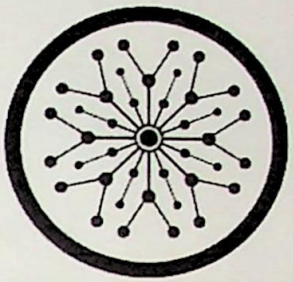
Cloud Based Patient Health Monitoring Using IoT



SUPERIOR UNIVERSITY

A Thesis submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING.

Submitted By
Murtaza Anjum
SU92-MSEEW-F23-002

Session: 2023-25

Supervised By
Dr. Mustafa Shakir

DEPARTMENT OF ELECTRICAL ENGINEERING
FACULTY OF ENGINEERING & TECHNOLOGY
SUPERIOR UNIVERSITY
LAHORE, PAKISTAN

AUTHOR'S DECLARATION

I Murtaza Anjum a student of "**Master**" (MS Electrical Engineering), at "**Faculty of Science**" (Department of Electrical Engineering), the Superior University, Lahore, hereby declare that this thesis titled, **"Cloud-Based Patient Health Monitoring Using IoT"** is my own research work and has not been submitted, published, or printed elsewhere in Pakistan or abroad. Additionally, I will not use this thesis to obtain any degree other than the one stated above.

I fully understand that if my statement is found to be incorrect at any stage, including after the award of the degree, the university has the right to revoke my MS Electrical

Signature of Student: _____

Name of Student: Murtaza Anjum

Roll Number: SU92-MSEEW-F23-002

Date: 1/17/2026

PLAGIARISM UNDERTAKING

I solemnly declare that the research work presented in this thesis titled, "**Cloud-Based Patient Health Monitoring Using IoT**" is solely my research work, and that the entire thesis has been completed by me, with no significant contribution from any other person or institution. Any small contribution, wherever taken, has been duly acknowledged.

I understand the zero-tolerance policy of the HEC and Superior University, Lahore, towards plagiarism. Therefore, I, as the author of the above-titled thesis, declare that no portion of my thesis has been plagiarized and that all material used from other sources has been properly acknowledged, cited, and referenced.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis, even after the award of MS Electrical degree, the University reserves the right to revoke my degree, and that HEC and the University have the right to publish my name on the HEC/University website for submitting a plagiarized thesis.

Murtaza

Signature of Student: _____

Name of Student: Murtaza Anjum

Signature Supervisor: _____

Supervisor Name: Dr. Mustafa Shakir.

CERTIFICATE OF RESEARCH COMPLETION

It is certified that the research work contained in this thesis, titled "**Cloud-Based Patient Health Monitoring Using IoT**" was conducted by **"Murtaza Anjum"**under the supervision of "**Dr. Mustafa Shakir**" and is adequate for the award of "**MS Electrical Engineering**"

No part of this thesis has been submitted anywhere else to any other degree. This thesis is submitted to the "**Faculty of Engineering & Technology**", The Superior University, Lahore in partial fulfillment of the requirements for the degree.

Supervisor's Signature: _____

Supervisor's Name: Dr. Mustafa Shakir

Designation: Professor (Department of Electrical Department)

CERTIFICATE OF APPROVAL

It is certified that the research work contained in this thesis, titled "**Cloud-Based Patient Health Monitoring Using IoT**" was conducted by "Murtaza Anjum"under the supervision of "Dr. Mustafa Shakir" and is adequate for the award of "Degree Name" Master of Electrical Engineering.

No part of this thesis has been submitted anywhere else to any other degree. This thesis is submitted to the "Faculty of Engineering & Technology", The Superior University, Lahore in partial fulfillment of the requirements for the degree.

Student Name: Murtaza Anjum Signature: Murtaza

Examination Committee:

a) **Session Chair**: Dr. Muhammad Zafrullah Signature: _____

b) **External Examiner**

Name: Dr. Haroon Farooq Signature: _____
Designation: Associate Professor
University Name: UET, Lhr (RCET, Grw)

c) **Internal Examiner**

Name: Dr. M.Owais Tariq Signature: _____
Designation: Assistant Professor
University Name: Superior University Lahore

d) **Supervisor Name:** Dr. Mustafa Shakir Signature: _____

e) **Head of Department:** Dr. Manzoor Ellahi Signature: _____

f) **Name of the Dean:** Dr. Muhammad Zafrullah Signature: _____

g) **COE:** Dr. Muhammad Haris Signature: _____

DEDICATION

I would like to express my sincere gratitude to all those who have supported and guided me throughout the course of my research and thesis work. First and foremost, I would like to thank my supervisor, Dr. Mustafa Shakir, for their unwavering guidance, expertise, and valuable feedback that helped shape the direction of this research. I am also deeply grateful to my family and friends for their constant encouragement and understanding during this journey. Their support has been an essential source of motivation.

Additionally, I would like to acknowledge the support from Superior University, whose resources and facilities were invaluable in completing this project. My heartfelt thanks also go to all the researchers and professionals whose work has been referenced and whose contributions to the field of IoT and healthcare have inspired this study.

Thank you,
Murtaza Anjum

ACKNOWLEDGEMENTS

All praise and thanks are due to Almighty Allah for granting me the ability to complete this research work.

I express my sincere gratitude to my supervisor, Dr. Mustafa Shakir, for his guidance, support, and valuable suggestions throughout this research. I am also thankful to the faculty of the Department of Electrical Engineering, The Superior College, Lahore, for their academic support and cooperation.

I am deeply grateful to my parents for their unconditional love, prayers, and constant encouragement, which served as a source of motivation throughout my studies.

PUBLICATIONS LIST

It is certified that the following publication(s) have been made of the research work that has been carried out for this thesis

Journal Article:

Murtaza Anjum, Wasif Arshad, and Mustafa Shakir, "*Design and Implementation of a Cloud-Based Remote Patient Monitoring*"*Research Journal for Social Affairs*Vol. 3 No. 5 (2025): DOI: *https://doi.org/10.71317/RJSA.003.05.0323*

Conference Paper

Wasif Arshad, Mustafa Shakir, Murtaza Anjum, Muhammad Faiq, and Zafarullah khan "Evaluation of SARIMAX and LSTM for continuous glucose monitoring data prediction," submitted to *Proc. 11th Int. Conf. Eng. Emerging Technol. (ICEET 2025)*, Kuala Lumpu, Malaysia, Oct. 22–23, 2025. Submitted Aug. 30, 2025.

TABLE OF CONTENTS

AUTHOR'S DECLARATION VII

PLAGIARISM UNDERTAKING IVII

CERTIFICATE OF RESEARCH COMPLETION IIVII

CERTIFICATE OF APPROVAL IV

DEDICATION V

ACKNOWLEDGEMENTS VI

PUBLICATIONS LIST VII

LIST OF TABLES XV

LIST OF FIGURES XVI

ABSTRACT 1

CHAPTER 1 2

INTRODUCTION 2

1.1 Background and Motivation 2

1.2 Problem Statement 3

1.3 Objectives of the Study 4

1.3.1 To enhance real time processing of data 5

1.3.2 To Reduce Delay in Data Transmission 5

1.3.3 To Improve Scalability of Cloudiofficoma healthcare systems 5

1.3.4 To Enhance the Monitoring System with Reliability and Availability ... 6

1.3.5 Safely and Scalably Delivering Healthcare. 6

1.4 Scope and Limitations 6

1.4.1 Scope of the Study 6

1.4.2 Limitations of the Study 7

1.4.3 Data Security and Privacy 7

1.4.4 The Interoperability Problems. 7

1.5 Research Questions / Hypotheses 9
1.5.1 Research Questions 9
1.5.2 Hypotheses Aligned with Sustainable Development Goals (SDGs): ... 10
1.5.3 Alignment with Objectives 11
1.5.4 Objective Function – Goal: Minimize Resource Usage While Maximizing System Performance 11
1.6 Significance of the Study 12
1.6.1 Technological Advancement 13
1.6.2 Medical Impact 13
1.6.3 Financial and Social Language Influence 13
1.6.4 Academic Contribution 13
1.7 Main Contributions of the Study 14
1.7.1 ESP32 Real-time Patient Health Monitoring System Development 14
1.7.2 Use of MQTT Protocol to have an efficient Communication. 14
1.7.3 AWS IoT Core and AWS Lamda functions integration 14
1.7.4 Alert Mechanism in the Real Time based on AWS Lambda and SNS. 14
1.7.5 Visualization and Data storage on cloud. 15
CHAPTER 2 16
LITERATURE REVIEW 16
2.1 Overview of Patient Health Monitoring Systems 17
2.2 Internet of Things (IoT) in Healthcare 18
2.3 Cloud Computing in Healthcare 20
2.4 AWS in Healthcare Applications 20
2.5 Comparative Study of Existing Systems 22
2.6 Gaps in Current Research 25
CHAPTER 3 29
SYSTEM ARCHITECTURE AND DESIGN 29

3.1 Overview of the Proposed System 29
3.1.1 System Objectives 29
3.1.2 Sensor and Device Integration 30
3.1.3 Cloud Communication and Data Flow 30
3.1.4 Real-Time Alerting Mechanism 30
3.1.5 Data Storage and Visualization 31
3.2 Hardware Components 31
3.2.1 Development Methodology Overview 31
3.2.2 Structural Overview 33
3.2.3 Selected IoT Sensors and Modules 35
3.2.4 Pulse Oximeter and Heart Rate (BME1300) 36
3.2.5 (Body Temperature) 36
3.2.6 Respiration Rate and Environmental Sensor 37
3.2.7 Microcontroller Selection Justification 38
3.2.8 Hardware modules for implementing cloud-based system management 38
3.2.9 Key Reasons for Selecting ESP-32 Devkit 1 or ESP-32S 43
3.2.10 Integration with IoT 44
3.3 Software Components (IoT Platforms, Cloud Services, etc.) 48
3.3.1 IoT Platform: AWS IoT Core 49
3.3.2 Data Processing: AWS Lambda 49
3.3.3 Data storage Amazon DynamoDB 50
3.3.4 Notification Services: Amazon SNS 50
3.3.5 Authorization and User Interface. 50
3.3.6 Monitoring and Logging: AWS Cloud Watch 51
3.4 Network Architecture and Data Flow 51
3.4.1 MQTT Protocol on Data transmission. 52

3.4.2 Secure Connection via secrets.h File 52
3.4.3 Wi-Fi Access and Network Protection 53
3.4.4 Conversion of Sensor Data to JSON Format 54
3.4.5 Data Publishing to AWS IoT Core 54
3.4.6 Data Processing with AWS Kinesis Firehose 55
3.4.7 Scalable Storage with Amazon S3 55
3.4.8 Obtaining Data with Amazon Athena. 55
3.4.9 Visualization in Amazon Quick Sight. 55
3.4.10 Access Control with AWS IAM 56
3.5 Generalized Operation Flow 56
3.5.1 Generalized System Operation Flow 57
CHAPTER 4 61
IMPLEMENTATION – HARDWARE SETUP AND CONFIGURATION 61
4.1 Overview of Hardware Implementation 61
4.2 Components Used 61
4.3 ESP32 Microcontroller Configuration 62
4.3.1 Pin Mapping 62
4.3.2 Sensor Interfacing and Wiring 62
4.3.3 Temperature Sensor 62
4.3.4 Pulse Oximeter Module 63
4.3.5 Environmental Sensor (Respiration Estimation) 63
4.3.6 Pull-up Resistors for I2C Lines 63
4.3.7 Power Supply and Safety Considerations 64
4.3.8 PCB/Breadboard Assembly 64
4.4 IoT Device Integration (e.g., ESP32) 64
4.4.1 Overview of ESP32 as an IoT Device 64
4.4.2 Role in the System 65

4.4.3 Communication Interfaces 65
4.4.4 I2C Bus Sharing and Optimization 66
4.4.5 Bluetooth UART for 66
4.4.6 Wi-Fi Configuration and MQTT Publishing 66
4.4.7 Error Handling and Reliability 67
4.4.8 Power Management and Sleep Modes 67
4.5 Cloud Configuration (AWS IoT Core, S3, Lambda, DynamoDB, etc.) 67
4.5.1 AWS IoT Core Setup 68
4.5.2 AWS Lambda for Real-Time Processing 68
4.5.3 Amazon DynamoDB for Structured Storage 69
4.6 AWS IoT Rule Engine 69
4.6.1 SQL Rule Example: 69
4.6.2 AWS IAM (Identity and Access Management) 70
4.6.3 Optional: AWS SNS for Alerts (Future Work) 70
4.7 Real-time Data Transmission and Storage 70
4.7.1 MQTT Protocol for Lightweight Communication 71
4.7.2 JSON Payload Formatting 71
4.7.3 Real-Time Data Flow Pipeline 72
4.8 Real-Time Storage in DynamoDB 72
4.8.1 Key Features: 72
4.8.2 Archival Storage in S3 73
4.8.3 Timestamp Synchronization and Data Accuracy 73
4.8.4 Error Handling and Data Redundancy 74
4.9 Dashboard/User Interface Design (e.g., React.js, AWS Amplify) 74
4.9.1 React.js Web Frontend Development 74
4.9.2 Main Dashboard Sections: 75
4.9.3 Integration with AWS Amplify 75

4.9.4 Real-Time Visualization with Recharts 76
4.9.5 Role-Based Access (Doctor vs Patient) 76
4.9.6 API Integration for Data Access 76
4.10 Deployment Workflow: 77
4.10.2 Hardware Validation 78
4.10.3 Software Testing 78
4.10.4 Cloud Service Validation 79
4.10.5 Real-Time Transmission Testing 79
4.10.6 Front end and Web Interface testing. 79
4.10.7 Security and Access Control Validation 80
4.11 Mock Data Simulation Results 80
4.11.1 Simulated Body Temperature 81
4.11.2 Simulated Pulse and SpO_2 (Pulse Oximeter) 83
4.11.3 Simulated Respiratory Air Quality (IAQ) Index 86
4.12 AWS Console Setup for Alert Generation 89
4.12.1 Create an Amazon SNS Topic for Notifications 89
4.12.2 Develop and Deploy AWS Lambda Function 90
4.12.3 Lambda Function Code (Python Example): 90
4.12.4 Configure AWS IoT Rule to Trigger Lambda 90
4.12.5 System Workflow 91
4.12.6 Best Practices 91
CHAPTER 5 93
RESULTS, DISCUSSION, AND SYSTEM VALIDATION 93
5.1 Introduction 93
5.2 Real-Time Data Transmission via MQTT 93
5.3 Cloud Storage and Processing Results 94
5.4 AWS Lambda Function Findings and Treatment. 94

5.5 Real-Time Dashboard Visualization 96
5.6 Alert Generation through AWS SNS 97
5.7 Authentication and Security Results (AWS Cognito + AES) 100
5.8 System Testing and Validation 101
5.9 Discussion of Findings 102
5.10 Validation and Observations 102
CHAPTER 6 104
CONCLUSION AND FUTURE WORK 104
6.1 Introduction 104
6.2 Summary of the Research Work 104
6.3 Key Contributions 105
6.4 Limitations of the Research 106
6.5 Recommendations for Improvement 106
6.6 Future Research Directions 107
6.7 Final Conclusion 108
REFERENCES 109
Appendix A 109

LIST OF TABLES

Table 1.1 Summary Table of Limitations in Cloud-Based Patient Health Monitoring Systems 9

Table 1.2 Alignment of Research Objectives with Questions and Hypotheses.....10

Table 2.1 Focus Area Drawbacks and Research Gaps 25

Table 3.1 Selected IoT Sensors and Modules for Vital Signs Monitoring..... 35

Table 3.2 Comparison of IoT Development Boards for Cloud-Based Patient Monitoring Systems 41

Table 4.1 Hardware Components Used in the System Implementation.....61

Table 4.2 ESP32 Pin Mapping for Sensor Integration 62

Table 4.3 Communication Interfaces between Sensors and ESP32.....65

Table 4.4 DynamoDB Table Structure for Patient Vital Data 73

Table 4.5 AWS Amplify Services Used in the System 75

Table 4.6 API Endpoints for Patient Data Access 77

Table 4.7 Typical Results from IAQ Simulation 86

Table 4.8 System Workflow for Patient Monitoring and Alerting 91

LIST OF FIGURES

Figure 3.1 Main concept of the developed cloud-based patient monitoring system... 33

Figure 3.2 Structural scheme of cloud-based patient monitoring 34

Figure 3.3 Pulse Oximeter and Heart Rate Transducer..... 36

Figure 3.4 Body Temperature Sensor..... 37

Figure 3.5 Respiration Rate and Environmental Sensor 37

Figure 3.6 (a) ESP-32 DevKit 1 Node MCU..... 39

Figure 3.7 (b) ESP-32S ESP-WROOM-32 40

Figure 3.8 (c) Raspberry Pi Pico..... 41

Figure 3.9 A Fragment of the secrets file 53

Figure 3.10 Generalized operation algorithm of the developed 56

Figure 4.1 Simulated MAX 30205 Temperature Readings Body Temperature 82

Figure 4.2 Simulated SpO_2 Reading from Pulse Oximeter.....84

Figure 4.3 Simulated Respiratory Air Quality (IAQ) Index.....87

Figure 4.4 AWS Console Setup for Alert Generation.....89

Figure 4.5 Configure AWS IoT Rule to Trigger Lambda 92

Figure 5.1 AWS IoT Core MQTT Test Console showing incoming sensor data.....93

Figure 5.2 AWS IoT Core – Device Certificate Activation for Secure MQTT Communication 94

Figure 5.3 AWS Lambda code editor showing the deployed function. 95

Figure 5.4 Online dashboard screenshot showing real-time and SpO_2 levels. 96

Figure 5.5 Online dashboard screenshot showing real-indoor Air Quality Index. 97

Figure 5.6 Online dashboard screenshot showing real- Body Temperature.....97

Figure 5.7 SNS alert dashboard of AWS lambda 98

Figure 5.8 AWS SNS Topic and Subscriptions for email alert. 98

Figure 5.9 AWS SNS Topic and Subscriptions for SMS alert. 99

Figure 5.10 SMS alert notification with abnormal patient data. 99

Figure 5.11 Email alert notification with abnormal patient data. 100

Figure 5.12 AWS Cognito User Pool with registered users. 101

Figure 5.13 AES encrypted payload sample before transmission.....101

ABSTRACT

This dissertation presents the design and implementation of a cloud-based patient health monitoring system that leverages Internet of Things (IoT) technologies and Amazon Web Services (AWS) to enhance healthcare delivery and chronic disease management. The proposed system enables real-time, remote monitoring of vital health parameters—such as heart rate, blood pressure, oxygen saturation, and body temperature—through IoT-enabled medical sensors. The networked devices store patient data safely in the Amazon Web Services (AWS) cloud through which the data is collected, processed and analysed with scalable service and machine learning capabilities. Such an arrangement will allow medical staff to access real-time patient data that will simplify the process of identifying abnormalities, provide quick medical response, and minimize the chances of rehospitalization.

The given research paper outlines the design and implementation of a cloud-based patient health monitoring that uses the Internet of Things (IoT) principles and Amazon Web Services (AWS) to facilitate the provision of healthcare through the use of the Internet. The given architecture provides solutions to the need of constant health monitoring in geographically distant and underserved zones when the available facilities of traditional medical services are limited. In line with the goals of Society 5.0 and Medicine 4.0, the system will assist in personalized and interconnected and data-driven care. It also incorporates sensors in the biomedical domain, the use of microcontrollers, and the AWS services, including AWS IoT core to provide the connection of the devices, AWS S3 to guarantee the storage of the data, and AWS Lambda to process the information in real time, thus making sure that the vital health parameters are properly acquired and analyzed. The system will increase efficiency and responsiveness in medical practice by applying a patient-centred approach, which is proactive and especially relevant in managing chronic conditions and taking care of elderly patients. The prototype of the system was built and tested via simulation and proved the integrity of its operations, scalability, as well as compatibility with modern digital healthcare infrastructures.

CHAPTER 1

INTRODUCTION

The chapter provides an in-depth account of contextual background related to Internet of Things (IoT)-enabled healthcare systems, especially remote patient monitoring (RPM) in hybrid edge cloud-based systems. It also outlines the main objectives of the dissertation, lists the main contributions, and demonstrates the system architecture and technical architecture chosen in general, to be implemented.

1.1 Background and Motivation

The high rate of change of the Internet of Things (IoT) has sparked tremendous changes in a myriad of industries, and healthcare is partly of them. Another of the most useful IoT implementation in the same sphere is the creation of electronic computing clouds to monitor the health of patients continuously. Individualized networked sensors of vital signs, such as heart rate, blood pressure, oxygen saturation, and glucose levels are made easily accessible through such systems at the time a patient is attended to. These technologies are particularly beneficial in chronic disease treatment like cardiovascular disease, diabetes mellitus and lung diseases which keep increasing across the globe [1].

Conventional methods in healthcare are often based on intermittent hospital sessions and paper documentation that may create delays in service delivery of diagnostic and treatment services, especially to the citizens of isolated or underserved communities. The World Health Organization (WHO) cite various reasons which explain the escalating prevalence of chronic diseases in the world due to the demographic ageing as well as limited access to normal healthcare services [2]. In order to overcome these difficulties, the IoT infrastructures with cloud computing would be more efficient, continuous, and proactive model of care. These devices use low-power microcontrollers like ESP32 to receive and send patient data to cloud services like the Amazon Web Services (AWS) so that the data can be safely stored, processed, and presented in real-time [3].

It has been shown that a hybrid architecture based on edge and cloud computing is specifically useful to healthcare monitoring. In this construction, local processing of urgent data, e.g. the detection of anomalous readings or the activation of alerts, is done by the microcontroller to reduce latency, but less time-sensitive data is sent to the cloud

to perform in-depth analytics, long-term storage, and professional analysis [4]. This 2-layered topology is useful in case of emergency and scalability, unreliability, and secure access across different sites are maintained at the same time [5].

Supported by the services of AWS IoT Core, AWS Lambda, Amazon S3 and Amazon SNS, the cloud component deals with constantly increasing working data, automated event recognition and immediate notification. As an example, with a vital sign of an over- or under-set safe range, AWS Lambda might send an alert to physicians or caregivers using SNS (via SMS, email, a mobile application) which will help to take timely measures regardless of the geographical location of the patient [6].

However, with the advent of these technologies, there is the large issue of privacy and data security. The safety of the passage and storage of confidential health data are of the utmost concern. In response to this, modern systems engage in strong encryption, validated access, and adherence to the global privacy regulation systems like the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR) [7]. Developed cryptographic algorithms (such as homomorphic encryption) are also under research to promote secrecy, but leave the usefulness of data intact to be used as an analysis tool [8].

This study also suggests a patient monitoring system that is based on cloud computing, IoT and edge computing to help provide a secure, scalable and reliable healthcare solution. With the real-time nature of the ESP32 microcontroller and the flexible nature of AWS cloud capabilities, the system aims at reducing the number of hospital visits, enhancing the management of chronic diseases and increasing the overall access to healthcare- especially to people in rural or resource constrained environments. The proposed system complies with the norms of Medicine 4.0, Society 5.0 and helps to create a more intelligent, inclusive and patient-centred care model.

1.2 Problem Statement

The intertwining of the Internet of Things (IoT) and cloud computing has spawned a great potential in the healthcare sector, and it has become possible to track patients in real-time, provide predictive analytics, and remotely supply them with medical services. However, the implementation of the cloud-based IoT systems in the critical healthcare environments is faced by a variety of issues that hinder their effectiveness

and slow their adoption.To begin with, the issue of the data security and confidentiality is primary. Sensitive patient data that is shared and exchanged among interconnected devices and decentralized cloud environments are prone to cyber-attacks, information breakages, and unauthorized access. Adherence to rules and regulations like the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR) also brings another layer of complexity especially in systems that cut across jurisdictions.

Second, the problem of latency and reliance on network connectivity become rather dangerous in time-sensitive medical conditions. Delays in transmitting data or response time by the cloud are some of the factors that might be at the expense of patient safety in distant or resource constrained areas where immediate action may be hard to take at a point when a patient started adopted abnormal health patterns.

Third, the density and the topics of scalability of cloud-based systems are troubled by the fast growth of IoT devices and unstopping production of high-frequency health data. This situation may cause inefficiencies in data processing, high cost of operations and slow system response time particularly in cases where real time analysis is needed.

Fourth, fault tolerance and reliability are of importance to healthcare applications. Malfunction of devices, lack of cloud services, loss of data may cause severe issues with the service and incorrect evaluation of the patient.

However, A system of full-effectiveness, security, scalability, and real-time monitoring of patient health has not been found yet despite the development of the IoT and cloud technologies. The need to implement a hybrid strategy, combining edge computing, which ensures the possibility of local processing, with cloud computing, which provides its products with scalability, and forms a solid security framework to provide quality patient-related data surveillance in a reliable, timely, and confidential manner is quite evident. To achieve the quality, access, and safety of digital healthcare, these difficulties should be overcome.

1.3 Objectives of the Study

The main aim of the study is to develop and deploy a patient health monitoring system based on cloud computing and Internet of Things (IoT) to enhance healthcare services by improving efficiency, responsiveness, scalability, and reliability. The study

addresses the limitations of traditional monitoring systems and enables continuous, proactive, and remote care by integrating real-time data collection, cloud analytics, and intelligent edge processing.

1.3.1 To enhance real time data processing

- Deploy edge and fog computing for local data processing, reducing dependency on cloud infrastructure and improving responsiveness.
- Implement data compression and filtering to minimize IoT data transmission while maintaining accuracy and integrity.
- Increase the efficiency of IoT-based medical devices in terms of both computation and resource consumption via the development of energy-aware and resource efficient data management algorithms.

1.3.2 To Reduce Delay in Data Transmission

- Employ low-latency communication frameworks such as MQTT and CoAP to ensure prompt and efficient data delivery.
- Utilize cloud-native technologies such as containerization and micro services to enable scalable deployment and integration of healthcare applications.
- Use predictive modelling to detect and eliminate network congestion or bandwidth constraints latency problems.

1.3.3 To Improve Scalability of healthcare systems.

- Develop an elastic cloud system capable of accommodating the increasing volume of IoT healthcare data.
- Introduce cloud-native technology, such as containerization and micro services, to allow the deployment, integration and maintenance of healthcare apps to scale.
- Standards Draft standardized APIs and protocols that will provide interoperability between various IoT devices and known cloud environments.

1.3.4 To Enhance System Reliability and Availability.

- Provide uninterrupted access to patient information through fault-tolerant, self-healing, and failover-capable architectures.
- Improve IoT sensor reliability using predictive maintenance to prevent device or measurement failures
- At the cloud level, deploy high-availability (HA) and disaster recovery (DR) support that the cloud infrastructure will maintain the availability and safety of the system and data in case of service disruptions.

1.3.5 Safely and Salable Delivering Healthcare.

- Build a secure cloud-IoT system compliant with healthcare regulations such as HIPAA and GDPR to protect patient privacy and data security.
- Assist with remote, data-driven, healthcare professional decision-making with an interactive, of dashboard, including real-time warning, visualization, and trend analysis.

