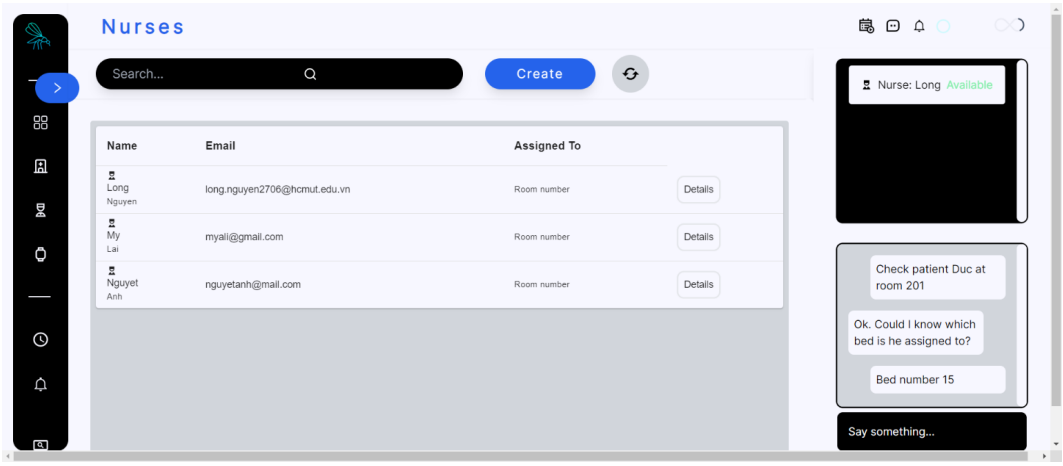

Figure 3.39: *List of nurses*

By navigating to the nurse section in the sidebar, the container will display a list of nurses in a table similar to the patient table. This table includes information about each nurse, such as their name, email and assigned rooms. Users can use the table to search for specific nurses, sort the list by different criteria, or filter the list based on specific characteristics. This can be useful for quickly finding the right nurse for a particular patient or task. The table can also be used to view or edit the details of a specific nurse, or to assign tasks or patients to individual nurses.