1.4 Scope and Limitations

1.4.1 Scope of the Study

- **IoT-Enabled and Energy-Efficient Data Collection (SDG 3, 9, and 12):** The system employs biomedical sensors integrated with low-power microcontrollers (e.g., ESP32) for continuous and energy-efficient monitoring of vital signs such as heart rate, body temperature, blood pressure, and oxygen saturation. This promotes sustainable use of electronic health devices.
- **Green Cloud Infrastructure via AWS (SDG9 and 13):** Amazon Web Services (AWS) is used to provide scalable and on-demand cloud resources (IoT Core, Lambda, S3), minimizing physical hardware requirements and reducing carbon footprints through optimized resource utilization and serverless computing.
- **Real-Time Health Monitoring and Early Warning Systems (SDG 3):** This system can specify the healthcare responsiveness by sending real-time notification via AWS Simple Notification Service (SNS) when unusual health

indicators are detected to facilitate preventive healthcare and lessen hospitalization.

- **Predictive Analytics to Preventive Health and Resource Planning (SDG 9 and 12):** SARIMAX predictive models are lightweight tools deployed to analyse health trends and forecast anomalies to make interventions promptly and facilitate the planning of healthcare resources effectively.
- **The necessary EEs are inclusive and Accessible User Interfaces (SDG 4, 5, and 10):** A web-based dashboard is created to be responsive and ensure inclusivity and friendliness among healthcare professionals and caregivers to meet digital literacy requirements and enhance gender and age inclusivity in digital health.
- **Reducing Health Inequality (SDG 3 and 10):** Catering to the long-term monitoring of vulnerable populations in the country such as the elderly, chronically ill, and rural or hard-to-reach regions will in turn support health equity alongside decreasing the gap in access to care.

1.4.2 Limitations of the Study

Although there are chances of the proposed system being used, the system has a number of limitations that can limit its applicability and scalability in the real world. These limitations are overcome as follows:

1.4.3 Data Security and Privacy

To handle any sensitive health-related data, the utmost level of security measures and rigidity should be applied in compliance with the particular regulations, e.g., HIPAA and GDPR. End-to-end encryptions, robust authentication and data access control on a granular level is an urgency to avoid breach and unauthorized access to data. Nevertheless, there is an overwhelming demand of adding holistic security capabilities to the resource constrained IoT devices [9].

1.4.4 The Interoperability Problems.

The lack of standard data format and communication protocols used by heterogeneous IoT devices decreases interoperability. This limitation impairs the seamless

incorporation with the current healthcare systems and could prevent data transfer between platforms [10].

1.4.4.1 Belatedness and Unconnection.

The process of real-time monitoring is based on stable and fast internet connectivity. High latency and lost packets in remote/under-developed countries with poor connectivity can slow down or interrupt the health information transmission thus affecting the responsiveness of the system [11].

1.4.4.2 User Acceptance and Usability

There is also a factor of acceptance by both patients and healthcare professionals in determining the success of the system. A user with a low level of digital literacy or unwillingness to embrace new technology can have challenges with usability that can reduce patient engagement and compliance [12].

1.4.4.3 Resource Constraints of IoT Devices

IoT devices traditionally have limitability in computer processing power, memory content and power. These restrictions inhibit the use of complicated algorithms and continuous surveillance, particularly in environments that have an energy cost [13].

1.4.4.4 Capability Limitations

With the increased quantity of connected devices and the quantity of data it produces, maintaining the performance of the system and achieving scalability with a reasonably low cost becomes an even more complex problem. The cloud infrastructure should be tuned to manage large scale implementation besides maintaining reliability [14].

1.4.4.5 Summary Table of Limitations

Table 1.1Summary Table of Limitations in Cloud-Based Patient Health Monitoring Systems

Limitation	Description	Impact	Reference
Data Security and Privacy	Risk of unauthorized access to sensitive health data	May lead to non-compliance with privacy laws	[9]
Interoperability	Incompatibility among different IoT devices and health systems	Limits integration and data sharing	[10]
Latency and Connectivity	Network delays in rural/remote areas	Affects timely monitoring and alert delivery	[11]
User Acceptance and Usability	Lack of digital literacy or resistance to technology	Reduces user engagement and system effectiveness	[12]
Resource Constraints	Low processing capacity and power limitations in IoT devices	May disrupt continuous and real-time monitoring	[13]
Scalability	Difficulty handling large-scale deployments and data growth	Impacts system performance and operational cost	[14]

1.5 Research Questions / Hypotheses

To address the inefficiencies, latency, scalability, and reliability issues identified in cloud-based IoT healthcare systems, this study is guided by the following research questions and hypotheses

1.5.1 Research Questions

RQ1: How can real-time patient health data be efficiently processed and transmitted using a cloud-based IoT system without overburdening device resources?

RQ2: What role can edge and fog computing play in reducing latency and ensuring timely health data transmission in environments with unstable connectivity?

RQ3: How scalable is the proposed cloud infrastructure to be when used in a multi-patient, multi-sensor setup?

RQ4: How can we make systems more reliable by introducing predictive maintenance and fault-tolerant systems in cloud-based monitoring of the Internet of Things in health?

RQ5: How do user experience (UX), and digital literacy affect the adoption and use of the patient monitoring system between healthcare providers and patients?

RQ6: What is the privacy and security threats of the IoT-based collection of patient data and cloud storage, and how it would be possible to overcome them?

1.5.2 Hypotheses Aligned with Sustainable Development Goals (SDGs):

H1-SD (Efficiency & Resource Optimization -SDG 9 and 12):

The combination of edge computing and cloud infrastructure is associated with significant energy savings and consumption of bandwidth, which can become a part of environmentally sustainable digital healthcare solutions.

H2-SD (Equity in Access – SDG 3 & 10):

Equitable access to real-time health facilities in underserved and remote areas is increased by the use of low latency and lightweight communication protocols like MQTT.

H3-SD (Scalability and Economic Sustainability- SDG 8 and 9):

The implementation of cloud-native microservice and elastic infrastructure makes the optimization of the scaling process cost-efficient, which leads to the long-term economic viability of e-health services.

H4-SD (Preventive Maintenance and Waste Reduction -SDG 9 and SDG 12):

By connecting with predictive maintenance based on the AI, electronic waste is minimized and the lifecycle of IoT devices is extended, thereby, promoting sustainable production and consumption processes.

H5-SD (Inclusive Digital Health -SDG 4 and 5):

User-centred design and digital literacy training enhance inclusiveness and acceptance of the system by different group of users especially women and the elderly.

H6-SD (Data Security & Trust – SDG 16):

Intense system data encryption, authentication, and access-control tools will enhance user confidence and instill ethical and resilient digital health systems.

1.5.3 Alignment with Objectives

The objectives of the study are supported by each research question and hypothesis.

Table 1.2 Alignment of Research Objectives with Questions and Hypotheses

Research Objective	Aligned Question	Corresponding Hypothesis
Improve real-time data handling	RQ1	H1
Reduce latency in data transmission	RQ2	H2
Enhance scalability of system infrastructure	RQ3	H3
Improve system and device reliability	RQ4	H4
Increase user engagement and system usability	RQ5	H5
Ensure data security and privacy compliance	RQ6	H6

1.5.4 Objective Function – Goal: Minimize Resource Usage While Maximizing System Performance

Let's define the system performance function as a multi-objective optimization problem.

1.5.4.1 Objective Function:

$$\min J = \alpha 1 \cdot L + \alpha 2 \cdot E + \alpha 3 \cdot C - \beta 1 \cdot A - \beta 2 \cdot R - \beta 3 \cdot S$$

Where:

- **JJJ:** Total system cost function (to be minimized)
- **LLL:** Average system latency (data acquisition → cloud → alert)
- **EEE:** Energy consumption of IoT devices
- **CCC:** Cloud service cost (AWS Lambda, IoT Core, S3, SNS, etc.)
- **AAA:** Availability / uptime of the system

- **RRR:** Responsiveness (rate of successful alerts and dashboard updates)
- **SSS:** Security score (based on encryption strength, access controls)

Constants $\alpha 1$, $\alpha 2$, $\alpha 3 \in R^+$ and $\beta 1$, $\beta 2$, $\beta 3 \in R^+$ are weighting factors that reflect the importance of each metric.

1.5.4.2 Constraints:

1. Latency Threshold:

$L \leq Lmax$

2. Energy Budget:

$E \leq Ebudget$

3. Budget Constraint:

$C \leq Cmax$

4. Availability Threshold:

$A \geq Amin$

5. Security Compliance (e.g., HIPAA/GDPR):

$S \geq Smin$

6. Scalability – Device Handling Capacity:

$N \leq Nmax$ where N is the number of active devices

1.5.4.3 Security Score Example (S)

Let the security score SSS be a weighted combination of components:

$S = w1 \cdot Enc_{score} + w2 \cdot Auth_{score} + w3 \cdot Access_{score}$

Enc_{score}: Encryption protocol strength (e.g., AES-256 = 1.0)

$Auth_{score}$: MFA, Cognito roles

$Access_{score}$: IAM policy granularity

1.6 Significance of the Study

The importance of this research lies in the fact that it contributes to the optimization of the provision of healthcare services through the implementation of Cloud Companies

and Internet of Things (IoT) technologies. The study provides empirically based solutions to vital issues such as real-time monitoring, system latency, scalability, and data reliability, which makes it a valuable resource in ensuring higher quality patient care.

1.6.1 Technological Advancement

The paper introduces a hybrid design which is a combination of edge computing and cloud infrastructure to support collection, processing and analysis of real time data. This synergy enhances less time taken in the system and it is effective in managing the large amount of health information produced by IoT enabled devices. This kind of architecture forms the basis of computing a seamless healthcare service, especially in situations that are limited by scarcity in terms of the number of computer resources.

1.6.2 Medical Impact

Medically, the system allows constant monitoring of patients, hence, aiding in the early identification of the abnormalities in the health conditions. This is particularly beneficial to those people with permanent illnesses, who need to be under close supervision. Remote monitoring improves the quality of diagnosis and reduces the need to visit a hospital often thus improving the overall health outcomes.

1.6.3 Financial and Social Language Influence.

The program provides significant results in non-urban or underserviced areas where remote healthcare is provided by the system. Patients will also receive care at the right timeline without necessarily having to travel long distances, which will enhance health equity. Moreover, reducing the number of hospitalization and decreasing operational costs spent by healthcare providers, the solution facilitates the sustainability of healthcare systems.

1.6.4 Academic Contribution

The study contributes to the academic community because it covers the modern issues in the IoT-based health systems such as data privacy, interoperability, and system scalability. It preconditions a real framework of future research and advancements of telemedicine, eHealth, and intelligent healthcare systems.

1.7 Main Contributions of the Study

This dissertation outlines the design and implementation of a real-time patient health monitoring architecture based on the ESP32 microcontroller, the MQTT communication protocol and the AWS cloud services and AWS Lambda in particular. The system assures the effective procurement, processing, examination, and accessibility of essential health measures through means of the cloud. The major findings of the study are the following:

1.7.1 ESP32 Real-time Patient Health Monitoring System Development.

The experiment resulted in a low-weight, inexpensive, and energy-efficient medical tracking system, which uses an ESP32 microcontroller, together with biomedical sensors (e.g., heart rate, temperature). The ESP32 reads sensor inputs and issues this information through the MQTT protocol it provides thus guaranteeing low-latency and lightweight communication.

1.7.2 Use of MQTT Protocol to have an efficient Communication.

Its setup has been based on the lowfi and lightweight protocol, MQTT, which can be used to send the patient health information of ESP32 to AWS IoT Core. This solution ensures the availability of trusted, fast, and scalable information utilization even under the limited network setup, which makes the system appropriate to rural and isolated healthcare implementation.

1.7.3 AWS IoT Core and AWS Lamda functions integration.

AWS IoT Cores receives patient data sent using the MQTT protocol and runs it in real time using AWS Lambda functions. Such serverless functions can provide the dynamic capability of rule-based processing, such as parsing, filtering and alert notifications without requiring to maintain pre-existing server infrastructure.

1.7.4 Alert Mechanism in the Real Time based on AWS Lambda and SNS.

This is through an automated alert subsystem, which is deployed through AWS Lambda and Amazon Simple Notification Service (SNS). A sort of sensor data are received by the incoming sensor, exceeding predetermined health limits, and Lambda is activated to cause real-time notices through SNS to clinicians or caregivers, thus permitting timely medical attention.

1.7.5 Visualization and Data storage on cloud.

Vital sign data is processed into AWS services (such as DynamoDB or Amazon Timestream) to securely store the data and presented in AWS dashboards (such as Amazon QuickSight or a new web portal). This structure provides the clinicians with a platform in which they monitor patients remotely in real time.

1.7.5.1 Lightweight and Reliable Edge Processing

Though the architecture is mainly based on cloud resources, the ESP32 does some basic edge processing to authenticate sensor values and ensure that nothing but quality data is sent. Such a practice reduces overheads in transmitting data and improves system responsiveness.

1.7.5.2 Scalable System Architecture and Security.

The architecture has TLS encryption and AWS Identity and Access Management (IAM) policy to ensure secure transfer of data and access control. In addition, it also facilitates horizontal scalability by allowing numerous ESP32 units to publish simultaneously without memory loss to make it suitable in large-scale applications to deployed in a clinical or hospital environment.

- Development of a hybrid IoT-cloud architecture for continuous patient health monitoring.
- Demonstration of low-latency communication and efficient data transmission via MQTT and AWS IoT Core.
- Ensuring system scalability, reliability, and availability through cloud-native design and fault-tolerant infrastructure.
- Integration of security and privacy measures aligned with healthcare regulations (HIPAA, GDPR).
- Provision of a framework that supports remote, preventive, and equitable healthcare delivery.

CHAPTER 2

LITERATURE REVIEW

The chapter is a comprehensive overview of the available resources in knowledge on the topic of cloud-based patient health monitoring systems, paying special attention to the overlapping of the Internet of Things (IoT), embedded microcontrollers, specifically the ESP32, cloud solutions, like Amazon Web Services (AWS), lightweight messaging frameworks, like the MQTT, and event-driven serverless computing companies, like AWS Lambda. With the help of the integration of these technologies, it is possible to implement scalable and real-time healthcare applications to assist in the remote monitoring of vital physiological parameters, such as heart rate, body temperature, the level of peripheral oxygen desaturation and glycaemic levels.Empirical research on cloud-based health monitoring systems reveals that in recent years there have been a fast increase in the number of IoT devices, lightweight communication protocols, like the MQTT, and microcontrollers like the ESP 32, which works effectively withaws clouds. These combinations allow to monitor the vital signs of patients in real-time remotely and at the same time give automation and scales. However, as also mentioned in Reference [17], a major issue is not just only related to the technical configuration, but the question is whether or not the users can really believe in the accuracy and reliability of these systems. Control systems and even the latest sensors and platforms despite usage of them are often doubted by both patients and health professionals concerning the authenticity of the information sent to reflect upon the health status of a person without errors or delays.

This evidence highlights the critical importance of the user trust in the acceptance and use of the remote monitoring systems. The hindrance faced by most people is their unwillingness to rely on such systems because of fear of a possible loss of data, sensor failure or burning up of the system with a slow response time. The characteristics of AWS Lambda like serverless processing or the ESP32 like the local data handling are beneficial to the efficiency, but they cannot guarantee the reliability or credibility of the outcomes. It is also argued in reference [17] that this challenge is not just technical but also psychological and behavioural. The concrete success of these systems is highly determined by the comfort of individuals, their confidence and previous experiences.

Finally, a significant knowledge gap in the research of Reference [17] is that expediency and accuracy are not enough; the system should also have transparency and trustworthiness. Future studies must focus on in-time data validation, optimize information flow to patients and clinicians, and be able to perform adequately under shifting networking or environmental circumstances. The inability to cover such human-centred factors could affect even the best of cloud-computing-based monitoring systems and prevent the achievement of their popularity and consistency in the real-life circumstances of healthcare.

2.1 Overview of Patient Health Monitoring Systems

In the past decade the development of patient health monitoring has been unprecedented as traditional, hospital-driven approaches have been replaced with more advanced and technologically oriented ones which enable constant monitoring of patients without the use of medical facilities. Using these innovative systems, essential physiological parameters, such as heart rate, blood pressure, core temperature, and others, are recorded and sent real-time to the clinicians, thus enabling prompt interventions.

The use of IoT-based solutions with complementary cloud computing and wireless communication has significantly enhanced the efficiency and the availability of health monitoring. Precisely, cloud-based systems provide scalable data storage, real-time data storage, and advanced analytics that enhance faster and accurate medical decision-making processes.These systems have become particularly useful to track chronic diseases, post-op recovery management, and elderly care since they can reduce the number of visits to hospitals coupled with untimely detection of possible health problems.

Nevertheless, these systems also experience significant challenges although they are technologically advanced. The degree of patient trust in remote monitoring solutions is rather low, as stated by Nguyen et al. [17]. The concern of most patients is sensor accuracy, equipment breakdowns or remote diagnosis errors. Also, there is the issue of privacy and data protection which still makes extensive use of this not strongly accepted in clinical practice.

In order to eliminate these shortcomings, scientists and technology experts are currently focusing on the increased reliability of systems by providing backup facilities, data

authentication, encryption, and the use of secure authentication by users. Enhancing the reliability and data integrity will help developers to increase the level of patient trust and promote the use of cloud-based healthcare monitoring systems by more people.

2.2 Internet of Things (IoT) in Healthcare

The convergence of the cloud computing and Internet of Things (IoT), technologies has initiated an intensive revolution in the healthcare sector, especially in the context of tracking of the patient health. Modern systems enable the means of remote, continuous monitoring of vital signs and, as a result, provide access to quick patient information. The empirical research shows that when cloud environments like Amazon Web Services (AWS) are combined with IoT-based embedded systems like the ESP32, they may result into cost-effective, scalable, and smart monitoring systems that are applicable to chronic disease management, as well as, the critical care.

The article by [18] confirms that selection of Artificial Intelligence (AI) in the context of the IoT-based healthcare system enhances the processes of decision-making and predictive analytics. In cases where the AI models are run on serverless cloud systems like the AWS Lambda, they will have the capability to identify anomalies in the data of patients at an early stage. This is a convenient means of detecting anomalies, which can be used to minimize hospitalizations and improve patient outcomes. The paper also notes that cloud platforms are able to automatically scale computing resources according to the demand and the data traffic of the system.

Microcontrollers with low-power, like the ESP32, have become the key particles in these constitutions because they have embedded Wi-Fi and Bluetooth potential, which allows easy pairing with biomedical sensors. These devices obtain vital parameters in the physiology, such as heart rate, body temperature, oxygen saturation and glucose levels. The gathered data will then be sent to the cloud through MQTT- a lightweight communication protocol that is very attuned to real-time protocols and low-energy usages [19].

The key component of this architecture is AWS IoT Core that manages the secure two-way communication between the devices and cloud services. It ensures that sensor data are transmitted to and immediately processed on the fly and may be sent to AWS Lambda to be processed on an event basis or stored in Amazon S3 and DynamoDB to

perform later analysis. As it is mentioned in [20], the introduction of 5G networks is expected to contribute to the better performance of such healthcare systems even in rural or remote locations by providing better transmission rates and reduced latency rates.

In addition, edge and cloud computing frameworks are also receiving growing interest in hybrid frameworks. Such models can access data processing device in the preliminary stage of filtering or aggregation with devices like ESP32 or Raspberry Pi and then send it to the cloud. It is a strategy that minimizes bandwidth usage and latency in the system and maximizes responsiveness [21]. This design, combined with the event-driven capabilities of AWS Lambda, will allow generating fast alerts, using resources more efficiently, and scaling without challenges to high workloads.

Although such technological advances exist, the issue of safety and privacy of medical data remain an issue of concern. These issues can be reduced through the blockchain technology because it can create tamper-free records and offer transparent access control as highlighted in [22]. AWS is also blockchain integrable and has stronger encryption capabilities, multi-factor authentication and rigorous data governance practices that are compliant with healthcare laws and regulations including HIPAA and GDPR.

Also, greater implementation of the IoT-cloud relies on the trust of users and compatibility with the system. Many healthcare facilities are faced with difficulties in integrating the IoT data into the already existing Electronic Health Record (EHR) systems, and patients tend to express doubts regarding the accuracy and dependability of continuous monitoring devices [23]. These issues will be necessary in building confidence and enhancing the wider use of these technologies.

Finally, the literature has proved the idea of integrating AWS cloud functionalities, sensor nodes, powered by ESP32, with IoT protocols in terms of the fundamental ability to radically change healthcare monitoring. Next-generation monitoring systems will be smarter, safer, and more universal in various healthcare settings, and it is expected that the merging of AI and machine learning with MQTT and 5G connectivity will make this possible.

2.3 Cloud Computing in Healthcare

Cloud computing has become a central pillar of modern healthcare innovation, mediating the design and implementation of intelligent, cloud-based patient monitoring systems. All these qualities have made remote, real-time monitoring of patients possible, even in hospitals or areas with limited resources, due to its flexibility, scalability, and cost-effectiveness, especially when using services like Amazon Web Services (AWS).

As explained in [24], cloud computing offers benefits such as on-demand computing power, scalable storage, and global connectivity. However, these studies often overlook practical challenges such as integration latency, dependency on stable network infrastructure, and potential bottlenecks when multiple devices simultaneously transmit high-frequency physiological data from IoT-based devices like ESP32. Allied with AWS IoT Core, these sensors monitor physiological parameters such as body temperature, heart rate, and oxygen saturation and transmit the data via MQTT. Although MQTT ensures fast and low-bandwidth data exchange, prior works fail to address scenarios of network congestion, sensor failure, or data packet loss, which may compromise real-time analytics.

Serverless computing with AWS Lambda is highlighted in [25] for automatic responses to critical medical conditions, such as arrhythmia detection, via AWS SNS notifications or event logging in DynamoDB. While this demonstrates scalability and responsiveness, prior studies provide limited evaluation of latency during traffic peaks or reliability under high-volume clinical loads, raising concerns about its dependability in urgent care situations.

Moreover, Amazon S3 provides long-term storage, and AWS SageMaker can implement predictive models to detect early patient deterioration, as shown in [26]. Nevertheless, these studies often assume ideal data quality and ignore issues like incomplete records, sensor noise, or biased historical datasets, which can reduce model accuracy and limit clinical applicability.

Another advantage cited is interoperability, where AWS supports IoT health devices, mobile apps, and EHR systems via secure APIs and AWS Glue, as discussed in [27]. Yet, existing research rarely evaluates integration across heterogeneous hospital

systems or compliance with legacy infrastructure, which may impede seamless implementation in real-world clinical settings.

Patient information protection remains a critical challenge. Although end-to-end encryption, role-based access control, and multi-factor authentication are possible with AWS IAM and KMS, prior works frequently assume flawless adherence to security standards (HIPAA, GDPR) without empirically testing risks from insider threats, misconfigurations, or cross-platform vulnerabilities.

Despite these benefits, adoption is constrained by human factors. Studies in [29] note potential reluctance among healthcare professionals to incorporate cloud-based systems into existing workflows and patient concerns regarding reliability or confidentiality. Yet, the extent of training required, the burden on staff, and the long-term trust-building process remain underexplored in the literature.

Looking forward, solutions such as 5G, edge computing, and AI are expected to enhance cloud-enabled healthcare. For example, [30] projects that AWS Greengrass combined with 5G can enable faster, low-latency data exchange, improving availability in rural areas. Nevertheless, most projections are theoretical, with few empirical evaluations on the cost, scalability, and operational challenges of deploying such advanced infrastructures in resource-constrained healthcare settings.

Moreover, further investigation is needed to assess the energy efficiency and cost-effectiveness of large-scale cloud-IoT deployments. Validation of predictive models across diverse patient populations and comorbidities is still limited. Potential psychological effects on patients, such as stress or alarm fatigue from continuous monitoring, remain underexplored. Additionally, the challenges of vendor lock-in and maintaining interoperability across evolving cloud and IoT systems require careful consideration

In conclusion, while existing studies support cloud computing and AWS services in IoT-based patient monitoring, critical limitations remain unaddressed, including network reliability, real-world scalability, data quality, system integration, and human adoption. Addressing these gaps is essential to ensure safe, scalable, and intelligent healthcare infrastructures that can genuinely improve clinical decision-making and enable proactive, preventive patient care.

2.4 AWS in Healthcare Applications

AWS has become one of the most notable cloud platforms that have been the foundation of innovation in modern healthcare, especially in the application of remote patient monitoring. The combination of the powerful cloud computing environment of AWS and the IoT-like medical devices has inspired the growth of reliable, scalable, and real-time monitoring systems which can maintain continuous monitoring of the patient.

Using the AWS IoT Core service, medical devices (equipped with electronic sensors) and wearables (ECG monitors, thermometers, etc.) can continuously gather physiological data, like heart rate, oxygen saturation, and blood pressure. These metrics are sent to the cloud via the MQTT protocol, which ensures concise and efficient communication. While prior works [31] highlight reduced transmission latency and parallel handling of large amounts of patient data, they often overlook potential network congestion, packet loss, and the practical limits of scaling in high-density clinical settings, which may compromise real-time performance.

AWS Lambda enhances system responsiveness as it listens to incoming data streams in real time. Abnormal vital signs can trigger automatic updates and enable immediate medical intervention without manual control. However, studies [32] rarely examine Lambda's performance under sustained high-load conditions or its reliability in multi-patient deployments, leaving questions about latency and resource contention during peak usage unaddressed.

For long-term data management, Amazon S3 provides secure, redundant, and durable storage of patient records and EHRs. While [33] notes compliance with healthcare regulations and data integrity, these studies assume ideal network conditions and do not evaluate risks from data corruption, migration errors, or interoperability challenges across multiple storage solutions. Meanwhile, Amazon Dynamo DB enables rapid access to patient history and current trends, yet potential limitations regarding query efficiency and consistency across distributed datasets are seldom discussed.

AWS Sage Maker provides predictive modeling for health risk assessment. According to [34], it can detect early pathological signals, such as arrhythmia, sleep apnea, or diabetic complications. Nevertheless, prior studies largely assume high-quality, noise-

free data and do not fully address model bias, generalizability across heterogeneous patient populations, or the impact of missing sensor readings on predictive accuracy.

AWS Green grass extends cloud capabilities to the network edge for local IoT data processing, reducing latency and ensuring consistent performance even with limited connectivity. Reference [35] demonstrates improved reliability, yet empirical evaluations on large-scale edge deployments, energy consumption, and device maintenance in real-world ambulance or rural clinic scenarios remain scarce.

Security is a critical principle in AWS-based healthcare systems. AWS KMS, IAM, and Cloud Trail provide encryption, user control, and audit logging, while AWS Managed Block chain ensures tamper-resistant record tracking [36]. Still, most prior research assumes flawless compliance with standards like HIPAA and GDPR and does not assess threats from human error, insider attacks, or misconfigured permissions, which could compromise system security in practice.

Despite technical sophistication, challenges remain in large-scale implementation. Integration with existing EHRs, staff training, and operational costs are substantial barriers, as discussed in [37]. Beyond these, patient trust and privacy remain underexplored, particularly regarding automated monitoring errors, data misuse fears, and the potential for reduced adherence to recommendations.

Looking ahead, the combination of AWS IoT services with AI and 5G networks promises transformative healthcare delivery. Reference [38] indicates 5G can reduce communication latency and support high-speed real-time diagnostics. Nevertheless, most projections are theoretical, with little empirical evidence on deployment costs, scalability in under-resourced areas, and the regulatory challenges of integrating AI with cloud-based clinical decision systems [39].

In conclusion, AWS is a core enabler of cloud-based healthcare systems, offering IoT, machine learning, analytics, and security tools to develop flexible, scalable, and secure patient monitoring solutions. Yet, critical limitations such as integration complexity, regulatory compliance, network reliability, AI model generalizability, and user trust remain under-addressed and will be key barriers to realizing the full potential of AWS in the healthcare ecosystem.

2.5 Comparative Study of Existing Systems

The recent developments of blending the use of cloud computing and Internet of Things (IoT) technologies have radically changed the remote patient monitoring practice in any healthcare system. Among all the possible platforms, Amazon Web Services (AWS) has received a specific amount of attention thanks to its flexibility, scalability, and advanced analytical nature. This section will overview and provide comparative analysis of a number of currently available cloud-based health-monitoring setups, including the ones that subscribe to the AWS IoT infrastructure.

The paper that has been developed in [40] suggested an AWS-based monitoring model that implements the IoT Core, Lambda and DynamoDB to monitor vital parameters of patients, including oxygen saturation and body temperature, especially in the COVID-19 pandemic. The system made sure that data flow between wearable devices to the cloud was uninterrupted, and serverless Lambda functions were effective to process/process data and send alerts. AWS architecture was lower in latency and cost performance compared to the traditional cloud-hosted solutions, particularly on variable load cases.

Likewise, [41] reviewed a system to monitor chronic illnesses based on Microsoft Azure IoT Hub. Even though the two platforms of handling real-time data are similar, stable, and have equivalent performance, the study found the benefit of AWS in handling event-driven operations with the help of its Lambda service, unlike Azure where it has been relying on instances of virtual machines more. The findings indicated that the serverless platform of AWS was more adaptable to the application that demanded quick response of the system.

In a comparison of another article, [42] proposed a wearable health tracking platform with Google Cloud IoT and Firebase. Whereas Firebase allowed deploying it more quickly and connect it to mobile apps, AWS provided more possible analytics and machine learning via Amazon SageMaker. The AWS-based solution offered a greater predictive precision which was mainly because it can train models on larger data sets and it can support sophisticated AI models.

The solution using an AWS Greengrass and IoT core is considered to be hybrid, making it integrated with edge computing to operate as a local data processing tool [43]. This

design had a minimum of latency and minimized bandwidth consumption since processing of information took place near the data source before being synchronized with the cloud. The system was particularly useful in the rural telemedicine systems where connectivity was low. AWS Greengrass was a more decentralized and resistant way of addressing the problems of fully depending on clouds, which competitors like IBM Cloud or Oracle Health Cloud had not achieved yet.

Additional comparative information was also offered in [44] wherein the compliance, privacy, and regulatory preparedness of major cloud vendors were assessed. AWS was discovered to be the most HIPAA convincing platform, with end-to-end encryption, CloudTrail showing detail audit trails, and fine-grained access control with the use of IAM. Secure operations were also supported by other providers, but AWS had more specialized healthcare compliance programs and was introduced to a blockchain technology through the AWS Managed Blockchain, which was demonstrated in [45] to be even more effective in tracking the access to data and preserving transparency.

However, [46] also found issues with interoperability in combining AWS-based systems with the current Electronic Health Records (EHR) systems. AWS was performing a little behind AWS Native integration options in comparison to Azure, which can use other EHR systems and Epic through at least built-in connectors. Newer technologies like AWS Health Lake have started to solve this problem by providing a way to store and analyze standard EHR data in a structure.

Lastly, [47] tested performance metrics of AWS, Azure and Google Cloud against latency, throughput, and uptime parameters on a monitored controlled situation. AWS has continued to offer reduced latency in the MQTT communication protocol and quicker alert delivery, which is unquestionably corroborating its robustness and applicability to time-led healthcare systems.

2.6 Gaps in Current Research

Although the evolution of cloud-based healthcare monitoring systems has been reported to have been applied in great ways, many challenges are still outstanding, particularly where the apps rely on the AWS IoT infrastructure. The key gaps in research are interoperability, data integration, accuracy of artificial intelligence models, security, scalability, and feasibility of functioning in an operational environment.

Even a basic weakness of known AWS IoT-based healthcare solutions is the lack of sufficient interoperability with Electronic Health Records (EHR) systems. As noted in [48], the majority of AWS-based monitoring platforms do not have standardized APIs capable of exchanging information with popular EHRs like Epic, Cerner, and OpenMRS. This limitation creates disjointed patient information, forming data silos that prevent IoT-generated health metrics from being fully integrated into clinical workflows, highlighting a research gap in practical interoperability solutions.

Another major gap relates to the limited use of personalized artificial intelligence in AWS healthcare systems. Although AWS provides advanced services such as Amazon SageMaker for predictive analytics, studies like [49] show that many current models rely on generic thresholds for generating alerts. Such homogeneous approaches fail to account for patient-specific variations, increasing the likelihood of false alarms or missed alerts, and revealing a research gap in developing adaptive, individualized AI models for continuous health monitoring.

A further critical gap is the lack of real-world validation. Most prior studies, including those in [50], have been conducted in simulated or small-scale pilot environments. This limitation indicates the necessity to evaluate AWS-based monitoring systems in long-term field trials with larger and more diverse patient populations to determine reliability, alert accuracy, and user experience under authentic clinical conditions.

Although edge computing technologies like AWS Greengrass have emerged, their deployment in areas with poor network coverage is minimal. As noted in [51], many AWS systems still heavily rely on constant cloud connectivity, making them less practical for rural or under-resourced regions. This limitation exposes a gap in designing architectures capable of offline or autonomous operation when network connections are disrupted.

Security and privacy remain significant concerns. Even though AWS provides encryption and access-management mechanisms, [52] highlights a gap in fine-grained access control for multi-tenant healthcare environments where multiple organizations share cloud resources. Additionally, post-breach recovery strategies and resilience testing against cyber-attacks are rarely addressed, revealing a crucial gap in system preparedness and incident management.

User acceptance is another underexplored area. According to [53], many older patients struggle with cloud-based monitoring due to limited digital literacy or lack of trust in automated systems. Current solutions rarely incorporate adaptive interfaces, voice support, or culturally sensitive designs, indicating a research gap in making cloud-based healthcare technologies more accessible and usable for diverse patient populations.

Finally, despite frequent discussion of AI and 5G convergence, practical implementations combining AWS IoT, edge AI inference, and 5G-enabled data transfer remain undeveloped. As pointed out in [54], while extensive frameworks address task distribution between edge and cloud and manage failover during connectivity loss or system overload, these solutions are largely absent in real-world deployments, highlighting a significant gap in operational feasibility and reliability.

Moreover, there is limited research addressing the energy efficiency and operational cost of large-scale AWS IoT deployments in real healthcare settings. The impact of network latency, packet loss, and device failures on multi-patient continuous monitoring remains largely unexamined. Evaluation of AI models under noisy, incomplete, or biased datasets is still scarce, creating gaps in reliability and generalizability. Practical strategies for integrating edge-cloud hybrid systems in rural or under-resourced areas are not well-established. Finally, empirical studies on long-term user acceptance, trust, and workflow adaptation are needed to ensure widespread adoption of cloud-based healthcare monitoring solutions.

Table 2.1 Focus Area Drawbacks and Research Gaps

Reference	Focus Area	Key Drawback(s)	Research Gaps / Needs
[19]	AWS Services in Healthcare	Data latency, inaccuracies	Improve predictive model accuracy
[22]	MQTT Protocol for Healthcare IoT	Potential delays in transmission	Enhance MQTT for ultra-low latency
[27]	Patient Trust in Remote Monitoring	Concerns about data accuracy	Improve data accuracy
[29]	5G Technology for Healthcare IoT	Latency issues, coverage limitations	Improve 5G reliability in underserved areas
[31]	Data Privacy in Healthcare IoT	Data breaches, unauthorized access	Stronger privacy policies, better encryption
[33]	Cloud-Edge Hybrid Systems	High latency, legacy system issues	Low-latency hybrid models, simplified integration
[36]	Real-time Healthcare Monitoring	Real-time processing delays	Enhance real-time data analytics

[38]	Remote Healthcare Monitoring	Lack of standards, data sovereignty concerns	Establish standards, address sovereignty
[40]	Automated Healthcare Systems	Data overload, integration difficulty	Develop better data handling methods
[41]	Blockchain for Medical Records	Scalability limitations, high cost	Enhance scalability, reduce costs
[46]	Privacy in Cloud Healthcare Monitoring	Potential for unauthorized access	Stricter privacy protocols, transparency improvements
[50]	Cloud Integration for Real-time Analytics	Security concerns, integration complexity	Enhance real-time security and integration simplification

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Overview of the Proposed System

In this section, the general design and operation methodology of the proposed health monitoring system on a cloud are outlined. The system applies Internet-of-Things (IoT)-enabled medical equipment, together with Amazon Web Services (AWS) cloud services to encourage uninterrupted monitoring of patient vital data and simultaneous control of health information. Its architecture is focused on the continuous data collection, scalable and Cloud processing, and secure relations between the sensors and microcontrollers on the one hand and the cloud-based modules on the other hand. Such integration allows the efficient transfer of data, its safe access, and instant warning, which makes it possible to provide quick medical attention when a deviant state of health is detected.

3.1.1 System Objectives

The main aims of the offered system can be outlined as follows:

- **Real-Time Monitoring**

  To allow endless and accurate observation of patient vitals, such as body temperature, oxygen saturation (SpO 2), pulse rate and respiratory rate, to enable the prompt detection of physiological abnormalities.

- **Intelligent Alerting**

  To automatically produce real-time alerts, which may be sent as either an SMS or email when any of the vital parameters pass a predetermined threshold, which will allow fast medical intervention and add more to patient safety.

- **Providing scalability as well as accessibility.**

  To use AWS cloud services to securely store, analyze, and visualize patient health data, which are then allowed to allow doctors and caregivers access to real-time data at any place and time.

- **Security and Data Integrity**

To ensure the safety of data delivery and storage through the use of MQTT over TLS to enable an encrypted data transmission and the use of the AWS Identity and Access Management (IAM) role and policy to control access.

3.1.2 Sensor and Device Integration

The system is centered around an ESP32 microcontroller, which interfaces with the following medical sensors:

- Measures body temperature with high precision and I2C communication.
- Pulse Oximeter (Bluetooth-connected): Captures SpO_2 and pulse rate data.
- Measures environmental parameters and is used for estimating respiration rate.

These sensors collect physiological data at regular intervals, which is then processed, formatted, and transmitted using the MQTT protocol over Wi-Fi to the AWS IoT Core. This service acts as the central communication hub, securely routing the incoming data from the ESP32 to other AWS components for storage, processing, and visualization.

3.1.3 Cloud Communication and Data Flow

Once the data reaches AWS IoT Core, the following process is executed:

- **Device Authentication:** ESP32 is authenticated using X.509 certificates and policies defined in AWS IoT.
- **Data Routing:** AWS IoT rules route incoming sensor data to other AWS services such as:
    - AWS Lambda for processing and threshold evaluation.
    - Amazon DynamoDB for persistent and scalable storage.
    - Amazon SNS for sending alerts if abnormal readings are detected.

3.1.4 Real-Time Alerting Mechanism

An automated response protocol will be activated when one or more vital signs of the patient exceed a specified limit, e.g. a high body temperature or low level of oxygen saturation.

1. The data received on the stream of the Internet of Things is initially evaluated on the AWS Lambda function.

2. In case any critical threshold is surpassed, the common triggers Amazon Simple Notification Service (SNS) and spreads an alert message.
3. This alert is sent directly to the involved caregivers or medical staff either via SMS or electronic mail thus promoting a quick medical response.

The automated workflow would significantly lower the reaction time and ensure proactive healthcare management to ensure that the performance of the patient is addressed in time instead of letting the conditions worsen.

3.1.5 Data Storage and Visualization

All incoming patient data is stored in Amazon DynamoDB which is a high-performance no-SQL database that is configured to ensure high read and write performance. Such an arrangement ensures that real-time and historical health data is quickly accessed as and when needed. The accessibility of such information is provided through a custom web dashboard, designed with web cloud frameworks, including React, providing an effective and interactive interface.

The dashboard is restricted to authorised users, and it uses role-based authentication to manage both clinicians, patients, and caregivers, which is done by considering AWS Cognito. The dashboard will visualise real-time and past trend of the vital parameters, which will allow the medical staff to track the health progress of a patient, identify irregularities, and make evidence-based clinical decisions over time.

3.2 Hardware Components

3.2.1 Development Methodology Overview

The recommended patient health monitoring system is built on a unified set of IoT-based medical sensors, microcontrollers, and wireless communication modules that can work together to ensure proper and real-time attainment and delivery of patient health information to the AWS cloud. All the elements of this hardware architecture are chosen because of their stability, accuracy, and integration with a cloud-based solution. This segment gives a clear account of the key hardware components used within the system and specifies their particular functions in supporting the processes of continuous and reliable health tracking.Creation of this hybrid health monitoring system is based on a systematic approach, which involves following the following steps:

Requirement Analysis

The proposed patient health monitoring system is based on cloud computing, a set of IoT devices, microcontrollers, and wireless communication modules, which work together to ensure high accuracy and real-time patient health data to the AWS cloud platform. The hardware components are carefully selected to ensure other components are reliable, are accurate when making such measurements and they are compatible with the cloud-based healthcare applications. Here, a detailed description of the main hardware elements of the system is also presented and the respective functions of each of this hardware elements explained in supporting efficient, continuous, and reliable health monitoring.

System Design

A scalable and engineered system architecture will have been envisioned to aid in credible and efficient health data handling. The basic AWS services, which include AWS IoT core, Amazon S3, and AWS lambda, are used to facilitate smooth connectivity of devices, secure storage of data and real time processing of data

Device Integration

The sensors and microcontrollers used in the IoT system are appropriate and are configured to capture the actual health parameters. These tools are connected via the MQTT protocol to facilitate good, low- latency transmission of data to AWS IoT Core.

Cloud Implementation

The Cloud elements are deployed such as secure endpoints, data routing rules, real-time processing Lambda functions, and data storage in the form of DynamoDB.

User Interface Development

The dashboard and alert mechanisms are created to enable medical professionals and caregivers to obtain their health information and notifications.

Security Implementation

The system uses the TLS encryption, device authentication, and IAM policies to make sure that the data about patients is safe and confidential.

Testing and Validation

Greater testing is done, including unit testing, integration testing, and user acceptance testing (UAT), which are intended to ensure that the system works.

Deployment and Monitoring

The completed system is then rolled out on AWS keeping the CloudWatch active and the updates are made based on the real-time performance indicators.

Evaluation and Feedback

User feedback and system performance reports are analyzed periodically to make the systems more usable, reliant and scalable.

3.2.2 Structural Overview

This process begins with the construction of a cloud-based patient health monitoring system that starts with an extensive requirement analysis. This first step will require the elicitation of data among the physicians, nursing staff and patients with a view of defining the functional goals the system should achieve. After this, the process of system design is carried out, thus where a robust and scalable architecture is developed using Amazon Web Services. Amazon Web Services IoT Core will facilitate the secure connections of all medical equipment, and Amazon Simple storage service (S3) will offer secure data storage of the assembled information. The next step is referred to as the device integration and in its suitable Internet of Things (IoT) sensors and microcontrollers to record accurate health metrics are selected and configured. These devices send the data to cloud through MQTT communication protocol that is rather quick and sound in transmitting data.

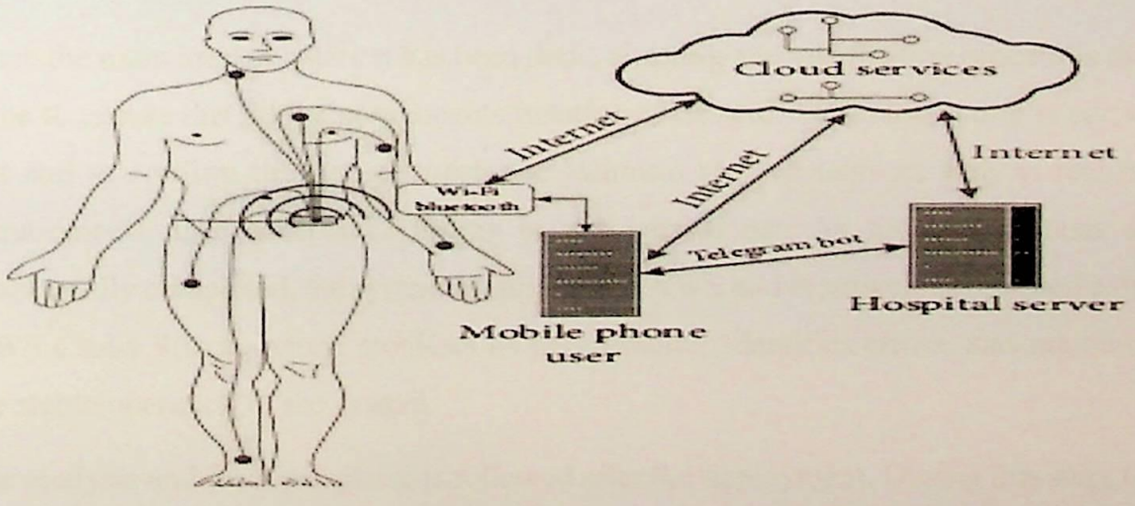


Figure 3.1 Main concept of the developed cloud-based patient monitoring system

When performing the cloud implementation, discrete Amazon Web Services (AWS) are set to operate and manipulate patient information in the most efficient manner. AWS Lambda is used to process data in perspective and conduct initial analytics every time new readings are obtained. At the same time, the user experience is crafted to be both web and mobile-friendly to ensure the physicians and patients are able to easily access the health information, reports, and other alerts.

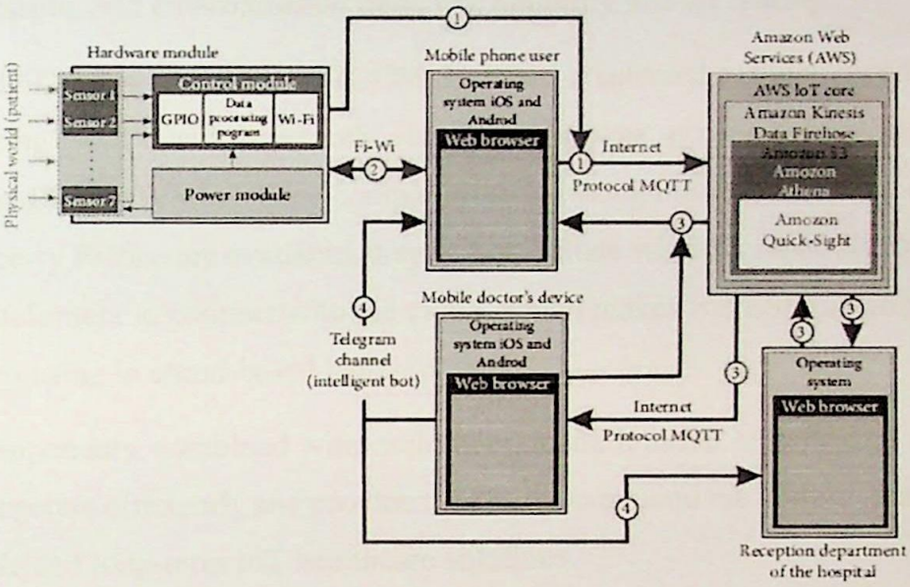


Figure 3.2 Structural scheme of cloud-based patient monitoring

Security configuration is a vital element in this phase where all data is encrypted and suitable access controls implemented to ensure protection of sensitive medical information.

Once the main implementation has been done, a testing and verification process is then done to ensure that all the components function as desired. Different testing is carried out and to confirm the system meets the technical requirements as well as realistic requirements, user acceptance testing is also carried out. As soon as all tests are successfully completed, the system is launched in AWS and is actively monitored using AWS Cloud Watch, which monitors its performance, identifies errors, and maintains the stable operation of the system.

The analysis and feedback phase is followed after the deployment. During this step, the performance indicators of the system and user satisfaction are questioned to exploit the

possible areas of improvement. The feedback obtained through the medical workers and patients will be used to make future changes and further enhancements where the solution can be modified to meet future healthcare demands as well as technological changes.

In the hardware choice, the system will be based on a reliable set of sensors and iot development kits to provide accurate and effective health monitoring. The Temperature Sensor, Bluetooth Pulse Oximeter, and respiratory sensor will be used to give a complete range of vital data, including body temperature, heart rate, SpO 2, and ECG, blood pressure, and environmental data, like humidity and air quality.

The ESP32 DevKit v1 or ESP-WROOM-32 board is selected as the main controller due to attracting Wi-Fi and Bluetooth plus features, low power-consuming, two-core system, and free AWS IoT Core compatibility. Although more affordable options like the Raspberry Pi Pico are available, they do not include wireless capabilities and require further equipment to connect with the cloud, which makes the ESP32 a more effective solution to using in cloud-based health project.

These components, combined with each other, made it easier to send and receive data quickly, operate efficiently and provide the ability to expand the system, making it ideal in scalable and long-term IoT healthcare solutions.

Collectively, all these elements enable quicker data exchange, eco-friendly operation, and scalability of the system changes making them better suited to scalable and long-term healthcare IoT solutions.

3.2.3 Selected IoT Sensors and Modules

For the accurate collection of vital signs, the following hardware components have been selected based on their precision, compatibility, and power efficiency:

Table 3.1 Selected IoT Sensors and Modules for Vital Signs Monitoring

Sensor/Module	Monitored Parameter	Features	Role in the System
Body Temperature sensor	Body Temperature	High precision, I2C-based	Measures core body temperature and send data to ESP32.

Sensor/Module	Monitored Parameter	Features	Role in the System
Pulse Oximeter	SpO_2 and Pulse Rate	Bluetooth-enabled, low power	Measures oxygen saturation and pulse rate via BLE.
IAQ Sensor	Respiration Rate, Temp, Humidity	Environmental sensor, gas detection	Estimates respiration rate and environmental impact.
ESP32 DevKit V1 / ESP-WROOM-32	Microcontroller	Dual-core, Wi-Fi + BLE, low power	Central controller for sensor integration and MQTT communication with AWS.

3.2.4 Pulse Oximeter and Heart Rate (BME1300)

The MAX30102 heart rate and SpO_2 sensor is used in this project because it provides accurate, real-time measurement of a patient's pulse rate and blood oxygen levels. These two parameters are essential for continuous health monitoring and help in the early detection of cardiac or respiratory issues. The sensor operates in a non-invasive manner, allowing safe and comfortable use for patients over long periods. Compared to other available options, the MAX30102 delivers higher precision, with an SpO_2 accuracy of about ±2% and a heart rate accuracy of ±3 beats per minute.

3.2.5 (Body Temperature)

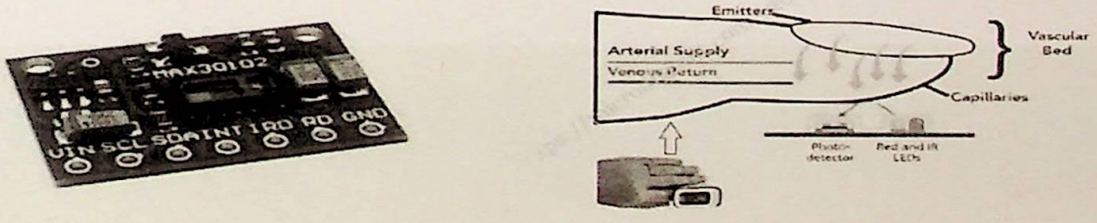


Figure 3.3 Pulse Oximeter and Heart Rate Transducer

Tl s

of human body temperature which is a critical requirement of continuous monitoring of patients. It provides valid and real-time data that makes it possible to identify cases of pyrexia and other physiological changes that are clinically relevant early. Compared to similar types of instruments, it has a better result of an accuracy of up to ±0.1 o C at a low level of power consumption. It is especially suitable in Internet-of-Things-based

healthcare platforms: the small size and simplified design of Studio Two-Block make the measurement of temperature in real time and precision a matter of life and death to patients and timely healthcare services.

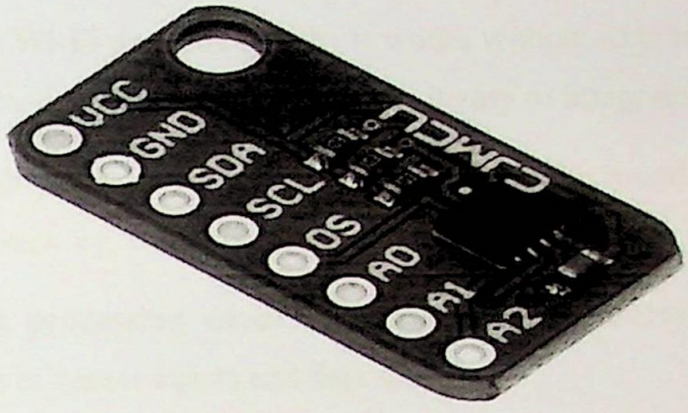


Figure 3.4 Body Temperature Sensor

3.2.6 Respiration Rate and Environmental Sensor

The sensor is spacious and universal and it has been invented by Bosch Sensortec. It incorporates various sensing capabilities in one module which is gas (VOC) detection, humidity, barometric pressure and ambient temperature measurement. Though the sensor is mostly used to monitor the environment, it can be used in healthcare systems to ensure that respiratory rate is approximated indirectly. This is done by monitoring minute variations in gaseous concentration and humidity with every breathing pattern which makes it a suitable technology in wearable or indoor health monitoring system where there is a need to measure proximity.

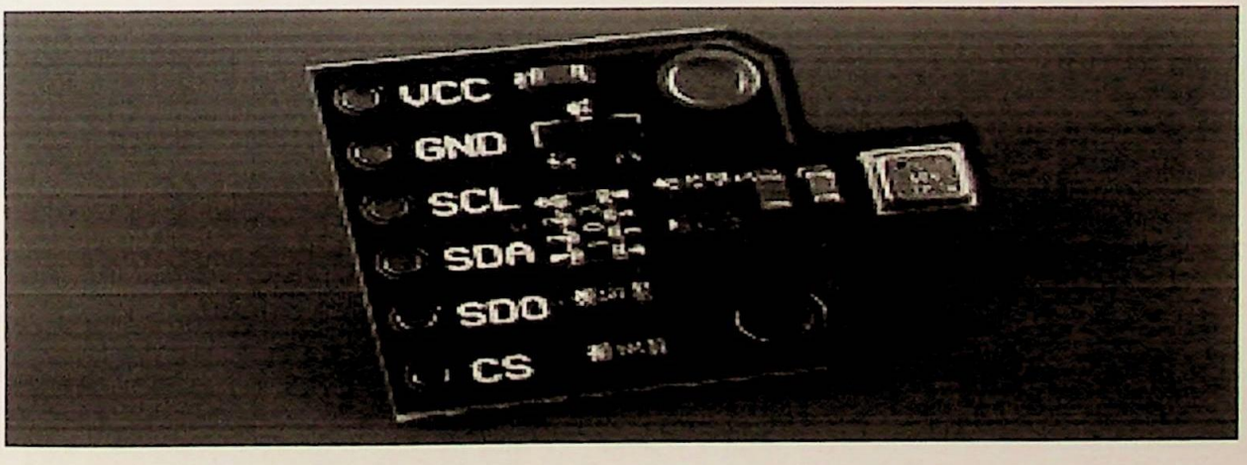


Figure 3.5 Respiration Rate and Environmental Sensor

3.2.7 Microcontroller Selection Justification

The ESP32 Dev Kit V1 (also known as ESP-WROOM-32) of various other development boards has been chosen due to the following strengths:

- **Inbuilt Wi-Fi and Bluetooth:** It works without external modules that would otherwise be used, therefore, making it easy to integrate into the cloud.
- **No high-power usage:** Appointed to working round-the-clock in terms of health tracking, which provides a long battery life.
- **Intense processing capacity**: The dual/core processor can handle a large number of sensor inputs and data format.
- **ESP32 is available on site**, and we have realized the AWS IoT integration using the domain name of the ESP32 functionality. To provide secure MQTT-based device provisioning and publishing, the AWS IoT SDK is utilized.