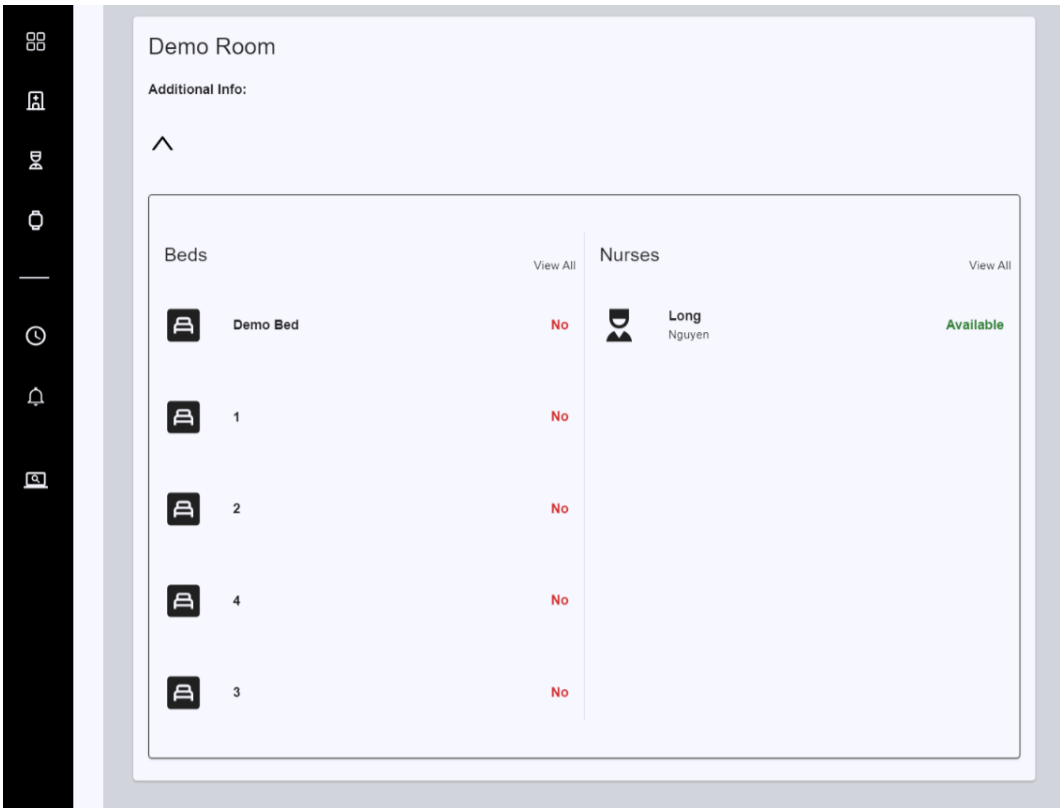

Figure 3.40: *View of facilities (rooms and beds)*

The facility section of the sidebar shows a list of rooms available in the facility. When a user clicks on a specific room, a list of beds and nurses assigned to that room will be displayed. This information can be useful for understanding the current occupancy and staffing levels within the facility, and can help doctors and nurses make informed decisions about where to place patients or assign tasks. The list of beds in the room includes information such as the bed number, whether the bed is currently being occupied. The list of nurses assigned to the room includes information such as the nurse's name as well as their availability. This information can be useful for coordinating care among multiple nurses and ensuring that patients receive the best possible care.

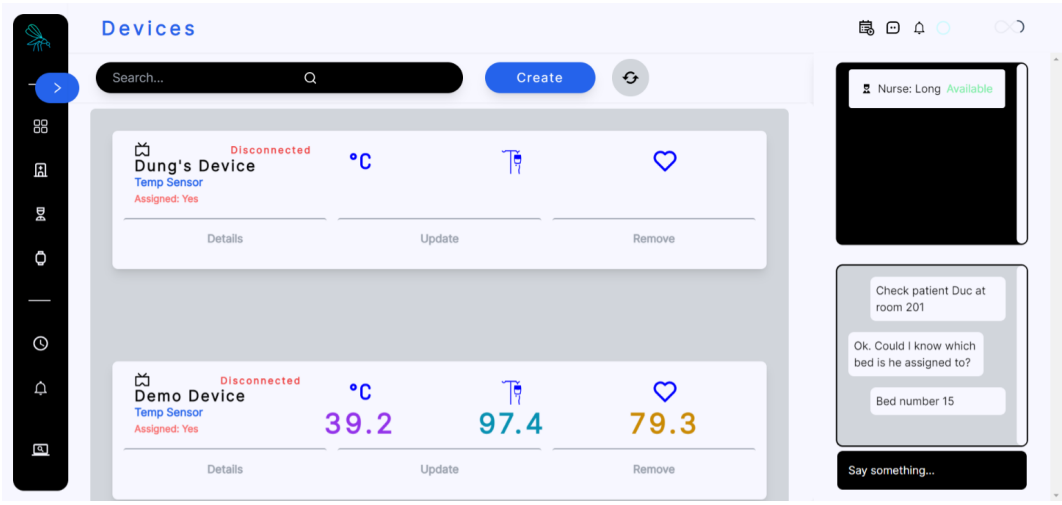

Figure 3.41: *List of current devices*

Finally, in the device section of the dashboard, there is a list of devices that are currently in use or available in the facility. The list includes information about each device, such as its name, type, and any telemetry data that is being collected by the device.

4 Result and Evaluation

4.1 Performance Testing

There are a number of steps and approaches that we can take to test our website and ensure that it is functioning correctly and efficiently. Some common ways to test a website are automated testing and performance testing.

Automated testing involves using specialized software and tools to automatically test the website, and to check for any errors or problems. Automated testing can be performed using a variety of tools and frameworks, such as Selenium, Jest, and Cypress. Among those, we use Jest to efficiently write and execute tests for our React components and functions, and verify that they are working as intended.