Although the Raspberry Pico and other options are cheaper, they lack the inbuilt wireless connectivity. This weakness requires the inclusion of extra Wi -Fi or Bluetooth devices, which complicate hardware and add to power usage. In comparison, ESP32 provides in-built wireless capabilities, low power consumption, and dependable performance hence the choice in cases of portable and battery-based healthcare monitoring.

3.2.8 Hardware modules for implementing cloud-based system management

(a) ESP-32 Devkit 1 Node MCU

The ESP Stringing ESP-32 Device professional kit is a prototyping platform which relies on the ESP-WROOM-32 splashboard, which integrates the ESP32 System-on-Chip (S) provided by Espressif Systems. Integrated Wi-Fi and Bluetooth connection is provided by this board, making it especially suitable to Internet of Things (IoT) activities. This is because it has the advantage of processing efficiency, low power consumption and dual-core architecture which, combined, enable the hassle-free performance of real time monitoring and control functions simultaneously.

The important characteristics of Esp 32 DevKit 1 are:

- Dual-core Tensilica LX6 processors, operating up to 240 MHz
- Integrated Wi-Fi (802.11 b/g/n) and Bluetooth v4.2 BR/EDR and BLE.

- 520 KB SRAM and 4 MB Flash memory, suitable for storing firmware and data.
- Rich peripheral interfaces: ADC, DAC, SPI, I2C, UART, PWM, capacitive touch, and more.
- Onboard USB-to-serial converter for easy programming and debugging via USB.

ESP 32 Devkit 1 is very ideal in making remote patient monitoring systems as it can be integrated with real time wireless communication with IoT core like AWS devices. Its inbuilt Wi-Fi and Bluetooth connectivity do not require any extra communication devices to be used hence making the systems simple to design and are not very expensive. The ESP-32 has a non-challenging interface with numerous types of biomedical sensors, such as temperature, pulse oximetry, respiration sensors, and others, in healthcare IoT applications, and gather patient health data, which is transmitted to the cloud to be stored and analyzed, and alerts issued. Even more, its low-power operating modes, combined with low power consumption, result in a high-quality choice of a wearable or portable medical device, where long battery life and reliable connectivity are crucial factors. In summary, ESP -32 Devkit -1 Node MCU is a microcontroller with reliable, scalable, and affordable features that can be used to build the cloud-linked healthcare monitoring solution.

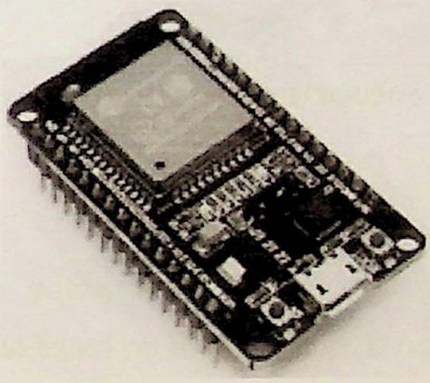


Figure 3.6 (a) ESP-32 DevKit 1 Node MCU

(b) ESP-32S ESP-WROOM-32

ESP is the Ethernet specific probing module ESP -32S (sometimes known as ESP -WROOM -32) is a Wi -Fi and Bluetooth able multi-purpose microcontroller board designed by Espressif Systems to power Internet of Things (IoT) devices. It has a dual-core Tensilica LX6 processor with frequencies of up to 240MHz and 2.4GHz Wi-Fi

with Bluetooth v4.2 (Classic and BLE) that make it easy to have reliable and efficient wireless interaction.

The module provides a wide range of general-purpose input/output (GPIO), analogue-to-digital converter (ADC), digital-to-analogue converter (DAC), universal asynchronous receiver/transmitter (UART), serial peripheral interface (SPI), inter-integrated circuit (I 2 C), pulse-width modulation (PWM). These interfaces allow easy integration with a wide range of sensors and actuators, its low-power operating modes makes it a promising choice at baby power levels in battery-powered healthcare monitoring systems with energy-efficiency becoming the key factor.

Due to its small footprint, affordability, and support to the ESP32 DevKit 1 NodeMCU platform, the ESP-32S has a strong usage in embedded medical applications.It enables real-time data collection, wireless transmission, and seamless integration with cloud services like AWS IoT Core, making it an ideal choice for modern patient health monitoring systems.

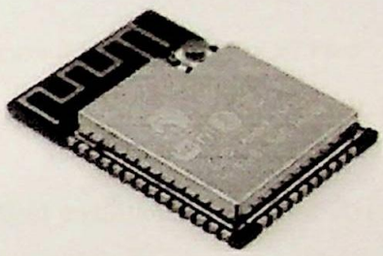


Figure 3.7 (b) ESP-32S ESP-WROOM-32

(c) Raspberry Pi Pico

Raspberry Pi Pico is a low pricing and powerful microparticle card created by the Raspberry Pi Foundation and based on RP2040 chip. It has a dual core ARM Cortex - M0 + processor with maximum clock speed of 133 MHz, 264 KB of SRAM and 2MB onboard flash memory. The board is aimed at embedded projects and IoT based projects that need dependable processing performance and yet at a minimal price.

The Raspberry Pi Pico does not have an integrated Wi-Fi or Bluetooth on board as the ESP-32S does, unlike the ESP-32S, the Raspberry lacks wireless communications capability and can only be used in this way with additional communication modules. However, it has a wide support of I/O, such as 26 multifunctional GPIO pins, I 2 C,

SPI, UART, PWM and ADC, supporting integration with a wide range of sensors and peripheral devices.

Raspberry Pi Pico can be used in educational and prototyping settings, and thus is popular among the audience without advanced skills in embedded development, signal processing and hardware interfacing. A local data acquisition unit may be used in a cloud-based health monitoring system to receive sensor data and preprocess such data. Nonetheless, since it involves an external wireless communication platform like the ESP8266 or ESP32 to be connected to the cloud, the ESP -32 platform is more of an integrated and more practical option when it comes to the healthcare-related IoT implementation that requires the constant connection to the cloud.

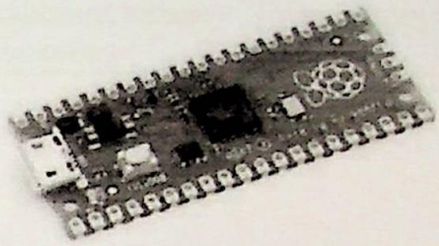


Figure 3.8 (c) Raspberry Pi Pico

Comparison of IoT Development Boards for Cloud-Based Patient Monitoring Systems

Here's a comparison table that evaluates the three IoT development boards—ESP-32 DevKit 1, ESP-32S (ESP-WROOM-32), and Raspberry Pi Pico—based on key factors such as cost, operational efficiency, and operation time:

Table 3.2 Comparison of IoT Development Boards for Cloud-Based Patient Monitoring Systems

Feature	ESP-32 Devkit 1 (NodeMCU)	ESP-32S (ESP-WROOM-32)	Raspberry Pi Pico
Cost	Moderate ($8 - $12 USD)	Low to Moderate ($5 - $10 USD)	Very Low ($4 - $6 USD)
Processor	Dual-core 240 MHz	Dual-core 240 MHz	Single-core 133 MHz
RAM	520 KB SRAM	520 KB SRAM	264 KB SRAM
Flash Memory	4 MB Flash	4 MB Flash	2 MB Flash
Wi-Fi Connectivity	Yes	Yes	No
Bluetooth Connectivity	Yes (Bluetooth Classic & BLE)	Yes (Bluetooth Classic & BLE)	No
I/O Pins	30+ GPIO pins	34 GPIO pins	26 GPIO pins

Power Consumption	Low to Moderate	Low to Moderate	Very Low
Communication Protocols	Wi-Fi, Bluetooth, SPI, I2C, UART	Wi-Fi, Bluetooth, SPI, I2C, UART	I2C, UART, SPI
Ease of Use (Programming)	Easy (Arduino IDE support)	Easy (Arduino IDE support)	Easy (MicroPython/C/C++ support)
Operational Efficiency	High (good for real-time tasks, low latency)	High (good for real-time tasks, low latency)	Moderate (lower processing power)
Operation Time (Battery Life)	Moderate (Depends on usage, around 200-300 hours on low power)	Moderate (Similar to ESP-32 DevKit)	High (Very low power consumption, lasts longer on small batteries)
Use Cases	IoT, home automation, healthcare, remote monitoring	IoT, wearable devices, healthcare, wireless applications	Prototyping, low-power applications, education, simple IoT projects

Key Takeaways

Cost:

The Raspberry Pi Pico is the least expensive solution and it makes it the choice of simple or low-budget projects. However, the ESP-32 DevKit-1 and ESP-32S, which are relatively a bit costlier, have augmented features, such as built-in Wi-Fi and Bluetooth 2.1 features, which cannot be done without in Internet-of-Things (IoT)-driven healthcare applications.

Operational Efficiency:

The ESP-32 DevKit 1 and ESP-32S have high efficiency of operation, which makes them ideal in the real-time internet of things E.g. in-patient health monitoring. As Raspberry Pi Pico can perform rudimentary IoT functions and, in contrast to major processors, it has relatively low processing capabilities and would not be as efficient at performing intricate data processing tasks or handling two interconnected wireless networks simultaneously.

Battery Life:

Raspberry Pi Pico proves to be highly efficient in terms of power consumption, thus, making it possible to run it during a long time in battery-powered applications. In contrast, the power consumption of both ESP-32 boards can be defined as moderate; it is additionally optimized by the low-power modes, which adjust the performance and

the power consumption settings, which is optimal in case the boards should be used in continuous monitoring devices.

3.2.9 Key Reasons for Selecting ESP-32 Devkit 1 or ESP-32S

For a cloud-based patient monitoring system utilizing IoT and AWS, the ESP-32 Devkit 1 or ESP-32S (ESP-WROOM-32) stands out as the most appropriate choice. The reasons are outlined below:

1. **Wi-Fi and Bluetooth Network:**
   Cloud-based health monitoring implies the incessant transfer of physiological information to cloud computing like AWS IoT Core. The two ESP-32 versions also provide built-in Wi-Fi and Bluetooth Low Energy (BLE) connectivity, thus providing a smooth and reliable wireless connexion. On the contrary, the Raspberry Pi Pico does not support wireless modules out of box and requires extra hardware in order to support equivalent connectivity, which increases the complexity and cost of the system.

2. **AWS Integration:**
   ESP-32 boards have a high-level of compatibility with AWS IoT Core and its other services, which allow transferring, storing, and analysing real-time data using MQTT or HTTP protocols. Even though the Raspberry Pi Pico could be used to connect to AWS by attaching external modules, it is less efficient and requires more setup, and programming overhead.

3. **Low Power Consumption:**
   The two ESP-32 modules have several low-power modes, which make them widely applicable in battery-operated healthcare systems, which need to occasionally transfer patient vitals without the need to recharge regularly. Although the Raspberry-Pi-Pico proves to be energy efficient, it has not been optimised to maintain low-power wireless communication hence limiting its usability in maintaining ongoing cloud connexion.

4. **Processing Power:**
   The ESP-32 DevKit 1 and ESP82 S offer a CPU with 240Hz and 520KB of SRAM, which is powerful enough to perform real-time data collection and signal processing and to use safe connexions to the cloud. The Raspberry Pi

Pico, with distinctly fewer memory and a single-core processor: 133 MHz processor, might face challenges in process core requirements simultaneously processing sensor data collection and communicating with the cloud through real-time experiences.

5. **Scalability:**

The ESP-32 platform also facilitates scalability among many devices, and it is easily added to a network of IoT-based patient monitoring departments under AWS iot core. This feature makes it suitable to massive healthcare networks that involve multiple patients or hospital systems. On the other hand, the Raspberry Pi Pico is better suited to small-scale or learners relating projects where scaling is not a major issue.

3.2.10 Integration with IoT

The implementation of the IoT technology in the offered patient health monitoring system is one of the key factors that will simplify the conduct of real-time data capture, wireless communication, and safe transmission of patient vital signs to the cloud. This section sketches the connection of a series of biomedical sensors with the ESP32 DevKit 1 microcontroller and then linked to Amazon Web Services (AWS) cloud solutions in order to realize real time monitoring and analytical processing.

The system architecture is divided into three main layers which include sensing layer, communication layer, and the cloud layer. There are biomedical sensors like thermistors, photoplethysmographic heart -rate sensors, pulse-oximetry units, connected to the ESP32 with the help of analog or digital input pins in the sensing layer. The onboard analog-to-digital converter knowledgeable (ADC) of the ESP32 converts the sensor analog input to digital input streams, which are later filtered onboard, dropping the electronic noise and promoting the precision of the measurements.

The ESP32 uses its inbuilt Wi-Fi capability in the communication layer to establish saved MQTT connection with the AWS IoT Core. Adoption of lightweight communication protocols provide efficient and reliable data transmissions even in environments that are of limited bandwidth. Every device is set up as separate Thing in AWS IoT Core where-in, exclusive X.509 certificates and cryptographic keys are used to offer an encrypted, authenticated communication path.

Lastly, the cloud layer directs incoming sensor data, through AWS IoT Core, to a set of backend services value services such as AWS Lambda, which is a serverless computing service; Amazon DynamoDB, which is a scalable data storage service; and Amazon QuickSight, which is a data analytics service. All these services allow healthcare providers to obtain the health parameters of patients in real time at a distance, detect abnormal physiological patterns, and issue timely warnings using automated rule-based services. As a result, the complete system provides the scalable, secure and energy saving platform that ensures the uninterrupted continuous remote health monitoring.

3.2.10.1 Sensor-to-Microcontroller Communication

Each biomedical sensor in the proposed system is interfaced with the ESP32 microcontroller, which functions as the central data aggregator responsible for acquiring and processing all physiological signals. The interfacing details of the sensors are as follows:

- (Body Temperature Sensor):The body temperature sensor is a digital device which can be connected to the ESP32 through the I 2 C protocol, thus providing the body temperature sensor with high-precision and constant temperature reading. Its 16-bit resolution with factory calibration can be used to effectively measure the minor changes in temperature that are very crucial in constant monitoring of the patients. (Bluetooth Pulse Oximeter): The pulse oximeter module forms a BLE connexion with the ESP32 to detect real-time heart rate and SpO 2 (oxygen saturation) rate as well as transmit it. The BLE interface is low power and is dependable in short range wireless communication making it appropriate in wearable medical devices.
- (Respiration and Environmental Sensor): The respiration and environmental sensor is an ESP32-connected device with the I 2 C interface, which reports several environmental parameters, including air pressure, humidity and ambient temperature. Changes in these parameters can be evaluated to indirectly determine the rate of respiration and added into a complete assessment of the state of health.

One of the features of the ESP32 is to periodically access data delivered by each of the biomedical sensors that are connected to it with a specific time delay. The

microcontroller is required to run data -acquisition algorithms that sample what the sensor outputs, noise and convert raw analog concentrations to digital numbers, which are calibrated. The processed readings then get the meaning transformed into meaningful health metrics which are the body temperature, heart rate, and oxygen levels of saturation using proper conversion algorithms. The process of validating and formatting the data and then serializing into the JSON or MQTT compatible payloads will prepare it to be sent to AWS IoT Core in a safe format.

3.2.10.2 Wireless Data Transmission via Wi-Fi

The ESP32 has an intrinsic Wi culmination bogarion to transmit physiological measurements to a cloud platform over the Wi internet. The process consists of four main steps: the raw sensor data is initially serialised into either JSON or formatted MQTT messages; the ESP32 then sends these messages via the MQTT protocol to the AWS IoT Core, thereby maintaining privacy of patient data over a network; these and TLS are used to protect the message transmission, finally the built-in wireless design allows the use of a wireless sensor without additional network connectivity devices.

3.2.10.3 AWS Cloud Integration

After reaching AWS IoT, the incoming data is redirected and run through a series of services offered by AWS thus leading to reliability, scaling and intelligent treatment of patient health facts. As explained, the system will transfer the data safely out of the ESP32 microcontroller to the AWS cloud where it will go through a set of actions process, persistent storage, and alerts generation, which will be implemented through a set of services, interconnected with one another:

AWS IoT core serves as the main entry point to all sensor output, it manages the authentication of devices, it receives incoming data streams of the ESP32, and it transmits messages to the Rules Engine, which forwards the messages to downstream services based on their content.

- Does not have servers: An AWS Lambda engages serverless processing of analytical operations in real time, such as data validation and threshold comparisons and anomaly dental procedures and pre-persistence format before data is stored or an alarm is generated.

- Amazon DynamoDB works as time-series database that stores structured health record which can be used to conduct longitudinal analysis, trends evaluation and visualisation on monitoring dashboards.
- Amazon Simple Notification Service (SNS) provides an automated notification system that sends providers of healthcare an SMS or e-mail notification whenever any abnormal measurements, including high temperature, low SpO2, and atypical heart rate are detected.

This unified cloud framework will ensure that patient information is continuously collected, sent safely, processed smartly and used in real time, enhancing the reliability and reactiveness of far-off health-checking techniques.

3.2.10.4 Advantages of IoT-Based Integration

The adoption of a patient health monitoring framework based on the IoT has numerous benefits that contribute greatly to the effectiveness, availability, and dependability in healthcare delivery:

- **Real time monitoring:** It also allows the constant acquisition of patient vital signs, and real time alert generation in the event of abnormalities, which will result in timely medical attention.
- **Remote accessibility:** Cloud dashboards enable certified medical practitioners to access patient health information at any time and place remotely, thus, making telemedicine and distance monitoring possible.
- **Scalability:** The system architecture allows easy attachment of more biomedical sensors or monitoring devices without imposing any substantial change in the hardware and software.
- **Also:** The built-in Wi-Fi and Bluetooth capabilities of ESP32 would help to eliminate the use of external communication system thus lowering cost and power requirement.
- **Interoperability:** The framework utilises common languages of communications such as MQTT, Bluetooth, and I 2 C, to ensure that there is seamless interconnectivity among a wide range of sensors, microcontrollers and cloud services.

Overall, this IoT architecture provides a scalable, low-cost, and flexible real-time patient health monitoring solution that uses the cloud, which makes it the perfect solution to both hospital and home-based health care services.

3.2.10.5 Communication and Power Management

The proposed cloud-based patient monitoring system incorporates optimized design choices to ensure reliability, energy efficiency, and effective data transmission.

- **Communication Protocol:** The system employs the MQTT (Message Queuing Telemetry Transport) protocol for communication between the ESP32 microcontroller and AWS IoT Core. MQTT's lightweight nature and publish–subscribe architecture enable efficient, low-latency, and reliable real-time data exchange, making it ideal for IoT-based healthcare applications.
- **Power Supply:** The hardware platform is powered by rechargeable Li-Ion batteries, ensuring portability and uninterrupted operation in field or home-based healthcare environments. The ESP32 and associated sensors are configured to operate in low-power modes during idle periods to extend battery life and support long-term patient monitoring.
- **Data Sampling Frequency:** Sensor readings are acquired at 5-second intervals under normal operation. The system dynamically adjusts the sampling rate based on patient condition or detected anomalies—for instance, increasing the frequency during emergency situations or when critical variations in vitals are detected.

3.3 Software Components (IoT Platforms, Cloud Services, etc.)

The software modules are the underlying software infrastructure that controls data flow, processing of analytics and interaction between users in an ecosystem of patient health monitoring in the cloud. This layer will guarantee a very efficient coordination among hardware sensors, Internet-of-Things (IoT) communication protocols, and cloud-based analytical services.It combines various IoT platforms and cloud solutions to make the data acquisition in real time, provide automated analytics and allow easy access to users, therefore, facilitating uninterrupted harmonisation of heterogeneous devices and services.The architecture will be built based on scalability, security and interoperability

requirements, such that the patient health data are standingly gathered, passed, processed, and shown without interruption.

3.3.1 IoT Platform: AWS IoT Core

AWS IoT core acts as the focal point that enables the connection of devices, secure communication as well as real time interchange of data between Internet-of-things (IoT) devices and cloud infrastructure. ESP32 microcontroller along with the biomedical sensor interfaces are connected to transfer the summation of the physiological data to the AWS IoT Core using the MQTT (Message Queuing Telemetry Transport) protocol. MQTT is a minimalist publish-subscribe messaging protocol with low-bandwidth, low-power, and high-reliability hardware designs, which makes it specifically viable in the context of the ongoing publication of vital signs of patients.

AWS IoT Core offers information integrity and authentication with the device with the assistance of the use of X.509 certificates, which is why only authorized devices are allowed to connect and publish information. Additionally, AWS IoT policies control the value of data-access permissions, consequently facilitating an end-to-end security throughout the communication chain. This is secure and scalable, which facilitates real-time health tracking, systematically logs and event-based analytics inside the AWS environment.

3.3.2 Data Processing: AWS Lambda

AWS Lambda is the serverless computing architecture of the health-monitoring system proposed, which allows real-time processing of data sets without using dedicated servers. When sensor data are received and processed by using AWS IoT Core to convey their content, an AWS Lambda run-time is triggered as well. Some of the critical functions that are carried out in this functionality include data validation, threshold comparison, and alert generation in case of anomalous health parameters.

AWS Lambda runs code only when it is needed resulting in resource efficiency, automatic scaling, and cost-efficiency by adopting serverless architecture. The method results in the system being capable of handling dynamic workloads, providing an opportunity to support an increasing number of connected devices or patients, with no decline in performance. As a result, AWS Lambda will ensure that patient information is run in a timely and secure manner, hence enhancing prompt medical treatment.

3.3.3 Data storage Amazon DynamoDB.

The main data-storing aspect of the cloud-based patient health-monitoring is Amazon DynamoDB. All incoming sensor readings and their respective Timestamp and patient identification tag are stored in a structured DynamoDB table. Being a NoSQL, DynamoDB provides high availability, low latency, and automatic scalability thus being specifically designed to be used with large datasets that continuously grow due to real-time health-monitoring applications.

Unable to accommodate different types of data between which many sensors are bringing data, schema-less design of DynamoDB provides a high level of flexibility without forcing the structural changes. This flexibility guarantees that things can be integrated smoothly with the introduction of new biomedical sensors or other new parameters of the patient. In general, DynamoDB supports quick access to data, high storage and organization of patient health records, which leads to the subsequent analysis and visualization.

3.3.4 Notification Services: Amazon SNS

The system uses Amazon Simple Notification Service (SNS) to send real-time notifications and alerts to people to guarantee promptness in communication during emergencies and other health-related incidents. AWS Lambda automatically sends a message to a designatedSNS topic whenever it notices that sensors have been reading abnormally or Fahrenheit spontaneouslyraising the threshold above the pre-determined alert threshold. The message is then sent as an email or SMS to the relevant medical practitioners or caregivers.

SNS allows medical personnel to react to possible emergency situations as soon as possible, minimizing the possibility of excessive response, by providing instant notification. The reliability, scalability, and the ability to combine with other elements in the AWS make the service a perfect solution to ensure the stability of constant and responsive monitoring of patients in the cloud-based healthcare system.

3.3.5 Authorization and User Interface.

Its system also contains an interactive and user-friendly web dashboard that was built in React.js and created to display both real-time and past health data in an approachable

format. The dashboard can be used to track the health status of patients by showing real-time graphs, health summaries, and patient history.

In order to have secure access, role-based authentication and authorization are applied with the use of AWS Cognito. This service controls the user identities and thus, doctors, caregivers, and patients possess the unique roles. Cognito will be connected to the web interface with the help of the JSON Web Tokens (JWT), so users can enjoy safe user sessions and the continuity of Web Sessions. Such a mixture of live image representation and the access control increases the reliability and convenience of the cloud-based monitoring system.

3.3.6 Monitoring and Logging: AWS Cloud Watch.

In order to achieve constant performance and create the reliability of systems, AWS Cloud Watch is applied to monitor, log records, and manage alarms comprehensively. It gathers the usage of logs in operation of AWS Lambda, records the consumption of resources and sends notifications to administrators whenever anomalies or errors have been identified. This functionality increases the transparency of the system, simplifies the debugging, and contributes to the optimization of the performance.

Finally, the software architecture is also solid, supporting the proposed health monitoring system. The system includes AWS IoT Core that enables the device-to-device communication safely; AWS Lambda that provides real-time data processing services; Amazon DynamoDB that provides efficient data storage services; Amazon SNS that provides automated notification services; AWS Cognito services because they are user authentication services; and Amazon CloudWatch that also supports uninterrupted monitoring.The combination of these integrated components is one of the reliable, scalable, and secure cloud-based health monitoring solutions that can address the current healthcare needs.

3.4 Network Architecture and Data Flow

The created Remote Patient Monitoring System (RPMS) is based on the approach to a layer-3 IoT-cloud architecture to support the constant observation of vital signs of patients. After the successful completion of the experimental prototype, there is an urgency to coordinate the key elements of the system, such as data acquisition, effective transmission, storage on the cloud, and visualization to provide healthcare workers with

stable long-distance access. This chapter describes the network structure of the system and the end-to-end data flow with a special emphasis on designing every step of the system to ensure its security, scalability, and real-time functionality. Having sensor data collection at the end of the patient, and cloud storage and processing, all the layers are combined to provide an effective communication approach as well as maintain data integrity and accessibility to make a clinical decision as soon as possible.

3.4.1 MQTT Protocol on Data transmission.

The Remote Patient Monitoring System (RPMS) uses the MQTT (Message Queuing Telemetry Transport) protocol to send biomedical information of sensors to the cloud. MQTT is a weightless, open-source, publishsubscribe device that is customized among constrained-resource devices, e.g. microcontrollers, and networks with low bandwidth or high latency [58]. It uses TCP/IP and port (1883) for unencrypted communication and port (8883) to use the encrypted model of communicating through the use of secure (SSL) encryption, thus, maintains a safe end to end protection of data [59].

The ESP-32 DevKit 1 Node captures patient vitals, such as body temperature, oxygen saturation (SpO 2), and respiration rate and shares it through a Wi-Fi network with the AWS IoT Core service using MQTT. This protocol has been chosen because of its:

- Comp act size and same overhead,
- Synchronous messaging capability,
- Quality of Service (QoS) of differing degrees,
- Stability- high network conditions,
- Smooth compatibility with other IoT hardware sources [58].

Pseudocode Reference:

- Pseudocode 3: Connection to Wi-Fi and AWS IoT Core using MQTT.