Performance testing, on the other hand, involves using specialized tools and techniques to test the performance and scalability of the website, and to determine how the website behaves under different workloads and conditions. Performance testing can help to identify any bottlenecks or limitations in the website, and to optimize the website for performance and efficiency. There are a number of tools and techniques that we can use to perform performance testing on our website. Some common tools and approaches for performance testing include:

- Load testing: This involves using specialized tools and software to simulate a large number of users or requests accessing the website or application at the same time, and to measure the website's performance and behavior under these conditions. Load testing can help to identify any bottlenecks or limitations in the website's performance, and to determine how the website scales and responds to high levels of traffic or activity.

- Stress testing: This involves using specialized tools and software to simulate extreme or unrealistic workloads or conditions, and to measure the website's performance and behavior under these conditions. Stress testing can help to identify the maximum capacity of the website, and to determine how the website behaves when it is pushed to its limits.
- Benchmarking: This involves using specialized tools and software to measure the performance of the website or application under a variety of different conditions, and to compare these results to other websites or applications. Benchmarking can help to identify areas where the website's performance can be improved, and to determine how the website compares to similar websites or applications.
- Profiling: This involves using specialized tools and software to analyze the performance of the website or application at a low-level, and to identify any specific issues or problems that may be impacting the website's performance. Profiling can help to identify specific areas or components of the website that are causing performance issues, and to provide detailed information about how the website is behaving and how it can be optimized.

For load and stress tests, there are many tools and softwares like Apache JMeter, LoadRunner, LoadImpact or k6, which we chose. k6 is a open-source load testing tool that can be used to simulate a large number of users or requests accessing our website or application, and to measure the website's performance and behavior under these conditions. With Grafana, an open-source data visualization and monitoring tool, we can easily and effectively perform load testing and stress testing on our application, and visualize and analyze the performance data collected by k6. Grafana also provides dashboards and reports that can help us understand and interpret the performance data collected by k6, and to identify trends and patterns in the data.

As for benchmarking, WebPageTest, Lighthouse and Google PageSpeed Insights are some common tools. We can use Lighthouse to generate a detailed report for the website or application. This report will include detailed information about the performance of the website, including specific recommendations for improving the performance, and a breakdown of the performance metrics and scores for each page or component of the website. We also need a set of standardized metrics called Web Vitals that can be used for benchmarking the performance of our website. These metrics, which are known as Core Web Vitals, focus on key aspects of the user experience, such as page load time, interactivity, and visual

stability to achieve optimal search engine optimization (SEO). This can help to improve the visibility and popularity of our apps, and to attract and retain more users.

4.2 Device testing

To perform an evaluation of a wearable that combines three sensors: a temperature sensor, a SpO2 (oxygen saturation) sensor, and a heart rate sensor, we would first need to identify the specific aspects of the device that we want to evaluate. In this case, its performance, reliability, and accuracy of the temperature, SpO2, and heart rate sensors. Evaluation criteria or metrics that are used to assess the performance of the temperature sensor are accuracy, stability, and response time. For the SpO2 sensor, we use metrics such as accuracy, repeatability, and sensitivity. For the heart rate sensor, those are accuracy, precision, and dynamic range. In future, we might also use metrics such as user satisfaction, ease of use, and security to evaluate the overall functionality and usability of the device.

Next is the evaluation process. This involves using specialized tools or software to assess the performance of the device, for some sensors we were be able to use the device itself to perform the evaluations. For example, we might use a temperature chamber to test the accuracy and stability of the temperature sensor, or we might use a pulse oximeter to test the accuracy and repeatability of the SpO2 sensor. Once we have collected and analyzed the data, we use the results of the evaluations to identify any issues or concerns with the performance of the device. This may involve identifying specific areas where the device is underperforming, or areas where there are potential security vulnerabilities or other risks. For example, if the temperature sensor is not accurate or stable, or if the SpO2 sensor is not sensitive or repeatable, we would need to identify the root cause of these issues and determine how to address them.