3.4.2 Secure Connection via secrets.h File

To prevent unauthorized access and ensure secure connections, all sensitive credentials are stored in a separate header file named secrets. h. This file contains:

- Wi-Fi SSID and password,
- AWS IoT Core endpoint URL,
- Root CA certificate,

- Device certificate,
- Private key.

These credentials are used by the ESP-32 to perform an SSL handshake with the AWS server. During runtime, the firmware reads this information to establish a secure connection, keeping sensitive data hidden from the main source code [60].

Method presents the connection code fragment below.

```
#include <pgmspace.h>

#define SECRET
#define THINGNAME "esp32"

const char WIFI_SSID[] = "**********";
const char WIFI_PASSWORD[] = "***************";
const char AWS_IOT_ENDPOINT[] = "a2v4p81z4x2d9k-ats.iot.eu-west-2.amazonaws.com";

// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
```

Figure 3.9 A Fragment of the secrets file

Pseudocode Reference:

- **Pseudocode 2:** Secure storage and use of Wi-Fi and AWS credentials from secrets.h.

3.4.3 Wi-Fi Access and Network Protection

The ESP-32 module connects to a nearby Wi-Fi access point [either a router or a hotspot] by means of static credentials Philipsheid has built into its firmware. In this way, dynamic pairing steps are not needed, which is used to protect the device at man-in-the-middle attacks and non-permitted network connections. When the wireless link is successfully established, the system continues to open a connection to AWS IoT Core

using a secure MQTT protocol based on the usage of a secure protocol, which is built on top of SSL/TLS.

The connection to the device is activated via the MQTT client part of the ESP 32 software. This customer is programmed to have automatic reconnection in case the active session becomes interrupted.

Pseudocode Reference:

- **Pseudocode 3:** Establishment of secure MQTT connection after Wi-Fi link is verified.

3.4.4 Conversion of Sensor Data to JSON Format

After creating a communication channel raw biomedical data received by sensors (e.g. temperature sensors, SpO 2 sensors, environmental sensors) are packaged in JSON (JavaScript Object Notation). JSON is a lightweight and human readable format that is widely used in web applications and cloud-based applications as a representation of data. Converterting sensor data to JSON will ensure that the messages sent through MQTT are standardized and can easily be processed by the AWS solutions of Lambda and IoT Core rules.

Pseudocode Reference:

- **Pseudocode 4:** Formatting patient vitals data—temperature, oxygen saturation (SpO_2), and respiration rate—into JSON before sending via MQTT.

3.4.5 Data Publishing to AWS IoT Core

Once the JSON dataset has been prepared, data is sent to a specified topic in AWS IoT core through the publishmessage () functionality. This role has an MQTT client that is used to transmit messages over channel allocated to each patient or device in a secure manner. The data is then received by AWSS IoT core where it is authenticated and forwarded to process it further.

As the gateway to all IoT data, AWS IoT core enforces rules and filters to send messages to relevant AWS services to either store, analyze, or receive an alert.

Pseudocode Reference:

- **Pseudocode 5:** Publishing of patient data to AWS IoT Core using MQTT.

3.4.6 Data Processing with AWS Kinesis Firehose

Messages sent to AWS IoT Core are forwarded to Amazon Kinesis Data Firehose, which handles ETL (Extract, Transform, Load) operations on the incoming data stream. These operations include:

- Aggregating data based on timestamps,
- Reformatting sensor metrics,
- Delivering the processed payload to Amazon S3 for storage [61].

Kinesis Firehose ensures minimal latency while enabling real-time data ingestion into AWS storage and analytics services.

3.4.7 Scalable Storage with Amazon S3

The information handled by Kinesis Data Firehose is stored in the Amazon S3 which has highly scalable and secure as well as durable storage. The data on patients is categorized into buckets and the access is heavily regulated via IAM policies, where only authorized users or applications will access and update the records [62].

3.4.8 Obtaining Data with Amazon Athena.

The results of patient health trends can be analyzed by querying the stored data of the time-series on S3 using Amazon Athena, a query engine utilizing the SQL language that is run on-demand and serverlessly. Athena also facilitates the real-time retrieval and analysis or reporting of data without requiring the complex setups of databases.

3.4.9 Visualization in Amazon Quick Sight.

The outcomes of the Athena queries are plotted with the help of Amazon QuickSight, which is a cloud-based business intelligence system that enables:

- Interactive charts,
- Trend analysis,
- Alarms in case of unusual states.

Doctors and hospital staff have been able to use this dashboard to make clinical decisions on it with confidence [63].

3.4.10 Access Control with AWS IAM

There is the Identity and Access Management (IAM) where the system employs a mechanism that ensures that security is upheld and that patient data is protected. IAM manages user roles, permissions, and access key to make sure that the health information is accessible to only the authorized personnel. The administrators are capable of establishing access regulations in detail depending on the role of the user like doctor, nurse or system administrator. Every activity such as access to data, system modification, is logged and audited to conform to the medicine 4.0 requirements regarding medical data security [64].

3.5 Generalized Operation Flow

The complete flow of data, from sensor acquisition to dashboard visualization, is illustrated in Figure 5: Generalized Operation Algorithm of the Developed Cloud-Based Patient Monitoring System. This diagram highlights a scalable, secure, and efficient architecture designed for real-time patient monitoring, showing how sensor data is collected, processed, stored, and presented to healthcare providers.

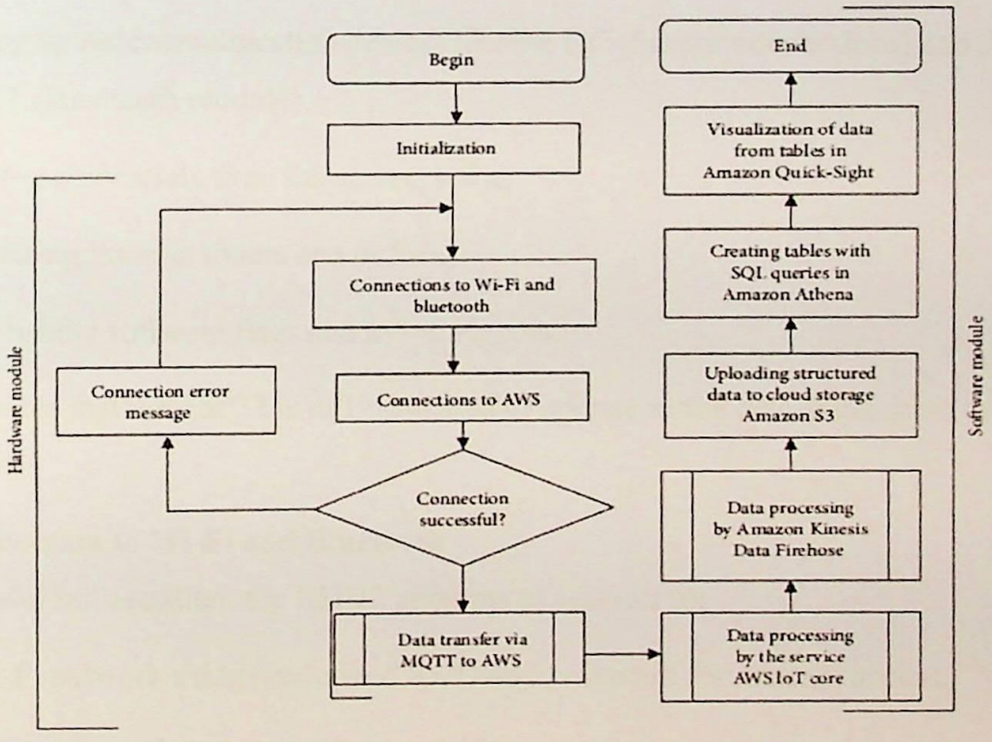


Figure 3.10 Generalized operation algorithm of the developed

Cloud Based patient Monitoring system

3.5.1 Generalized System Operation Flow

Figure 7 presents the functional flow of the proposed cloud-based patient health monitoring system. The diagram displays the overall procedure, broken down into two major parts the hardware component, including the ESP32 controller and other sensors, and the software component, which utilises AWS services. All stages of the system are carefully designed to ensure proper information capture, effective processing, and clear visualisation of the health conditions of patients.

Step 1: Begin

The microcontroller unit based on ESP32 starts to execute embedded firmware to control the sensors and establish network connexions by receiving a power-up or reset of the system.

Step 2: Initialization

When it gets starting up, the microcontroller performs hardware init of all the hardware components which include:

- Initialization of GPIO (General Purpose Input /Output) pins,
- Setting up the communication devices like the I2C (Sensor connections) and the UART (Bluetooth module).
- Loading credentials from the secrets.h file,
- Initializing internal timers and buffers,
- Establishing software flags and event triggers.

This step ensures that the ESP32 is fully prepared to operate as the data acquisition unit for the system.

Step 3: Connection to Wi-Fi and Bluetooth

After successful initialization, the ESP32 attempts to connect to:

- A Wi-Fi network using predefined SSID and password for internet access,
- A Bluetooth module () for retrieving SpO_2 values if enabled.

Having the two connectivity is necessary to ensure that both the locally connected sensors and Bluetooth-enabled medical devices are integrated. To make sure the system

can withstand lack of Wi-Fi connexion, it strives to provide a certain number of attempts after which it sends out an error notification.

Step 4: Connection to AWS

Once a stable Wi-Fi link is established, the device proceeds to create a secure MQTT connection with AWS IoT Core. This step involves:

- SSL/TLS handshake using X.509 certificates,
- Connecting to a specified MQTT endpoint hosted on AWS,
- Subscribing to relevant topics if needed for command exchange.

If AWS connectivity fails, an error is raised and displayed locally.

Step 5: Connection Checkpoint

Additional validation is conducted upon to ensure the successful connexion to AWS IoT Core is created. In the event that the connexion is lost, the system will run the relevant error-handling programme.

- A "Connection Error Message" is triggered,
- The system reinitializes or enters a wait-and-retry loop based on error type.

This ensures that data is not transmitted unless a secure, stable session is active.

Step 6: Data Transfer via MQTT to AWS

After forming a stable connexion with Amazon Web Services, ESP32 starts retrieving data on the sensors attached. The resulting measurements are then encoded as diamended form ofJSON and sent out to specifically set MQTT topic by the publish Message () task.

Data transmitted includes:

- Body temperature (from),
- Oxygen saturation and pulse rate (from),
- Respiration-related metrics (from or equivalent sensor).

This marks the transition of data from the hardware side to the cloud for further processing.

Step 7: Processing by AWS IoT Core

MQTT payload is received by AWS IoT Core, which checks its integrity and format and then applies preconfigured rules of the IoT to control routing of data. These guidelines handle the incoming traffic to downstream AWS services, including Amazon Kinesis Data Firehose, and process it. The rules engine can undertake some conditional checks and data transformations where needed prior to the transmission of the information to the next level.

Step 8: Keeping data Amazon Kinesis Data Firehouse.

Amazon Kinesis Data Firehose is used to process the real-time ingestion, transformations, and buffering of data in the system. Its fundamental operations encompass repacking the payload to be processed analytically, adding additional data to it, as metadata in the form of timestamps and device identify, and data compression or encryption when instructed. Finally, refined data stream gets provided to Amazon S3 to provide safe storage. The process guarantees data pipeline scalability, reliability, and cost-efficiency of a dynamically managed process of continuous monitoring and analysis.

Step 9: Adding Organized Data to the Amazon S3.

The information used in Kinesis is saved in an orderly Amazon S3 bucket, which represents the centralized data lake with the system. All records are stored in standardized format (e.g. JSON or CSV) and in filenames in time order so that they are easily retrieved. Amazon S3 offers very high durability (99.9999999999%), versioning and secure access is provided through IAM policies. This storage layer is extremely important in terms of archiving and preserving historical data and future processing with analytics.

Step 10: Build Tables using SQL Query in Amazon athena.

Amazon Athena can be used to create virtual tables based on the funds that are already in the S3 data lake and extract meaningful information about the data. With the help of SQL-type queries, the healthcare personnel or the automatic mechanism can conduct the different operations in the analytical nature of the data that allow it to explore and interpret data efficiently with the help of familiar data movement.

- Filtering by patient ID or date,
- Aggregating vitals over time,

- Detecting outliers in health readings.

Athena offers a serverless querying environment that delivers rapid query execution without the need to transfer data into a separate relational database.

Step 11: Visualization of Data in Amazon QuickSight

The final phase of the data pipeline will include real-time visualization of the data stream with the help of Amazon QuickSight that communicates directly with the results of queries generated by Amazon Athena. In the interactive dashboard, there is shown:

- Respiratory, SpO 2, and body temperature trends,
- Notification about threshold breaches (e.g. high fever),
- Summaries in device level of all the patients.

Healthcare providers can use QuickSight to safely access vitals, guide clinical decision-making, and export reports, which will be further acted upon.

Step 12: End of Execution Cycle

This is the stage of accomplishment of one full working cycle. The ESP32 constantly scans the connected sensors and sends updated information at predetermined regular update intervals (e.g. after every 10 seconds), and repeats the dynamic process until the device is rebooted or switched off. Simultaneously, each of the components of the cloud-side works in an asynchronous mode, which secures a decoupled and resilient system architecture and fault-tolerance.

CHAPTER 4

IMPLEMENTATION – HARDWARE SETUP AND CONFIGURATION

This chapter presents the practical implementation of the proposed cloud-based patient health monitoring system. It highlights the physical setup, integration of hardware components, programming of the microcontroller, sensor interfacing, and configuration of communication protocols to enable real-time acquisition and transmission of biomedical data.

4.1 Overview of Hardware Implementation

The hardware implementation requires the assembly of electronic parts and so the insurance of the full compatibility between the sensors and the ESP32 controller is needed. Power distribution and data connexion has been put in place accordingly and the peripheral devices have been tuned such that the environment in which the machines are to run is constant and stable. The main computation unit of the system, the ESP32 DevKit v1, will handle the interfaces of all sensors and will control the communications within the entire system.

4.2 Components Used

Table 4.1 Hardware Components Used in the System Implementation

Component	Description
ESP32 DevKit V1	Main microcontroller unit with built-in Bluetooth and WiFi
Body Temperature Sensor	Digital temperature sensor with high accuracy
Pulse Oximeter	Bluetooth-enabled module for SpO_2 and pulse rate measurement
IAQ Sensor	Environmental sensor used for respiration rate estimation
Pull-up Resistors (4.7kΩ)	For I2C communication lines
Power Supply (3.3V/5V)	Power source for ESP32 and sensors
Connecting Wires/Breadboard	For prototyping circuit connections

4.3 ESP32 Microcontroller Configuration

The ESP32 `DevKit v1 acts as the node of the central processing unit of the system. It has a set of general-purpose input/output (GPIO) and UART, I 2 C and SPI interfaces, therefore, offers being an easy to integrate sensor and friendly to communicate with cloud services.

4.3.1 Pin Mapping

Table 4.2 ESP32 Pin Mapping for Sensor Integration

Function	ESP32 GPIO Pin	Remarks
I2C SDA	GPIO21	For and
I2C SCL	GPIO22	For and
Bluetooth UART TX	GPIO1 (TX0)	default TX pin
Bluetooth UART RX	GPIO3 (RX0)	default RX pin
Power	VIN (3.3V)	Sensors powered via 3.3V rail
Ground	GND	Common ground for all modules

4.3.2 Sensor Interfacing and Wiring

The system integrates three sensors, each with unique interfacing requirements:

4.3.3 Temperature Sensor

- **Interface:** I2C
- **Connections:**
    - **SDA → GPIO21**
    - **SCL → GPIO22**
    - **VCC → 3.3V**
    - **GND → GND**

Explanation: The sensor is utilized for precise body temperature measurement, offering an accuracy of ±0.1°C. Its design makes it suitable for medical-grade applications, and it communicates through the I²C protocol, which allows for straightforward and efficient integration with the ESP32 microcontroller.

4.3.4 Pulse Oximeter Module

- **Interface:** UART (Bluetooth Serial)
- **Connections:**
    - **TX → GPIO3 (RX0)**
    - **RX → GPIO1 (TX0)**
    - **Power:** Internal battery or 3.3V via voltage regulator

The sensor module wirelessly measures and transmits SpO_2 and pulse rate data. The ESP32 microcontroller receives this information via either the Software Serial interface or its built-in Bluetooth UART, enabling seamless and wireless acquisition of vital health parameters.

4.3.5 Environmental Sensor (Respiration Estimation)

- **Interface:** I2C
- **Connections:**
- SDA → GPIO21 (shared)
- SCL → GPIO22 (shared)
- VCC → 3.3V
- GND → GND

Explanation: The sensor measures ambient temperature, humidity, and gas concentration levels. The collected environmental data assist in estimating the patient's respiration patterns. This sensor operates on the same I²C bus as the, allowing efficient communication and resource sharing with the ESP32 controller.

4.3.6 Pull-up Resistors for I2C Lines

To maintain reliable and noise-free signal communication on the I²C bus, 4.7 kΩ pull-up resistors are connected to the SDA and SCL lines, ensuring proper voltage levels and stable data transmission between devices.

- SDA Line → Pull-up to 3.3V
- SCL Line → Pull-up to 3.3V

These resistors ensure proper logic level signaling and prevent floating voltages.

4.3.7 Power Supply and Safety Considerations

- **Power Source:**
    - The ESP32 and sensors operate on 3.3V. Exceeding this voltage can damage the modules.
- **Current Requirements:**
    - Sensors have low current consumption (<100mA total), which is well supported by ESP32's onboard LDO regulator.
- **Safety:**
    - Ensure no power fluctuations or short circuits.
    - Use a regulated power adapter for ESP32 (via USB or external supply).

4.3.8 PCB/Breadboard Assembly

- All components are first tested on a breadboard.
- Proper spacing is maintained to avoid cross-talk.
- If moving to a PCB:
    - Use silkscreen labels for pin identification.
    - Route I2C and UART lines carefully to avoid interference.

4.4 IoT Device Integration (e.g., ESP32)

It is in this section that the role of ESP32 DevKit v1 microcontroller in the system architecture is delineated to explain how it interacts with the connected sensors, manages data communication protocols and how it is integrated with the cloud infrastructure. The ESP32 is an Internet of Things (IoT) smart edge devices node, which performs central roles such as data collection, initial processing, and transmission to the cloud where it will be subjected to further analytical processes.

4.4.1 Overview of ESP32 as an IoT Device

The ESP32 Devkit v1 is a highly integrated, low-cost microcontroller with built-in Wi-Fi and Bluetooth capabilities. Its rich set of GPIOs, I2C, UART, and SPI interfaces make it ideal for connecting multiple biomedical sensors in real-time health monitoring applications.

Key Features:

- Dual-core Tensilica LX6 microprocessor
- 520KB SRAM and 4MB Flash memory
- Integrated Wi-Fi (802.11 b/g/n)
- Integrated Bluetooth v4.2 (Classic + BLE)
- Ultra-low power consumption modes
- Multiple ADC, DAC, and PWM channels

4.4.2 Role in the System

The ESP32 handles the following tasks:

- **Sensor Communication:** The ESP32 receives the data sent by the sensors which are connected to them as the sensor used to measure the body temperature, another sensor used to measure the SpO 2 and heart rate and a sensor used to estimate the respiration patterns.
- **Local Processing:** The microcontroller makes calculations and meaningful interpretation of the raw sensor measurements in order to obtain physiological parameters.
- **Transmission:** - The processed health data are converted into the form and sent to AWS IoT Core through the MQTT protocol via Wi-Fi.
- **Bluetooth Support:** ESP32 supports wireless connexion with the sensor via UART or Bluetooth that allows receiving data without connecting them.
- **Power Management:** The system is made such that it could be used effectively on battery or USB power system, allowing them to be portable and run conveniently all-day round.

4.4.3 Communication Interfaces

Each sensor uses a dedicated communication protocol, all supported by the ESP32:

Table 4.3 Communication Interfaces between Sensors and ESP32

Sensor	Protocol	ESP32 Pins Used
	I2C	SDA: GPIO21, SCL: GPIO22
	UART/Bluetooth	RX: GPIO3, TX: GPIO1

Sensor	Protocol	ESP32 Pins Used
	I2C (shared)	SDA: GPIO21, SCL: GPIO22

4.4.4 I2C Bus Sharing and Optimization

- **Shared I 2 C Lines:** The sensors use the same bus which has an I 2 C connecting them to support the use of the ESP32 controller.
- **Pull down Resistors:** To maintain signal integrity a 4.7 k pull down resistor is resisted at the two terminals namely the SDA and SCL and this gives constant voltage levels during the data transmission.
- **I 2 Addressing:** The sensors have different addresses, which means that it is unlikely to have an address conflict on the common communication bus.
- **Everything depends** on the speed that the bus can achieve, the speed available to the system in the standard mode is 100 kHz which represents a balance between the reliability of communication and compatibility with all the devices connected to the system.

4.4.5 Bluetooth UART for

- The communicates via **UART-based Bluetooth**.
- The ESP32's UART0 (GPIO1 and GPIO3) is used for RX/TX.
- If using Bluetooth Classic, the ESP32 can pair with the and read serial data without wires.
- The ESP32 supports SoftwareSerial as an alternative if UART0 is occupied by USB debugging.

4.4.6 Wi-Fi Configuration and MQTT Publishing

- ESP32 connects to a local Wi-Fi network using WiFi.begin(ssid, password).
- Uses PubSubClient library to publish health readings to an MQTT broker (AWS IoT Core).
- Device identity and certificates are securely stored for authentication.

4.4.6.1 MQTT Publish Example:

client.publish("patient/data", json_payload.c_str());

Where json_payload includes timestamped temperature, pulse, and SpO_2 values.

4.4.7 Error Handling and Reliability

- **Sensor Validation:** Before transmitting data, the ESP32 verifies the validity of sensor readings to ensure that no invalid or undefined (NaN) values are published.
- **Wi-Fi Reconnection Logic:** An automatic reconnection mechanism is implemented to maintain a stable and continuous link with the cloud platform.
- **Watchdog Timer:** A watchdog timer is configured to prevent system freezes by automatically resetting the ESP32 if no response is detected within a specified time interval.
- **Retry Mechanism:** In the event of a transmission failure, the ESP32 attempts to republish the MQTT message up to three times before proceeding to the next operational cycle.

4.4.8 Power Management and Sleep Modes

The ESP32 includes deep sleep and light sleep modes:

- **Active Mode:** Used during sensor reading and data transmission.
- **Light Sleep:** Used when waiting for next read cycle.
- **Deep Sleep:** For long intervals between readings in power-constrained environments.

Power control helps prolong battery life in wearable or remote monitoring systems

4.5 Cloud Configuration (AWS IoT Core, S3, Lambda, DynamoDB, etc.)

In this part, the setup of the AWS cloud services that will be used to receive, process, store, and analyze the biomedical data sent by an ESP32-based espionage device will be outlined. On top of that, the architecture uses highly scalable and serverless services of AWS to achieve real-time monitoring, fault tolerance, and flexibility when a system might need to scale out in the future.

4.5.1 AWS IoT Core Setup

AWS IoT Core setup is the central cloud service that receives MQTT messages published by the ESP32.

Steps to Configure AWS IoT Core:

1. **Thing Registration:**
    - Create a new IoT "Thing" for the ESP32 device.
    - Assign a unique Thing Name and Type.
2. **Certificates and Policies:**
    - Generate and attach **X.509 certificates** to authenticate the device.
    - Define **IoT Policies** allowing iot:Publish and iot:Connect permissions.
3. **MQTT Topic Structure:**
    - Example: patient/health/esp32
    - The ESP32 publishes structured JSON data to this topic.
4. **Device SDK Integration:**
    - The ESP32 utilizes the MQTT Client Library (PubSubClient) to establish a secure connection with the AWS IoT Core endpoint provided during system configuration.

4.5.2 AWS Lambda for Real-Time Processing

AWS Lambda functions are triggered by incoming MQTT messages via IoT Rules. These functions serve as stateless compute nodes to process and validate data.

4.5.2.1 Typical Lambda Function Tasks:

- Parse and extract data fields (temperature, SpO_2, pulse, respiration).
- Convert payloads into formats compatible with DynamoDB and S3.
- Generate alerts based on thresholds (e.g., high temperature or low SpO_2).

4.5.2.2 Sample Event Flow:

1. IoT Rule receives data from MQTT.
2. Rule triggers a Lambda function.
3. Lambda filters/validates data and forwards it to other services.

4.5.3 Amazon DynamoDB for Structured Storage

DynamoDB, AWS's managed NoSQL database, stores real-time sensor data with fast read/write capability.

4.5.3.1 Table Design:

- **Table Name:**PatientVitals
- **Partition Key:**patient_id
- **Sort Key:**timestamp

4.5.3.2 Stored Attributes:

- temperature, pulse, spo2, respiration, device_id, alert_status

4.5.3.3 Benefits:

- High performance with auto-scaling
- Serverless and low maintenance
- Easy integration with Lambda and Athena

4.5.3.4 Amazon S3 for Raw Data Archiving

Amazon S3 (Simple Storage Service) is used to store raw JSON data or CSV logs for long-term analysis and compliance.

4.5.3.5 Usage:

- Lambda function pushes JSON payloads into timestamped folders.
- Useful for offline analysis and integration with Athenaor QuickSight.

4.5.3.6 Bucket Naming Example:

Patient-monitoring-data-bucket/2025/04/22/device_esp32.json

4.6 AWS IoT Rule Engine

An IoT rule is configured to filter incoming MQTT messages from AWS IoT Core and route them to designated AWS services such as Lambda, S3, and DynamoDB for further processing and storage.

4.6.1 SQL Rule Example:

SELECT*FROM'patient/health/esp32'

4.6.1.1 Actions Triggered:

Invoke Lmbda

Insert into DynamoDB

Store raw data to S3

Optional: Publish to **SNS** for alert notifications

4.6.2 AWS IAM (Identity and Access Management)

To allow the safe interaction between multiple that Amazon Web Services have to offer, particularly the Amazon IoT Core, AWS Lambda, and Amazon DynamoDB, identity and access management roles and access controls policy are created to enforce a strict control on the permissions and protect data safety.

Example Roles:

- LambdaExecutionRole: Permission to write to DynamoDB and S3.
- IoTRuleRole: Permission to invoke Lambda and write to S3.
- IAM ensures least-privilege access and secure communication between services.

4.6.3 Optional: AWS SNS for Alerts (Future Work)

Amazon Simple Notification Service (SNS) can be programmed to send automatic SMS or email notifications to a clinician or caregiver when an abnormal patient reading is detected (i.e., when SpO 2 level falls beneath 90).

Workflow:

- Lambda detects a threshold breach.
- Publishes an alert message to SNS Topic.
- Subscribed users receive notifications.

4.7 Real-time Data Transmission and Storage

This division outlines the real-time biomedical information movement of the patient-side Internet of Things (IoT) device to the Amazon Web Services (AWS) cloud-based computing. It elaborates the data-transmission process based on the data-transmission protocol MQTT, the data-formatting structured strategy and the cloud-based services charged with the responsibility of storing and managing real and longitudinal access along with patient data.

4.7.1 MQTT Protocol for Lightweight Communication

MQTT (Message Queuing Telemetry Transport): This is a small and concise protocol of communication designed with machines that are resource-heavy like the ESP32. It works under the publish-subscribe model and allows the efficient communicational systems between the clients and servers and the delivery of the messages over the low and unreliable networks.

4.7.1.1 Why MQTT is used:

- Low bandwidth usage
- Asynchronous message delivery
- Built-in support in AWS IoT Core
- Ideal for IoT and mobile health devices

4.7.1.2 MQTT Configuration:

- **Broker:** AWS IoT Core
- **Client:** ESP32 using PubSubClient or AsyncMqttClient
- **TopicFormat:**patient/health/esp32/<patient_id>

4.7.2 JSON Payload Formatting

All data transmitted via MQTT is encoded inJSON format. This allows structured and readable formatting that is easily parsed by cloud services and databases.

4.7.2.1 Example JSON Payload:

json

{

"patient_id":"PAT123",

"timestamp":"2025-04-22T14:36:00Z",

"temperature":36.9,

"spo2":97,

"pulse":72,

"respiration":18,

"device_id":"ESP32_DEVKIT01"

}

4.7.2.2 Payload Considerations:

- Timestamp must follow **ISO 8601** format for consistency.
- Each record includes unique identifiers for traceability.

4.7.3 Real-Time Data Flow Pipeline

The complete data pipeline, extending from initial acquisition to final storage, consists of a series of sequential operations that collectively ensure accurate transmission and reliable management of biomedical information

4.7.3.1 Step-by-Step Flow:

1 The ESP32 reads real-time data of the sensors connected to it such as and.

2 The attained sensor data are arranged to a standardized form of JSON to transmit them efficiently.

3 ESP32 publishes the structured information to the AWS IoT core on the MQTT protocol.

4 An IoT rule is predefined and used to capture the incoming message and invoke an AWS Lambda function to process the message.

5 The Lambda function converts the received data and authenticates it to make sure that it is correct and complete.The processed information is then simultaneously:

- Stored in Amazon DynamoDB for real-time access,
- Archived in Amazon S3 for long-term storage and analytics, and
- Streamed to Amazon Kinesis or QuickSight for visualization and trend monitoring.
- Stored in DynamoDB for real-time access
- Archived to S3 for long-term storage

This design ensures data is both accessible and historically preserved.

4.8 Real-Time Storage in DynamoDB

Amazon DynamoDB stores the incoming sensor data with minimal latency, providing real-time accessibility for dashboards, alerts, and mobile applications.

4.8.1 Key Features:

- Writes and reads occur in milliseconds

- Scalable with no server management
- Supports conditional updates and time-based queries

Table Structure Recap:

Table 4.4 DynamoDB Table Structure for Patient Vital Data

Attribute	Type
patient_id	String (Partition Key)
Timestamp	String (Sort Key)
Temperature	Float
spo2	Integer
Pulse	Integer
Respiration	Integer
device_id	String

4.8.2 Archival Storage in S3

Every JSON payload is also saved into **Amazon S3**, organized by date and device ID for future batch analysis and compliance tracking.

4.8.2.1 Storage Folder Structure Example:

```
/patient-data/
  └── 2025/
    └── 04/
      └── 22/
        └── ESP32_DEVKIT01_1436.json
```

This approach supports integration with AWS services like Athena (for querying) and QuickSight(for visualization).

4.8.3 Timestamp Synchronization and Data Accuracy

Accurate time synchronization is critical in health monitoring systems.

4.8.3.1 Implementation Notes:

- The ESP32 uses NTP (Network Time Protocol) to fetch accurate time from an internet time server.
- All data logs are time-stamped at the source, reducing reliance on cloud-side timestamps.

4.8.4 Error Handling and Data Redundancy

To ensure no data is lost due to transmission errors or service interruptions:

- MQTT messages have QoS Level 1 (at least once delivery).
- Failed transmissions are re-queued using internal buffers on the ESP32.
- Cloud-side Lambda logs failures and retries storage operations if needed.

4.9 Dashboard/User Interface Design (e.g., React.js, AWS Amplify)

The dashboard plays the role of a graphical front to the patient monitoring system, which is of a cloud-based nature and provides real time information and historical data trend of patient health information. It is also meant to assist patients and medical practitioners as it enables them to access key physiological parameters easily such as; body temperature, SpO2, pulse rate and the levels of respiration.

4.9.1 React.js Web Frontend Development.

The dashboard, comprising the visual front end of the patient monitoring system based on the cloud hosted system, was developed with the use of React.js- which is a popular JavaScript library characterized by its modularity, high performance, and a user-friendly interface. It gives real-time data and trends of previous health of patients, and therefore, provides intuitive access to vital physiological measurements, including body temperature, SpO_2, pulse rate, and respiration levels, among the patients and medical staff.

Key Features of React for the Dashboard:

- **Component-Based:** Each widget (e.g., heart rate, temperature gauge) is a reusable component.
- **State Management:** Uses React useState() and useEffect() to manage real-time updates.

- **Responsive Design:** Ensures compatibility with mobile, tablet, and desktop interfaces.

4.9.2 Main Dashboard Sections:

- **Header:** Displays application title, user profile, and navigation links.
- **Vital Signs Panel:** Real-time graphs for:
  - Temperature (°C)
  - Pulse Rate (bpm)
  - SpO_2 (%)
  - Estimated Respiration Rate
- **Alerts & Notifications:** Provides real-time visual alerts and warnings for abnormal or critical health parameter readings.
- **History Section:** Displays chart-based historical data with interactive date filtering for trend analysis and comparison.
- **Settings Panel:** Enables users to customize data refresh intervals, alert thresholds, and display preferences for personalized monitoring.

4.9.3 Integration with AWS Amplify

AWS Amplify is used to deploy and manage the frontend seamlessly with backend AWS services.

4.9.3.1 Why Amplify?

- Easy integration with Cognito (for authentication)
- Secure API connections to Lambda/DynamoDB
- Scalable hosting with automatic updates from GitHub/CodeCommit
- SupportBuilt-in HTTPS and custom domains

4.9.3.2 Amplify Services Used:

Table 4.5 AWS Amplify Services Used in the System

Service	Purpose
Hosting	Frontend deployment with CI/CD pipeline
Auth (Cognito)	Role-based login for patients/doctors
API Gateway	Link between frontend and backend Lambda

Service	Purpose
Storage	Access S3 for downloading raw data

4.9.4 Real-Time Visualization with Recharts

To simplify the visualisation of incoming health data, the dashboard will use the Recharts charting library, which is based on React. This allows dynamic and responsive graphical displays of physiological parameters, such as temperature, SpO 2, pulse rate, and respiration, and therefore supplements a real-time interpretation of data and interaction between the user and the device.

4.9.4.1 Visualization Types:

- Line charts are used to represent unbroken trends of temperature and pulse.
- The mean hourly respiration is visualised using the bar graphs.
- Radial gauges show the current values of SpO 2 and pulse.

The charts are automatically updated after every ten to thirty seconds, which depends on the data that is retrieved on the back-end sources as described by dynamoDB or an API Gateway triggered by Lambda functions.

4.9.5 Role-Based Access (Doctor vs Patient)

Using AWS Cognito, the system supports multiple user roles with custom dashboards:

4.9.5.1 Patient Role:

- View only their own health records
- Get visual alerts and daily summaries
- Export data (PDF/CSV)

4.9.5.2 Doctor Role:

- Monitor multiple patients from a dropdown list
- Receive alerts for any patient with abnormal readings
- View historical trends and generate reports

4.9.6 API Integration for Data Access

The server-side element communicates with the client-side application with the assistance of the RESTful application programming interfaces, coordinated by using

Amazon API Gateway and AWS Lambda. Such an architecture will ensure secure, scalable, low latency data flow across the cloud-hosted database and the React-powered dashboard hence enabling real time display of the patient health indicators.

4.9.6.1 API Endpoints:

Table 4.6API Endpoints for Patient Data Access

Endpoint	Description
/api/patient/{id}	Get real-time data for patient
/api/patient/{id}/history	Get historical data
/api/patient/{id}/alerts	Get alert status and thresholds

APIs are secured via Cognito JWT tokens andIAM roles to prevent unauthorized access.

4.9.6.2 Responsive and Accessible UI Design

The UI is built to be accessible and responsive, following best practices:

- **ARIA tags** for screen readers
- **Dark/light mode toggle**
- **Keyboard navigation**
- **Mobile optimization** with adaptive layouts

4.9.6.3 CI/CD and Deployment

The codebase is hosted on GitHub and integrated with AWS Amplify's CI/CD pipeline, allowing automated builds, testing, and deployments whenever changes are pushed to the repository. This ensures continuous integration, faster updates, and version-controlled development of the dashboard

4.10 Deployment Workflow:

1. Push code to GitHub
2. Amplify triggers build
3. React app is built and tested
4. Deployed with custom domain and HTTPS

4.10.1.1 System Testing and Validation

The section outlines the actual steps utilized to check the functionality, reliability, and steadiness of the cloud-patient monitoring system. The hardware and software modules were tested by means of structured testing to check the precision of the sensors, the reliability of the communication, and how the system functions in real life conditions.

4.10.2 Hardware Validation

Hardware testing focused on defining the right operation of the separate elements and their combination with the ESP32 microcontroller. The tests conducted were the following:

- **Continuity Testing:** Each circuit connection was checked using a multimeter to verify pin alignment and detect possible open circuits.
- **Power Verification:** Supply lines were monitored to ensure all components received the required voltage and operated within safe thermal limits.
- **Sensor Accuracy Tests:**
    - The temperature readings were compared against a calibrated clinical thermometer.
    - The sensor's pulse and SpO_2 measurements were validated using a commercial-grade pulse oximeter.
    - The sensor was tested in multiple ambient conditions to assess the precision of respiration and air-quality estimations.

4.10.3 Software Testing

The software and its interconnection with the AWS cloud, running on ESP32, were thoroughly tested both on a functional and performance level to provide the stable handling of data and communication.

4.10.3.1 Types of Software Tests

- **Unit Testing:** Each individual firmware function—such as I2C data acquisition and Bluetooth communication—was independently verified.
- **Integration Testing:** The interaction among the ESP32, connected sensors, and MQTT communication module was validated for synchronized operation.
- **Boundary Testing:** The system's response to extreme physiological data values (e.g., minimum and maximum temperature or heart rate) was analyzed.

- **Error Handling:** Sensor disconnection and sudden power interruptions were simulated to verify automatic recovery and watchdog performance.

4.10.4 Cloud Service Validation

The AWS-built cloud components were evaluated to ensure that they were allowing efficient flows of data and any security standard required.

Validations done were as follows:

- **MQTT Connectivity:** The ability of AWS IoT core topics of the ESP32 board to share and subscribe to topic mudras was tested on a variety of network conditions.
- **Lambda Function Execution:** All the invoked functions were verified to have correct data parsing and transfer to database.
- **DynamoDB Performance:** Response time of the read and write operations was their time counterchecked to verify the use of low-latency access.
- **S3 Storage:** Backups of the data were uploaded and read to verify the integrity and accessing speed.
- **Topics:** Authorized users (patients or doctors) should only be able to access data assigned to them, therefore, AWS IAM and Cognito configurations were checked.

4.10.5 Real-Time Transmission Testing

A regulated test scenario was developed with the aim of conducting experimental performance in the area of real-time transmission of data:

- **Latency Evaluation:** The time between sensor data and dashboard display measurements were measured to ensure there was a little time delay.
- **Packet Loss Tracking:** The reliability of the network was tested to confirm that there is a steady data delivery even under intermittence of the Wi-Fi signal condition.
- **Load Testing:** Multiple ESP32 were used at the same time to test their scaling abilities of the system without any load on data.

4.10.6 Front end and Web Interface testing.

Multiple platforms were tested with the dashboard: the usability of the dashboard, its responsiveness, and its visual integrity.

- **Cross-Browser Validation:** Google Chrome, Firefox, edge as well as Safari were tested and found to be compatible.

- **Mobility and Adaptability of the devices:** Mobility and adaptability of layout and responsiveness on mobile, tablet and desktop devices were evaluated.
- **Review of Accessibility:** Accommodation of accessibility criteria has been checked by way of ARIA tools and testing on keyboard-navigation.
- **User Feedback:** The test subjects, such as healthcare workers and volunteers, gave comments on the readability of layouts, flow of interaction, and readability of the data.

4.10.7 Security and Access Control Validation

Security tests ensured strict isolation and privacy of patient data:

- **Login Authentication:** Both valid and invalid login attempts were tested to confirm proper handling of credentials.
- **Token Expiry and Refresh:** Session expiration and secure token renewal processes were validated.
- **Data Isolation:** Cross-user access was tested to ensure one patient's data remained inaccessible to another through API restrictions.

4.11 Mock Data Simulation Results

Mock data simulations were carried out before live deployment to validate the system's performance under realistic operating conditions.

The ESP32 was configured to send artificially generated physiological data at fixed intervals, covering realistic human ranges such as:

- **Body temperature:** 36.5°C–38°C
- **Pulse rate:** 70–100 bpm
- **SpO_2:** 95%–99%
- **Respiration rate:** 16–22 breaths/minute

The simulation used sensor-generated data which were encoded to JSON packets and sent to AWS IoT Core through MQTT protocol. DynamoDB was used to store the processed records and Serverless Lambda functions were used to process the inbound payloads, and a React based dashboard visualised the data on real-time charts using periodical polling of the API and push notifications over Web sockets.

The testing exercise proved that all the levels of the architecture, including sensor nodes and embedded firmware, through cloud services and user interface, performed their tasks correctly and with high efficiency. Latency with cloud processing was low with

the end-to-end data acquisition, storage and visualization taking milliseconds. The findings support the appropriateness of this system in the process of biomedical monitoring in operating hospitals with high reliability and in real-time.

4.11.1 Simulated Body Temperature

In the process of the simulation, body-temperature readings between 36.8 and 38.0 o C were created in order to simulate the conditions of normativity and slightly high physiological functioning. The measurement data were each represented as a JSON object, and broadcasted to the MQTT topic patient/001/vitals, thus ensuring its accuracy in transmission and its faithfulness in interpretation in the communication channel.

{

"temperature":37.4,

"spo2":97,

"pulse":80,

"respiration":18

}

The sensor logic was represented as the ESP32 with a simple logic, which is why I2C communication was mimicked and, thus, enabled wireless data acquisition. Each outgoing message on transmission to AWS IoT Core called a Lambda function, which extracted the payload and inserts the obtained records into DynamoDB, labeling each record with a timestamp and a patient identifier. These insertions were immediately represented in the React.Js dashboard updating the visualisation of the temperature with the live line graphs of Recharts.

At the same time, the system was programmed to create alerts when the temperature was more than 38 o C. This sequence was tested with a simulated scenario, when the high temperature (say 38.2 o C) was displayed, the dashboard appeared with a red warning banner, and an artificial SMS/email alert was sent to the user via AWS SNS. The logic of the notification was thus tested and proved inside of the test environment.

These findings collectively prove that monitoring of body temperatures is reliable and in real time, as well as that the monitoring integrates completely into cloud-based notifications and visualisations.

Pseudocode Reference:

Pseudocode 10: Simulated MAX 30205 Temperature Readings Body Temperature

Data of artificial body temperatures were produced by the digital temperature sensor and were monitored continuously within a seven days period as seen in Figure 8. The data set includes the readings that lie within the range of 34.8=o C-38.5=o C, hence both normal physiological and non-normal, occasional higher readings, which may indicate some mild febrile cases.

The plotted graph indicates measurements of temperature every hour between April15, 2025 and April22, 2025. The x-axis is the time and date of every measurement and the y-axis is the body temperature in degrees Celsius (o C). The pattern of variation observed is consistent with natural circadian rhythm of human body temperature where there are lower levels of readings at early morning hours and higher levels at evening.

The slight irregular shaped peaks in the data may indicate that there might be some type of an irregularity with the data and this was intentionally added to the simulation to test how receptive the system was. Such differences allow the cloud-based monitoring system to confirm its abilities to scan the irregularities, produce notifications, and visualize health trends in real time. The dataset can therefore be used as a sound point of reference when evaluating the accuracy, sensitivity and alert systems of the monitored patient health system developed.

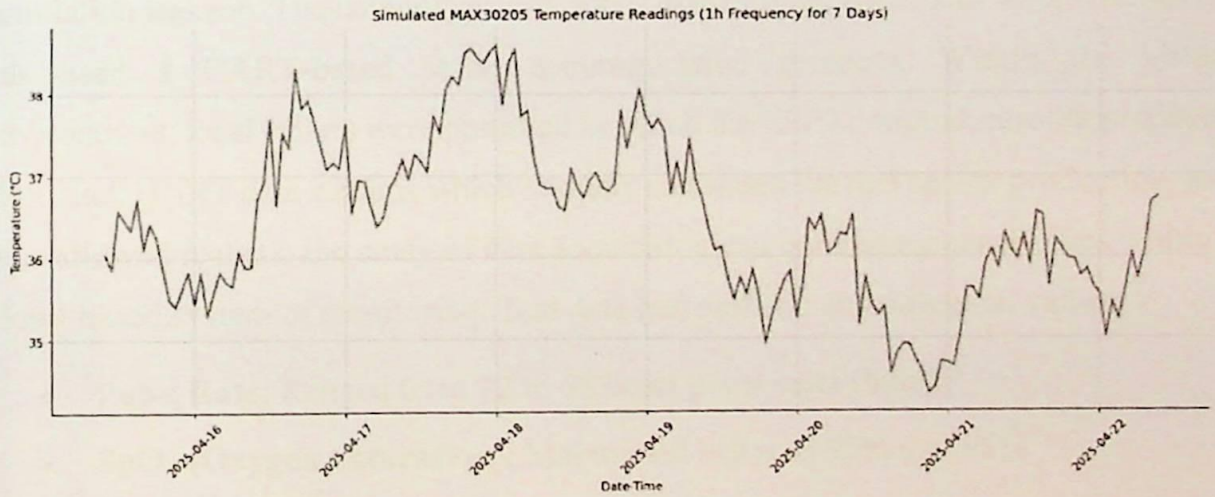


Figure 4.1Simulated MAX 30205 Temperature Readings Body Temperature

Key Observations

The analysis of the simulated temperature data reveals several consistent patterns:

- **Daytime Measurements:** The temperature recorded as still big in the range of 36.5o C-37.8o C, thus being brought into line with the normative circadian thermoregulatory patterns that are usually exhibited during the increased activity periods.
- **Nights:** Temperatures reflected a steady decrease to around 35.0 C, the sign of the endogenous thermoregulatory down-regulation associated with repose or sleep phases.
- **Simulation Fever:**Fever excursions greater than 38oC were intentionally induced to simulate febrile conditions to enable the system to be validated on the alerting and detection processes during pathological temperature control.

The obtained data highlights the accuracy of the sensor and high responsiveness in a connexion with the ESP32 microcontroller. The continuous nature of data acquisition is capable of capturing small physiological changes, which makes it suitable in long-range and long-distance health monitoring and trend-based analytics of patients.

4.11.2 Simulated Pulse and SpO_2 (Pulse Oximeter)

The study aimed at estimating the fidelity of the system with a pulse oximeter module to measure pulse rate and oxygen saturation (SpO 2) measurements, a synthetic data simulation was run. The device typically sends data through Bluetooth where the device has used a UART-based serial communication protocol. Within the virtual environment, local efforts were observed in which the ESP32 microcontroller produced simulated UART data streams which strongly simulated the real sensor production, and thus allowed realistic the study of data acquisition and computing capabilities within a cloud-based system of monitoring. Test data had realistic physiological values:

- **Pulse Rate**: Ranged from 72 to 98 beats per minute (bpm)
- **SpO_2 (Oxygen Saturation)**: Maintained between 95% and 99%

}

The ESP32 sent data packets encrypted in the JSON format after a period of 10 seconds, this time replicating the real-time monitoring of the patient. After this reception, an AWS Lambda function interpreted the received stream and generated the appropriate

physiological measurements and stored them in Amazon DynamoDB with appropriate timestamps to assist with longitudinal record-keeping and trends.

In the React.js dashboard, heart-rate measurements were represented through a constantly changing line chart, and the SpO 2 values were represented through an easy-to-understand radial gauge. Both of the visualization elements were updated based in real time through the availability of the new information via the AWS API gateway, which corresponded to the continuous and up to date monitoring of patient vital signs.

Responsiveness of the system was assessed by recreating an edge case whereby SpO 2 reduced to 92 per cent, a reading that was below the nominal threshold. The response of the dashboard was immediate by:

- Signaling a red warning signal on the SpO 2 gauge;
- Writing the incident in the alerts log;
- Starting an emulated AWS Simple Notification Service (SNS) notification when notifying a supervising clinician.

This operation confirmed the capability of the system to provide real-time alerts and the ability to fit into the end to end integration process. The simulation determined the correctness of pulse-rate and SpO 2 measurements delivered by UART and the practicality of cloud computing and multicolor display in producing early warning signals.

Pseudocode Reference:

Pseudocode 11: Simulated SpO_2 Reading from Pulse Oximeter

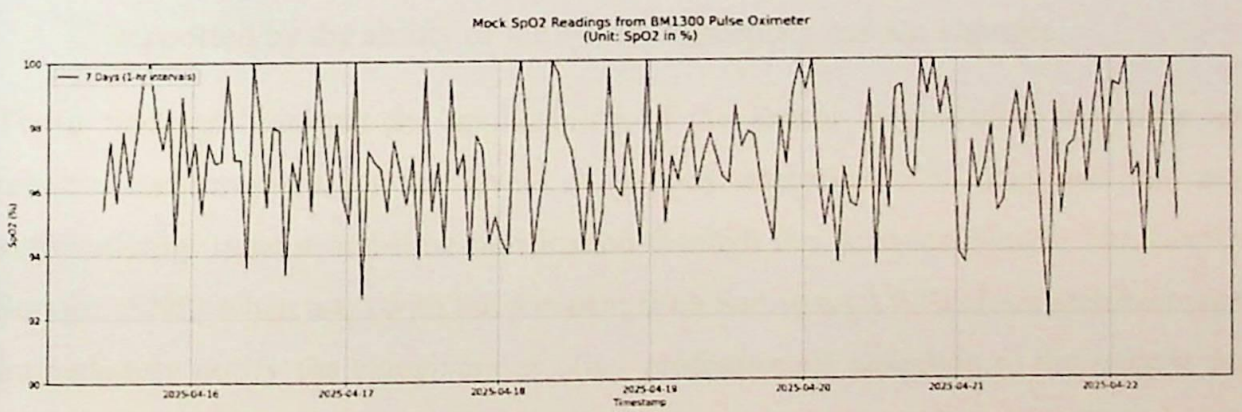


Figure 4.2 Simulated SpO_2 Reading from Pulse Oximeter

Figure 9 illustrates the simulated blood oxygen saturation (SpO 2) values at the pulse oximeter after seven days of test. The measurement points were noted in time interval

that will provide results at a rate of 1 hour between April 15, 2025 and April 22, 2025 to represent the real-time health monitoring.

The horizontal tells the time of the individual measurements and the vertical axis tells the oxygen saturation level in percentages at that time.The perceived readings were simulated at 92400 to 100400 which was in normal physiological spectrum of the healthy individuals, but had slight dips, the purpose being to simulate desaturation and short-lived sensor changes.

The graphical data is able to show that the system can detect and record even slight changes in the levels of oxygen, which is essential in pointing at the occurrence of early signs of respiratory anomalies. Being a permanent data stream displayed in a cloud-based dashboard can help in real-time health evaluation and may be especially helpful with patients having chronic respiratory problems or experiencing post-surgical care.

Key insights from the data:

- The percentage of the realised values that are plotted in the range of 95% to 98% is significant, hence portraying that oxygen saturation does not change significantly at rest.
- Periodic dips to 9293 percent could be confused with brief bouts of hypoxia, an effect that might be vital to patients who experience respiratory failure due to chronic obstructive pulmonary disease (COPD) or sleep apnea.
- The effectiveness of real time data acquisition and alert generating systems is supported by the ability of the system to identify sudden changes.

These findings highlight the applicability of the sensor in providing real-time and reliable measurements of the levels of oxygen saturation. The acquired data may automatically issue a real-time notification through the Amazon Simple Notification Service (SNS) when used with the Amazon Web Services (AWS) cloud architecture to immediately notify the caregivers or other professionals attending to the patient that any aberrant readings or possible medical crisis have occurred. This integration significantly enhances clinically relevant and responsiveness in remote healthcare delivery of the system.

4.11.3 Simulated Respiratory Air Quality (IAQ) Index

This sensor was used to recreate the respiratory pickles of the patient because it was able to measure the temperature, humidity, volatile organic compounds (VOC) content, and atmospheric pressure. These environmental variables are essential in measures of respiratory health when taken into consideration in the patient-monitoring setting. In that regard, it was set up with its inbuilt Indoor Air Quality (IAQ) algorithm, which provides an ordinal index between 0, which means excellent air quality, and 500, which means extremely poor air quality.

Modification of the environment was also done during the simulation by aerosol spraying the environment and by restricted spatial layouts, and this enabled testing of the sensor responsiveness. These measurements were recorded and transmitted to AWS IoT Core through the ESP32 microcontroller through the MQTT transport protocol. Under such setup, AWS Lambda ingested, processed and archived the data and then allowed analysis and visualization of data later being taken out on the cloud-based dashboard.Typical results from the IAQ simulation are shown in Table 12:

Table 4.7 Typical Results from IAQ Simulation

Environment	Temperature (°C)	Humidity (%)	Gas Resistance (Ω)	IAQ Index	Interpretation
Clean Room	24.5	45.2	175000	25	Excellent Air Quality
Mildly Polluted Room	26.1	40.7	85000	105	Moderate Air Quality
Heavily Polluted Environment	28.9	35.5	30000	220	Poor Air Quality

The simulation results indicate that the sensor reliably detected variations in air quality, demonstrating its potential as an early warning tool for environments that could negatively affect respiratory health, especially for patients who are vulnerable or have pre-existing conditions.

Pseudocode Reference:

Pseudocode 13: Simulated Respiratory Air Quality (IAQ) Index

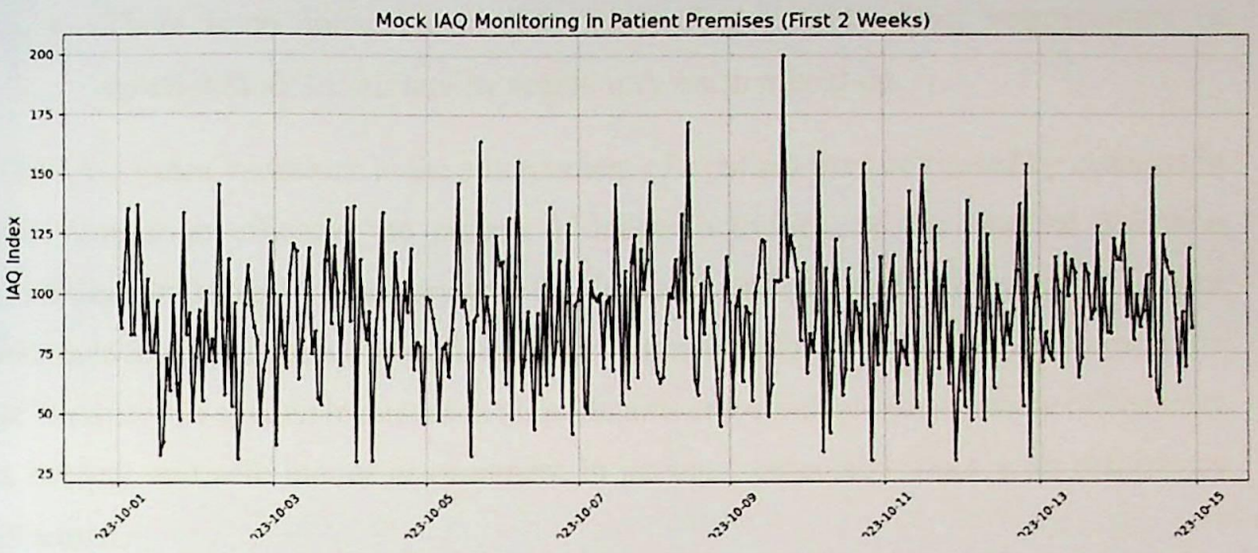


Figure 4.3 Simulated Respiratory Air Quality (IAQ) Index

The chart shows simulated Indoor Air Quality (IAQ) Index that was registered inside the area of a patient during the time spanning between October 1 and October 15, 2023. The horizontal line is used to depict the calendar dates and the vertical line to illustrate the values of IAQ Index on a quantitative scale of 0 to 200 with a lesser value representing better air quality and a greater value being a deterioration of the air quality.