In order to compare the actual measurement results of our circuit with the accurate measuring device. The device used to measure the results in this study is the Microlife B2 heart rate and blood pressure monitor and Microlife body temperature.

Figure 4.1: *Device measure heart rate Microlife-b2-basic*

Figure 4.2: *Device measure body temperature Microlife*

Table 4.1: *Device test results*

	Heart rate		Temperature	
	Device (Beat/min)	Microlife blood pressure & heart rate (Beat/min)	Device (Celsius Degree)	Microlife infrared thermometer (Celsius Degree)
Do Nguyen Dat	90	92	33.61	35.1
	86	81	34.07	36.2
	82	83	34.25	36.5
	75	75	34.37	36.8
	71	76	34.59	36.4
	76	73	34.09	35.9
	86	80	34.69	36.3
	78	82	34.61	36.6
	73	77	33.87	36.8
	93	89	34.13	36.1

Based on the obtained results, we see the heart rate measurement results: the parameters are quite stable, there is a small error between the device and the construction circuit. The temperature measurement results are quite different because the temperature when soldering damages the sensor.

5 Conclusion and Future Work

5.1 Achievement

In conclusion, we have achieved our goal of building an application that would help hospitals monitor patients with dengue fever during outbreaks. We achieved this goal by designing and implementing a microservices-based system that included a distributed network of IoT devices, as well as an application for patient management. To design the system, we performed a thorough requirements analysis to identify the functional and non-functional requirements of the system. We then used this information to design the system architecture, including the microservices and the database. We also designed the database to ensure that it was able to support the data storage and management requirements of the system.

To implement the system, we first implemented the IoT devices, which were responsible for collecting and transmitting data from the patients. We then coded the application, which was responsible for processing and analyzing the data, as well as providing real-time data streaming and alerts to the healthcare workers. Finally, we performed testing to ensure that the system was reliable and secure. This included both performance tests and device tests, which allowed us to evaluate the performance of the system and the IoT devices under different conditions and scenarios.

Access our code on Github: https://github.com/nhlong27/dengueapp

5.2 Future Plans & Development

As the project continues to evolve and expand, we have identified several opportunities for further development. One of the key areas of focus is to expand the original goal of the project, which was to help hospitals monitor patients with dengue fever during outbreaks, to other diseases outside of dengue. By doing this, we will be able to provide a more comprehensive solution that can help hospitals monitor and manage patients with a wider range of conditions.

Another key area of focus is to implement machine learning models using TensorFlow.js to help predict patients' conditions in the future. By using machine learning, we will be able to develop predictive models that can help healthcare providers anticipate potential complications and intervene early to prevent them. This will improve the overall effectiveness and efficiency of the system, and will help to improve patient outcomes.

We also plan to improve the IoT device by adding an alarm device and a screen for displaying data on the wearables. This will allow patients to quickly and easily access important information about their condition, and will provide healthcare providers with a more comprehensive view of the patient's status. By adding these features, we will be able to improve the usability and functionality of the system, and will be better able to meet the needs of patients and healthcare providers.

To accommodate the increasing user base, we will also need to perform horizontal scaling when the user base increases. This will involve adding additional servers and databases to the system, in order to support the increased workload. By using horizontal scaling, we will be able to ensure that the system remains performant and reliable, even as the user base grows.

Finally, we plan to use Apache JMeter to test the system more extensively, and to expand the system to support mobile app development. This will allow us to test the system under a wider range of conditions and scenarios, and will help to ensure that the system is robust and reliable. By expanding the system to support mobile app development, we will be able to provide a more convenient and user-friendly experience for patients and healthcare providers.

Overall, these further developments will help to expand and improve the original project, and will enable us to provide a more comprehensive and effective solution for hospitals and patients. By implementing these developments, we will be able to better meet the needs of our users, and will be able to provide a more

robust and reliable system.