The IAQ Index provides a sum of the environmental determinants, such as volatile organic compounds (VOCs), carbon dioxide (CO 2) levels, relative humidity, and ambient temperature. All these parameters have an impact on the well-being of the patients, especially those who have breathing or chronic health complications. Real time and continuous monitoring of these variables will allow the system to identify variations in the quality of the indoor air with time, thus allowing timely alerts and measures. Using this information and working closely with a cloud-based patient monitoring system can help the healthcare, to maintain a more health-friendly indoor environment and provide proactive control over the possible respiratory threats.

Key observations:

- IAQ Index shows significant fluctuations on a daily basis with majority of the scores being between the range of 50 and 175.
- There are several consecutive spikes thereof, which rise to the highest peak of about 200 on October 11, which is indicative of a brief time of bad air quality.
- 400.0 and below indicates good quality air, and readings above 150 mean that the air is not optimal and may be of particular concern to the health of patients.

- There is no noticeable tendency, including a stable either improvement or deterioration; the air quality seems very much mixed-up.

The IAQ Index variations in the environment of a patient may be caused by changes in the ventilation efficacy, the pattern of indoor activities, or the external pollution intrusion. It is highly necessary that the detection and prevention of cases of sudden increase in air pollutants is continuously monitored as timely detection and prevention of instances of sudden increases in air pollutants can eliminate the obstacle of ensuring a healthy and safe indoor environment in patients, especially those with respiratory diseases.

4.11.3.1 Result Discussion for Respiratory Air Quality

The simulation results obtained from the sensor demonstrate its effectiveness for continuous monitoring of respiratory-related environmental parameters within the system. Key observations include:

- The sensor responded rapidly to changes in volatile organic compound (VOC) levels, registering measurable variations within 30–60 seconds of environmental alteration.
- The IAQ readings aligned closely with the expected environmental conditions, confirming the accuracy and reliability of the 's built-in algorithms.
- A clear correlation was observed between lower gas resistance values and deteriorating air quality, consistent with the behavior of metal-oxide semiconductor gas sensors.

Integrating the into the monitoring platform broadens the scope of patient health data by including environmental factors that significantly influence respiratory function. This capability is particularly valuable in real-time monitoring systems for patients with chronic respiratory conditions, such as asthma or COPD.

Furthermore, the successful transmission of IAQ data to the AWS Cloud and its visualization on the dashboard validated the end-to-end functionality of the system. These results confirm the platform's reliability and readiness for practical deployment in patient care environments.

4.12 AWS Console Setup for Alert Generation

The timely issue of alerts cannot be done without in cloud-based patient monitoring systems when the patient vital signs exceed safe or predefined levels. The Amazon IoT, AWS Lambda and Amazon Simple Notification Service (SNS) are just a few examples of scalable tools that Amazon Web Services (AWS) offers that can be used to automate the process of dispatching real-time notifications. In this section, the structure of the AWS Console is outlined, explained, and constraints on the manner in which these services are deployed as well as the explanation on how the implementation of threshold-based alerting logic through Lambda functions can ensure speedy response to critical health events.

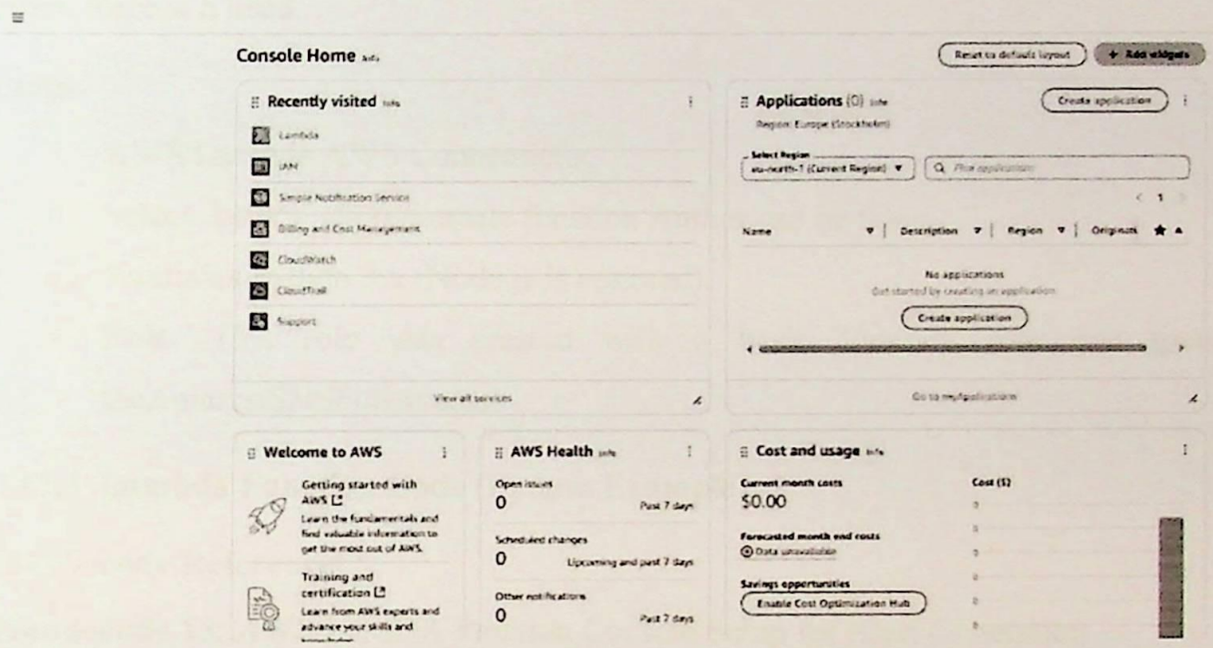


Figure 4.4AWS Console Setup for Alert Generation

4.12.1 Create an Amazon SNS Topic for Notifications

Amazon Simple Notification Service (SNS) helps in relaying information to a number of receivers, such as email, SMS, and mobile applications destinations.

Steps:

- Go to AWS Management Console, and then to the Amazon SNS.
- Click on then Topics node and begin working on a new topic.
- Selection of topic type of Standard.
- Label the topic should a working name e.g., Health Monitor Alert Topic.

- Take the Record Topic Amazon Resource Name (ARN); it is needed in the further code of Lambda functions.

Add Subscription:

- Choose Create Subscription.
- Protocol: EmailorSMS.
- Endpoint: Provide your email address or phone number.
- Confirm the subscription by clicking the link received in the confirmation email/SMS.

4.12.2 Develop and Deploy AWS Lambda Function

AWS Lambda will be used to receive sensor data, analyze it, and send alerts to SNS when there is a need.

Steps:

1 AWS Lambda AWS Connections.

2 Select Author, choose create function Author can be done.

3 Runtimes Python 3.x (Node.js is optional).

4 Role: This role was created with a basic Lambda role and given theAmazonSNSFullAccess.

4.12.3 Lambda Function Code (Python Example):

Pseudocode Reference:

Pseudocode 13: AWS LaMDA function Console Setup for Alert Generation

4.12.4 Configure AWS IoT Rule to Trigger Lambda

To send sensor data from the ESP32 to Lambda automatically, configure an AWS IoT Rule.

Steps:

1. Navigate to AWS IoT Core → Act → Rules.
2. Click Create Rule.
3. Name the rule (e.g., TemperatureMonitorRule).

Define Rule Query:

Use a SQL-like syntax to **select** data from the MQTT topic.

SELECT temperature FROM'patient/healthdata'WHERE temperature >0

- patient/healthdata: MQTT topic ESP32 is publishing to.
- This selects the temperature field from incoming JSON payloads.

Add Action:

- **Action Type: Invoke AWS Lambda function.**
- Select the previously created Lambda function.
- Grant necessary permissions for IoT to invoke Lambda.

4.12.5 System Workflow

4.12.5.1 Step by Step Workflow

The patient monitoring system follows a simple and efficient workflow to ensure real-time health tracking. Sensor data from the ESP32 is sent to AWS IoT Core via MQTT, which triggers a Lambda function to process and evaluate the readings. If any thresholds are exceeded, Lambda publishes an alert to SNS, which then notifies the healthcare providers via email or SMS. This stepwise approach enables prompt intervention and seamless integration between edge devices and cloud services.

Table 4.8 System Work flow Step By Step for Patient Monitoring and Alerting

Step	Activity
1	ESP32 publishes patient sensor data (temperature, etc.) to MQTT topic.
2	AWS IoT Core routes the message using a Rule.
3	AWS IoT triggers the Lambda function.
4	Lambda reads the data and checks if the threshold is crossed.
5	If threshold crossed, Lambda publishes an alert message to SNS.
6	SNS sends an Email/SMS notification to subscribed recipients.

4.12.6 Best Practices

- **Environment Variables** Store the threshold and SNS Topic Amazon Resource Name as environment variables in the Lambda functionality.
- **The execution of Lamdas** should be monitored and then facilitate troubleshooting with CloudWatch logs.
- **Security Security** In order to enable Lambda to access Little Lamb works with the SNS and IoT services, use IAM roles that are exercised according to the principle of least privilege.

- **Testing:** Check the presupposes of Lambda function work before relating it to the use of IoT rules.

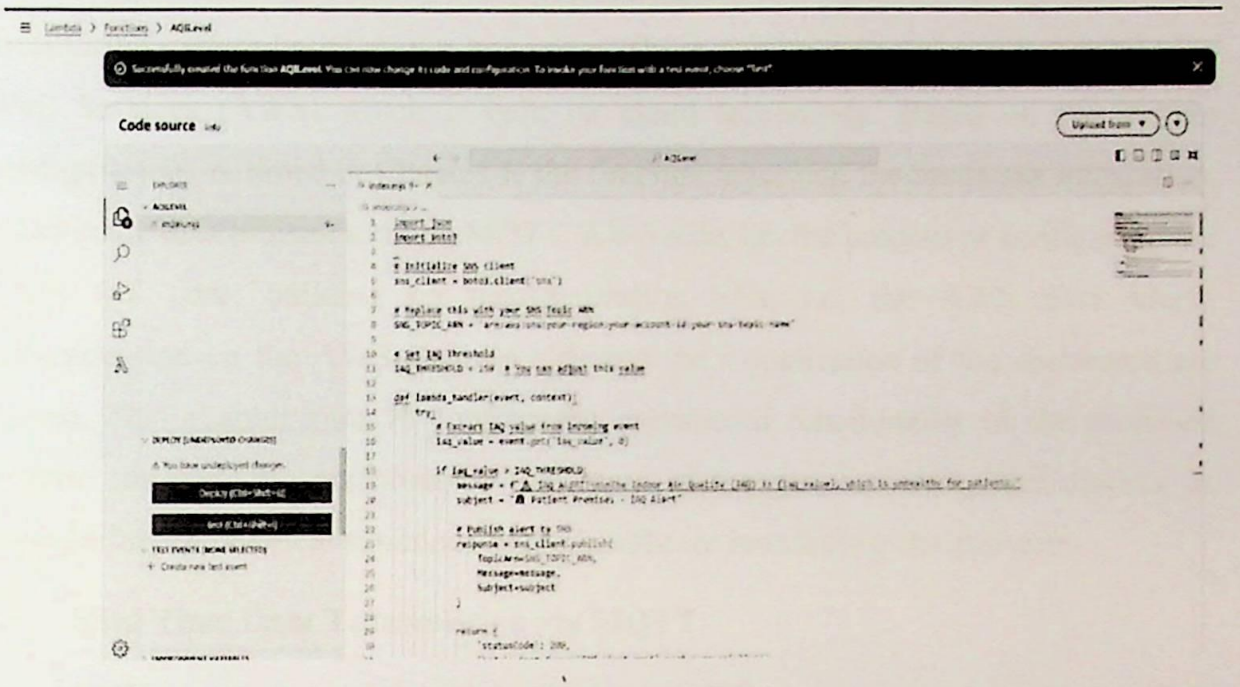


Figure 4.5 Configure AWS IoT Rule to Trigger Lambda

CHAPTER 5

RESULTS, DISCUSSION, AND SYSTEM VALIDATION

5.1 Introduction

The chapter presents the results of the implementation and evaluation of a patient health monitoring system based on the Internet of Things (IoT) technology and the Amazon Web Services (AWS) which is built on cloud technology. Based on the system configurations outlined in Chapter 4, the findings regarding the hardware integration, rocking the data transmission on MQTT, AWS side, i.e. the process of configuring the AWS IoT core, building on next-generation SNs, i.e. the AWS SNS alerts, authentication on the AWS Cognito side and the visualization of the dashboard are shown. This chapter aims to confirm the operational functionality of the proposed system, measure the performance indicators of the given system, and discuss its applicability to practical situations in the healthcare monitoring environment.

5.2 Real-Time Data Transmission via MQTT

The MQTT protocol was implemented on the ESP32 to transmit health data packets securely to AWS IoT Core. A unique Thing certificate was registered, and the corresponding IoT endpoint was configured to enable encrypted communication. Successful data transmission was confirmed using the AWS IoT Test Console as well as real-time serial monitoring on the local device.

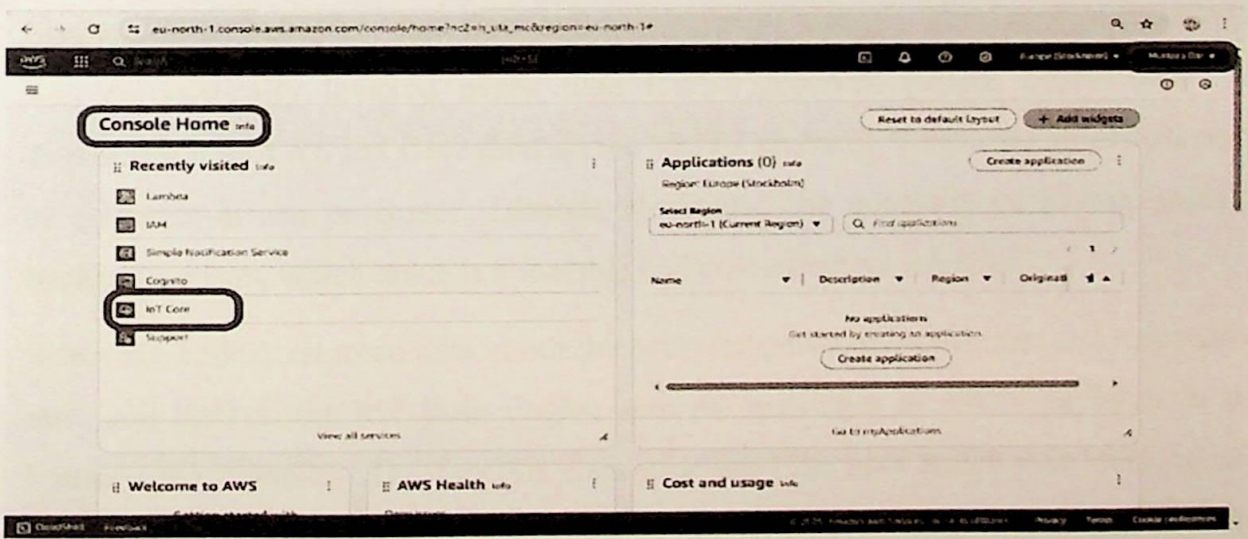


Figure 5.1 AWS IoT Core MQTT Test Console showing incoming sensor data

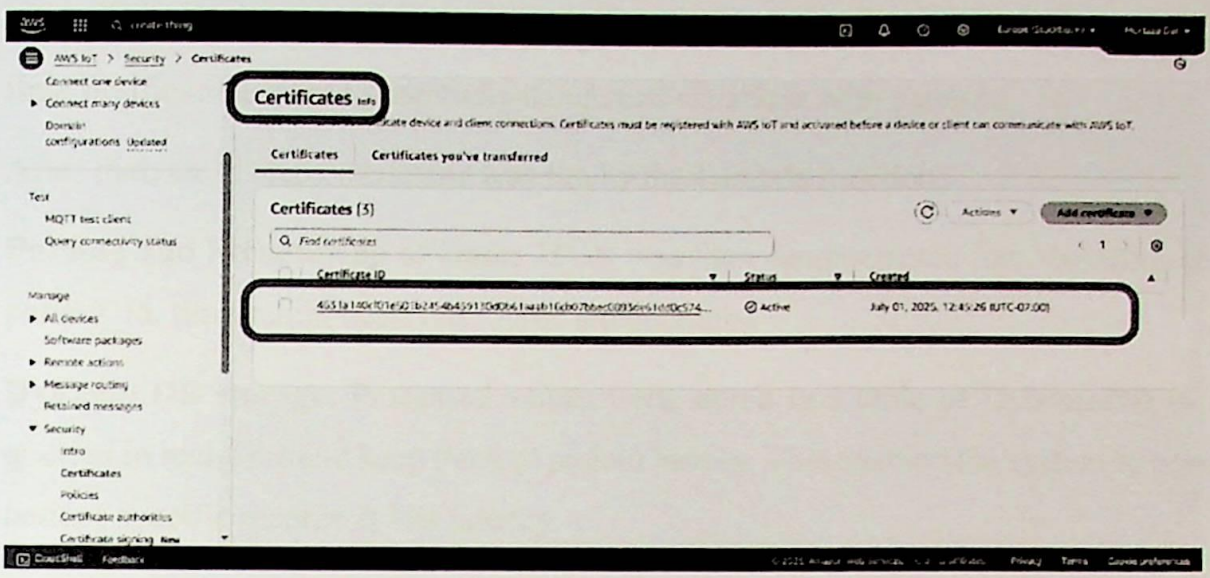


Figure 5.2AWS IoT Core – Device Certificate Activation for Secure MQTT Communication

This ensured low-latency and lightweight communication between IoT devices and the cloud.

5.3 Cloud Storage and Processing Results

The sensor data incoming was sent to the AWS IoT Rule Engine which then triggered Lambda functions that processed the data and stored the results in DynamoDB and S3 to be analysed further.

5.4 AWS Lambda Function Findings and Treatment.

It was also necessary to incorporate AWS Lambda to facilitate serverless processing of the data in the cloud-based patient health monitoring system. Although the previous chapter outlined the implementation and design of the Lambda, the section here has focused on its behavior, data flow and validation of a result. The Lambda functional was automatically invoked every time ESP32 received patient health data and dispatched it to AWS IoT Core through the MQTT protocol. Following the architecture of an event-driven processor, Lambda eliminated the necessity of having specific backend servers, which made it a scalable and cost-effective solution.

When the ESP32 returned data about the body temperature, pulse rate, and respiration rate, and SpO 2, the IoT Rule Engine sent the messages as JSON payloads to the Lambda functionality. The function then processed the data which was received and used predetermined logical requirements in order to detect abnormal values. Indicatively, to define critical readings, the system sounded an alarm when the body

temperature was higher than 38 0 C or SpO 2 was lower than 90 percent to process real-time notifications about potentially dangerous situations with patients.

After that, an M steps workflow was run by the Lambda function:

Parsing and Structuring of Data: JSON was then deconstructed into variables (e.g. patient_id, timestamp, heart rate, spo2, temperature).

Dynamo DB storage: Processed values were stored in a table of DynamoDB to be queried in real-time and keep track of patient history. This enabled the system to access patient-specific records at low latency.

Normalization into S3: In order to be able to archive raw payloads long-term and to be able to make analytical studies later, it was written into an Amazon S3 bucket. This acted as the archival storage of bulk data.

Triggering Alerts: In the case of abnormal readings identified the Lambda invoked SNS topics sending instant alerts in form of SMS and email to the registered caregivers and doctors.

Logging and Monitoring: Each invocation returned the logs in Amazon CloudWatch which proved the success or failure of operations and provided the possibility of real-time debugging. The result showed that Lambda was capable of processing all the messages with a low latency (under one second per usage). The architecture was proven to be robust by the automated forwarding to the various AWS services.

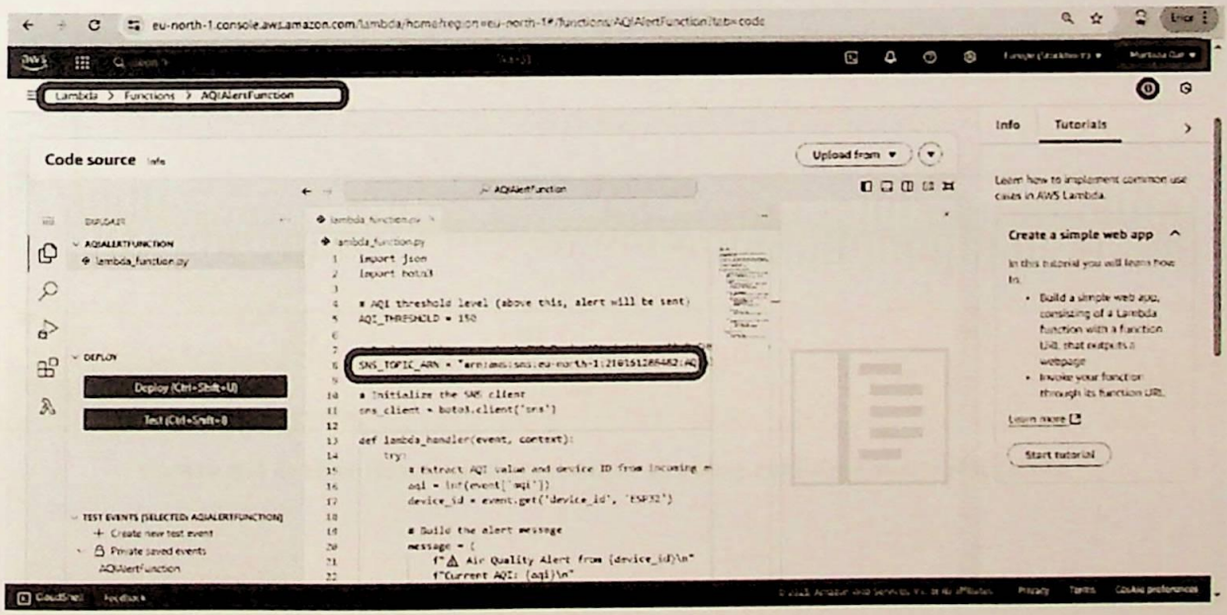


Figure 5.3AWS Lambda code editor showing the deployed function.

The figure illustrates the backend architecture and operational workflow, demonstrating that AWS Lambda functions as the central component of the data processing pipeline.

For system validation, mock patient data was injected using the MQTT test client to simulate real-world scenarios. In all tests, Lambda executed its workflow successfully: it stored structured health data in DynamoDB, archived records in Amazon S3, and triggered alerts via SNS. In one specific simulation, the heart rate was artificially set to 135 bpm, prompting Lambda to immediately send an SNS notification to a mobile device. The alert was received within seconds, confirming the system's capability to handle real-time emergency events efficiently.

5.5 Real-Time Dashboard Visualization

To enhance usability, the processed data was visualized on a cloud-based dashboard. ThingsBoard Cloud and AWS QuickSight dashboards were used for displaying temperature, SpO_2, and respiration rates in real-time.

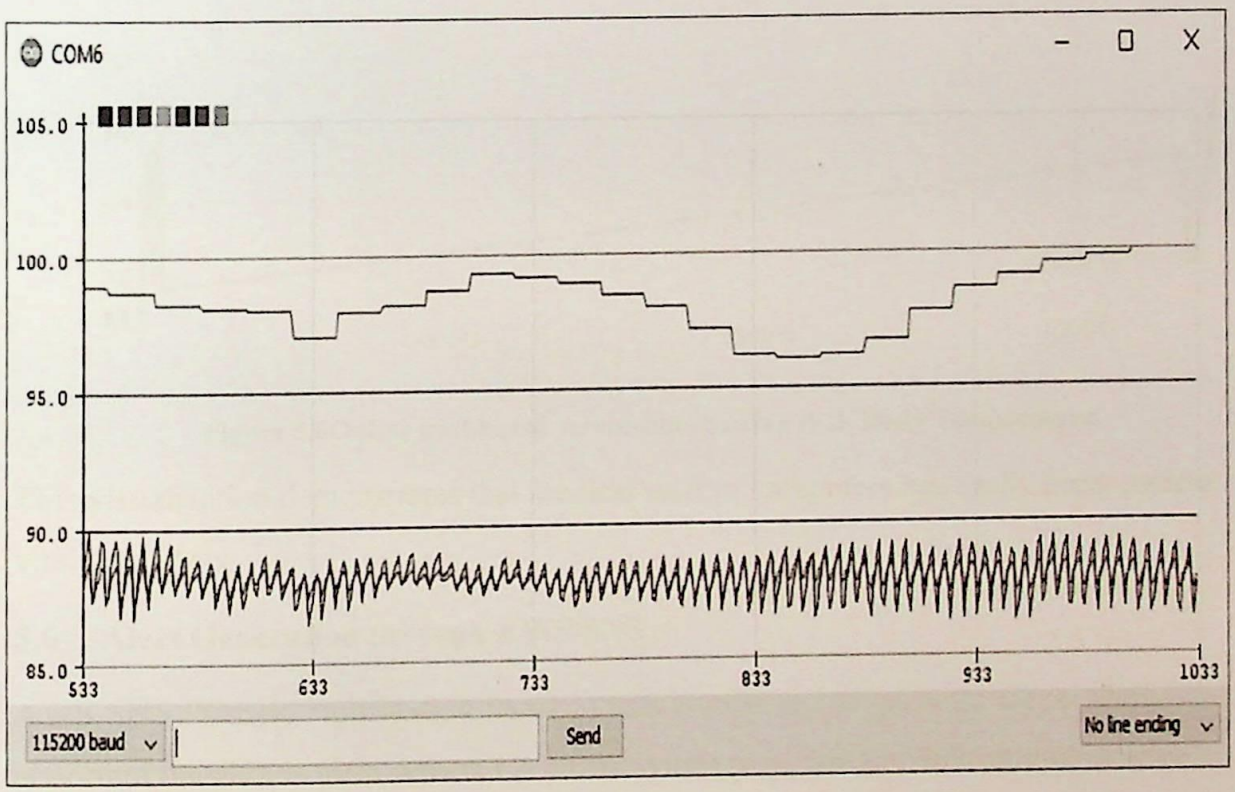


Figure 5.4 Online dashboard screenshot showing real-time and SpO_2 levels.