Bibliography

[1] A. F. Arham *et al.*, "Participatory: Stakeholder's engagement toward dengue control techniques in klang valley, malaysia," *SAGE Open*, vol. 11, no. 1, p. 2 158 244 020 982 605, 2021.

[2] W. N. F. W. Yahya and N. C. Dom, "Profile of aedes breeding in urban high-rise buildings: A case study of dengue outbreak in malacca, malaysia," *Planning Malaysia*, vol. 18, 2020.

[3] M. A. F. Mahmud *et al.*, "Environmental management for dengue control: A systematic review protocol," *BMJ open*, vol. 9, no. 5, e026101, 2019.

[4] Sggp and Q. H. .-. T. by Anh Quan, *Dengue fever in ho chi minh city prompts red alerts*, May 2022. [Online]. Available: https://www.sggpnews.org.vn/health/dengue-fever-in-ho-chi-minh-city-prompts-red-alerts-99287.html.

[5] *Dengue increase likely during rainy season, ministry of health, who warn.* [Online]. Available: https://www.who.int/vietnam/news/detail/16-07-2019-dengue-increase-likely-during-rainy-season-ministry-of-health-who-warn.

[6] *Internet of medical things (iomt) market size, share and covid-19 impact analysis*, Oct. 2021. [Online]. Available: https://www.fortunebusinessinsights.com/industry-reports/internet-of-medical-things-iomt-market-101844.

[7] F. Qureshi and S. Krishnan, "Wearable hardware design for the internet of medical things (iomt)," *Sensors*, vol. 18, no. 11, p. 3812, 2018.

[8] N. Petrović and Đ. Kocić, "Iot-based system for covid-19 indoor safety monitoring," *IcETRAN Belgrade*, 2020.

[9] R. Lalitha, G. Hariharan, N. Lokesh, *et al.*, "Tracking the covid zones through geo-fencing technique," *International Journal of Pervasive Computing and Communications*, 2020.

[10] S. Nubenthan and C. Shalomy, "A wireless continuous patient monitoring system for dengue: Wi-mon," in *2017 6th National Conference on Technology and Management (NCTM)*, IEEE, 2017, pp. 23–27.

[11] Q. H. Nguyen *et al.*, "Developing an architecture for iot interoperability in healthcare: A case study of real-time spo2 signal monitoring and analysis," in *2020 IEEE International Conference on Big Data (Big Data)*, IEEE, 2020, pp. 3394–3403.

[12] *Apache kafka*. [Online]. Available: http://kafka.apache.org/ (visited on 08/30/2022).

[13] *Micro-controller esp8266*. [Online]. Available: https://www.espressif.com/en/products/socs/esp8266 (visited on 08/30/2022).

[14] *Functional block diagram of esp8266*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf (visited on 08/30/2022).

[15] *Mlx90614 temperature sensor*. [Online]. Available: https://www.melexis.com/en/product/mlx90614/digital-plug-play-infrared-thermometer-to-can (visited on 08/30/2022).

[16] *Max30100 heart rate sensor*. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf (visited on 11/22/2022).

[17] *Oled ssd1306*. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf (visited on 11/23/2022).

[18] *I2c commmunication standard*. [Online]. Available: https://training.ti.com/sites/default/files/docs/basics-of-i2c-the-i2c-protocol-presentation.pdf.

[19] *Spi commmunication standard*. [Online]. Available: https://training.ti.com/sites/default/files/docs/adcs-spi-communication-basics-presentation.pdf.

[20] *Advanced role-based access control (rbac) for iot devices and applications*. [Online]. Available: https://thingsboard.io/docs/pe/user-guide/rbac/.

[21] S. Rajapakse, C. Rodrigo, and A. Rajapakse, "Treatment of dengue fever," *Infection and drug resistance*, vol. 5, p. 103, 2012.

[22] *Patterns for building javascript websites in 2022*, Jun. 2022. [Online]. Available: https://dev.to/this-is-learning/patterns-for-building-javascript-websites-in-2022-5a93.