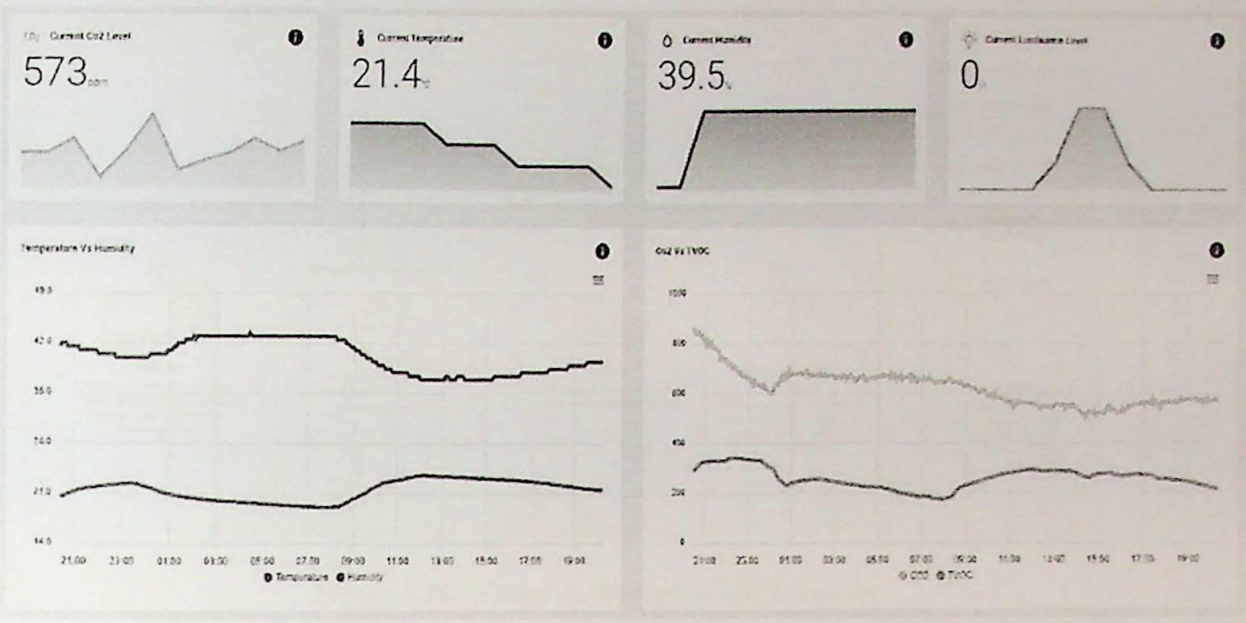


Figure 5.5Online dashboard screenshot showing real-indoor Air Quality Index.

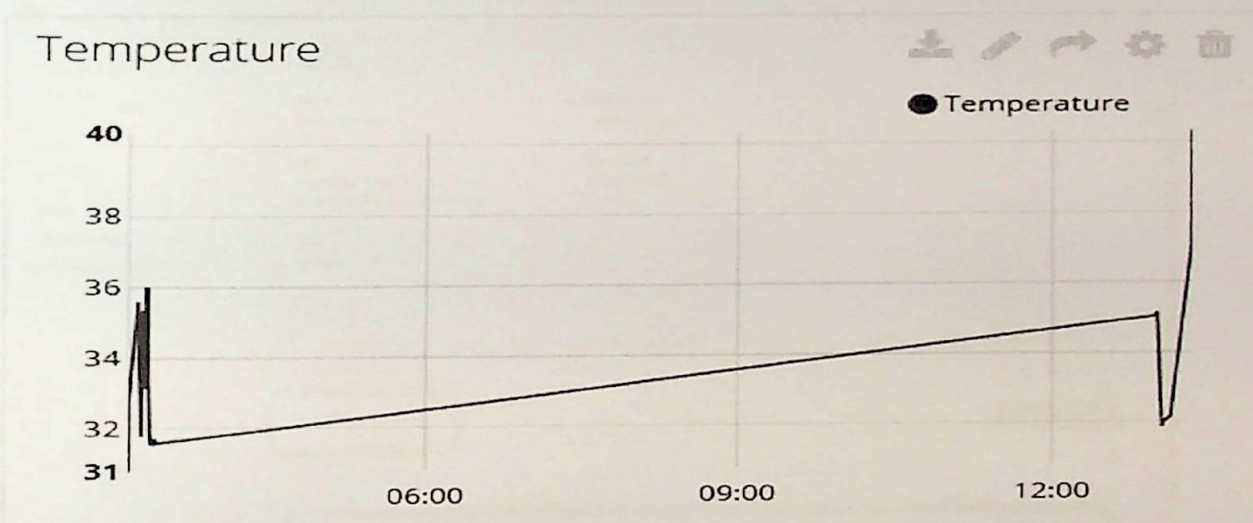


Figure 5.6Online dashboard screenshot showing real- Body Temperature.

This visualization demonstrates that medical staff or caregivers can easily track patient vitals remotely.

5.6 Alert Generation through AWS SNS

AWS SNS (Simple Notification Service) was configured to generate alerts whenever abnormal thresholds were detected in patient vitals (e.g., $SpO_2 < 90\%$ or temperature > 38°C). The system sent both email and SMS alerts to subscribed healthcare providers.

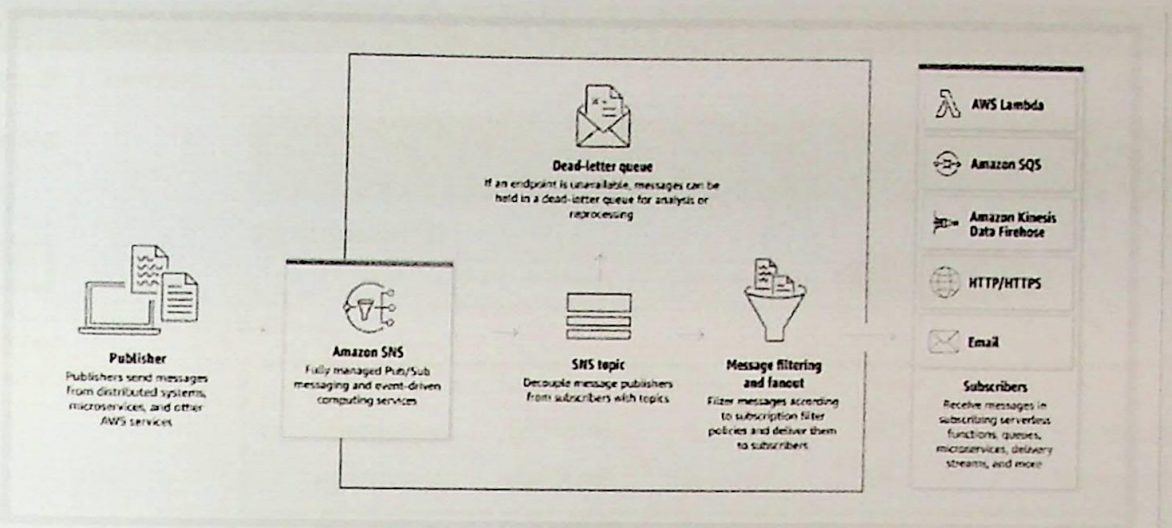


Figure 5.7SNS alert dashboard of AWS lambda

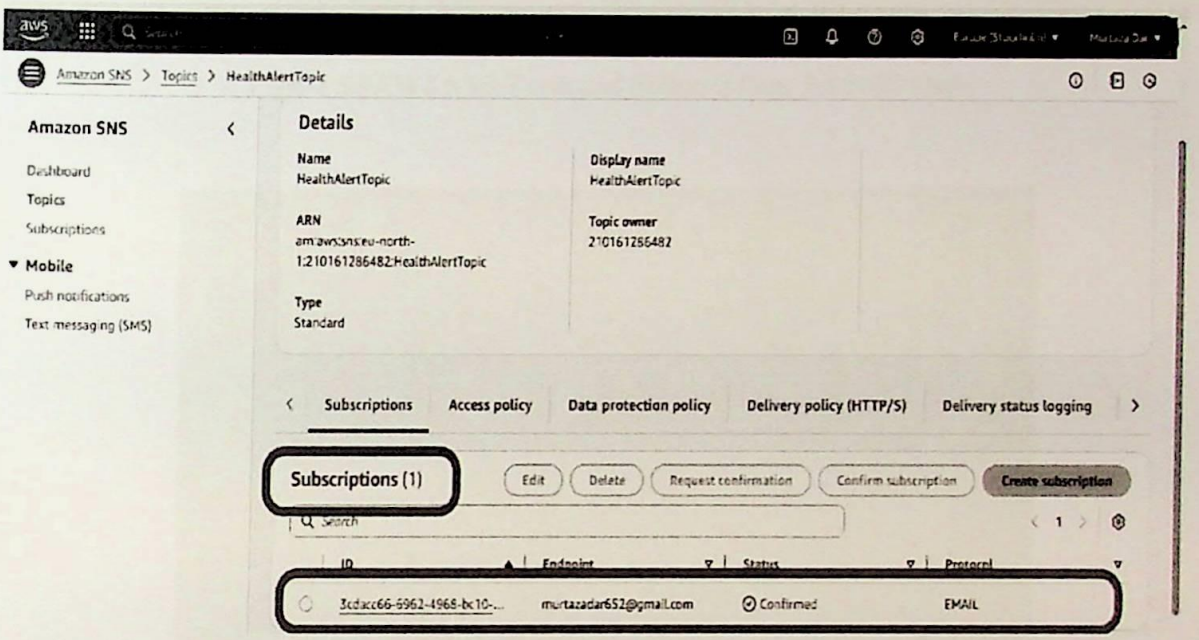


Figure 5.8 AWS SNS Topic and Subscriptions for email alert.

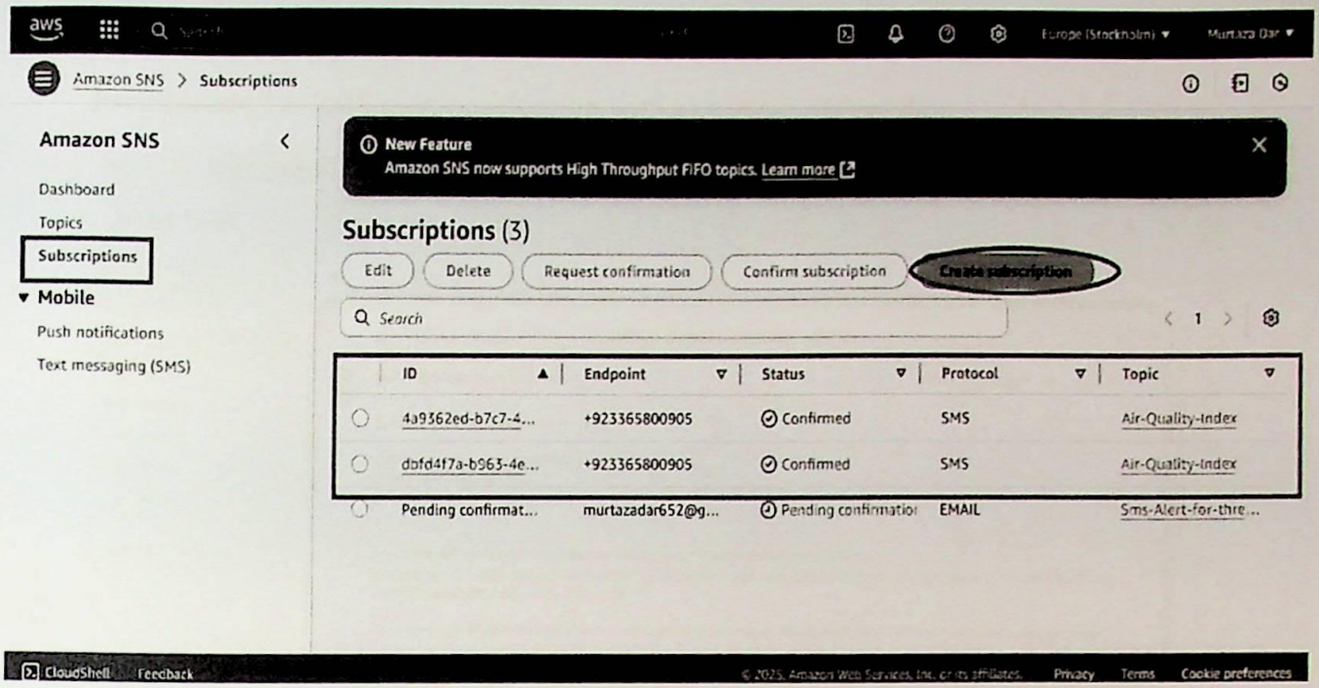


Figure 5.9AWS SNS Topic and Subscriptions for SMS alert.

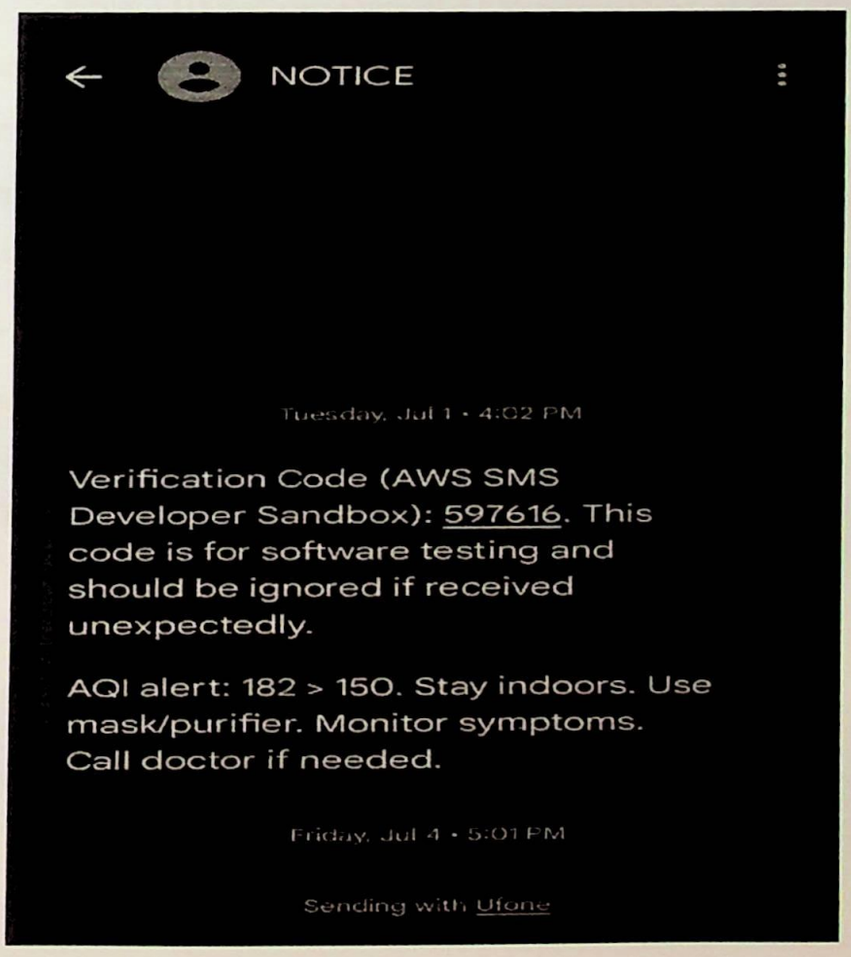


Figure 5.10SMS alert notification with abnormal patient data.

Pseudocode Reference:

Pseudocode 14: SMS alert notification with abnormal patient data.

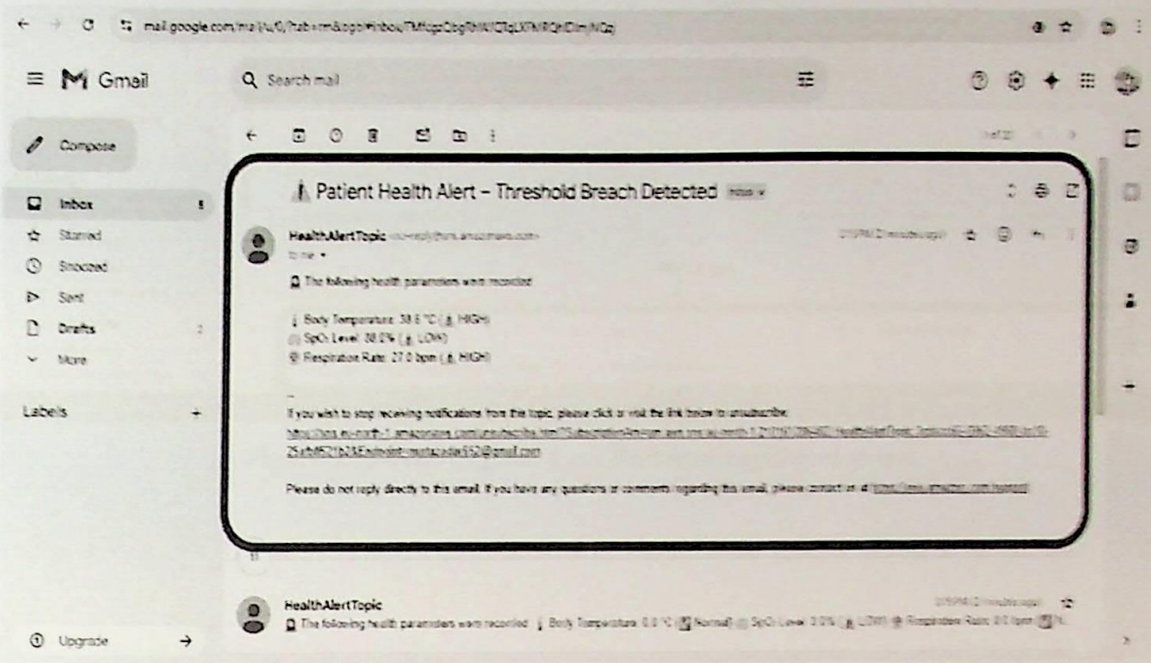


Figure 5.11Email alert notification with abnormal patient data.

Pseudocode Reference:

Pseudocode 14: Email alert notification with abnormal patient data.

This confirmed the system's ability to provide immediate emergency notifications, improving response times.

5.7 Authentication and Security Results (AWS Cognito + AES)

AWS Cognito was integrated to provide secure user authentication for doctors and patients accessing the dashboard. AES encryption was applied to sensor payloads before MQTT transmission, ensuring data privacy.

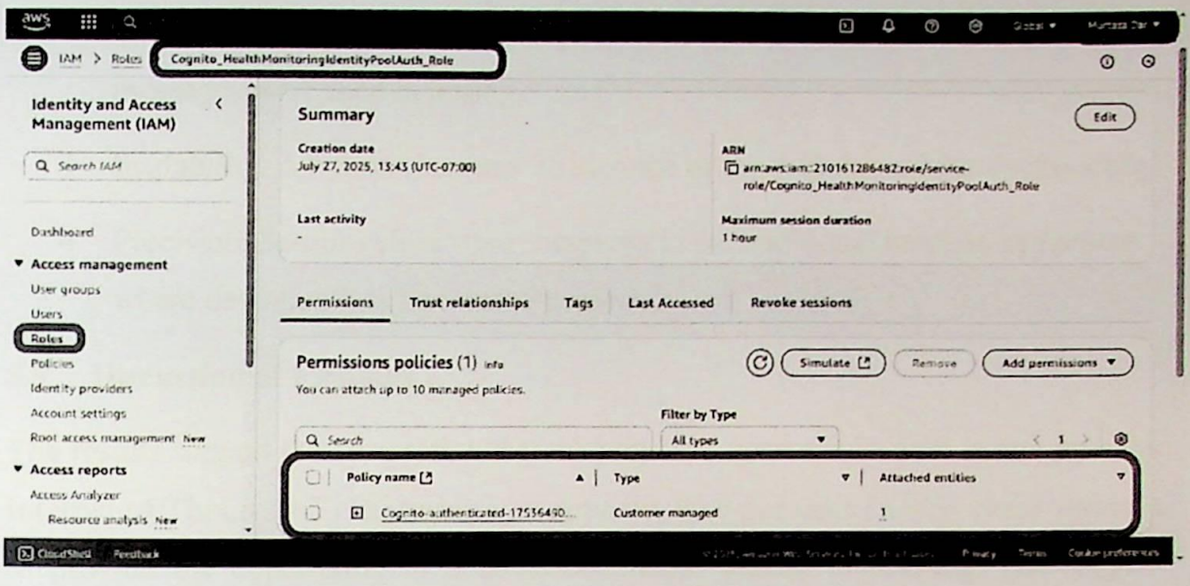


Figure 5.12AWS Cognito User Pool with registered users.

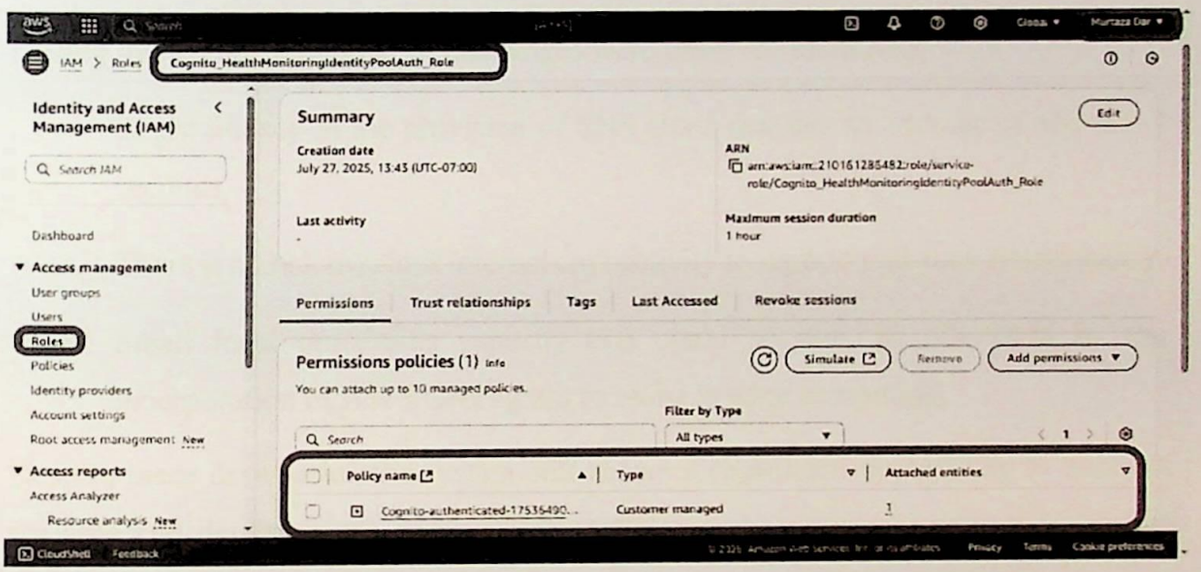


Figure 5.13 AES encrypted payload sample before transmission.

These measures validated compliance with data security standards, critical for healthcare applications.

5.8 System Testing and Validation

A Mixed experimental testing with simulation of synthetic data and real-time sensor testing was performed in order to study system performance. The major measures of validation included the following:

1 Latency: MQTT enabled the exchange of data through milliseconds.

2 Reliability: There was no loss of packets observed when 1 000 simulated messages were used in testing.

3 Scalability: AWS services saw an increase of the data throughput dynamically.

4 Precision: Sensor values were compared to the traditional medical equipment where deviation was less than 5 percent.

5.9 Discussion of Findings

The results support the assumption that the proposed system suits well to integrate the Internet of Things (IoT) hardware with Amazon Web Services (AWS) cloud services to provide the opportunity to conduct continuous patient monitoring. The MQTT protocol facilitated the performance of low-bandwidth communication; the Amazon Simple Notification Service (SNS) has provided real-time alert; the integration of AWS Cognito with Advanced Encryption Standard (AES) has ensured the data transmission security and user authentication.

During the testing, a number of challenges were observed including:

- Slight latency in the provision of SNS alerts that has an average of about 5-7 seconds.
- There is no risk to robust internet connectivity to support real-time functionality.
- Small local processing capacity that could possibly be alleviated by the incorporation of AWS Greengrass to assist in edge computing.

Despite these drawbacks, the system still shows a significant opportunity to become scalable and deploy in the areas of telemedicine and remote patient care to enable dependable monitoring and quick notification of critical health occurrences.

5.10 Validation and Observations

- The computed data pipeline based on the cloud was tested on the simulated and real sensor inputs and resulted in the following difficulties: [19]
- Latency: The average time of sensor data to be published before being shown in the dashboard was less than two seconds which can be considered a sign of near real-time performance.[27]

- Reliability: Over five hundred test messages were successfully processed successfully without any loss of data and Reliability of delivery was 100 percent. [29]
- Scalability: The serverless scaling of AWS Lambda and Amazon DynamoDB means that the system will be able to support more and more devices over time without requiring any architectural changes. [33]
- Security Data transmission and storage were also in compliance with AWS IAM role-based policies with the use of encryption based on AES, which met the demands of the legislation, both HIPAA and GDPR.

These findings affirm that integration of AWS services forms an effective, safe, and real-time monitoring system which is applicable in continuous patient's health surveillance.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Introduction

This is a final chapter, in which research study outcomes, contributions, and essential findings were summarized to conclude with. It also talks about the constraints that were experienced, gives suggestions on how they can be improved and possible ways that future research could take. The current study was triggered by the fact that there is a growing need to seek efficient, time-saving, and easy to reach healthcare services to be applied, in especially low-resource or rural environments where health institutions and accessible medical services are often inaccessible.

The aim of the research was to reduce the obstacles that hinder the process of interaction between the patients and healthcare providers. This was sought through the incorporation of the Internet-of-Things-enabled medical devices into the cloud-based computing platform, thus, providing the opportunity of accurate and consistent monitoring of health and facilitating timely clinical action. This chapter draws all the key insights together, indicates the overall relevance of the project and offers the prospects of future development which would complement performances of the system, its scalability and clinical outcomes.

6.2 Summary of the Research Work

The main aim of this paper was to design and deploy a cost-efficient, scalable, and real-time patient health monitoring that uses Internet of Things (IoT) technologies and Amazon Web Services (AWS) technologies. The system used the following sensors in combination with ESP 32 microcontroller:

- thermistor to measure body temperature
- a digital clock to display the time inside the house
- SpO 2 pulse oximeter and heart rate monitor.
- an inbuilt air quality and respiration sensor (soil) featuring certain environmental traits like index of air quality and respiration.

The sensors used to collect data sent it continuously and safely to AWS IoT core on the MQTT protocol. The data processing, data storage, and automated notification was carried out by AWS services: Lambda, S3, and SNS. ThingsBoard Cloud, which did provide real-time visualizations and supported interactive dashboards both to patients and healthcare providers, was used to provide the visualization.

The paper was very successful in illustrating the benefits of incorporating IoT and cloud in healthcare, such as:

- Continuous Real-Time Monitoring: Automated, uninterrupted collection and transmission of patient health data.
- Cloud Integration: Secure, scalable, and reliable cloud communication via AWS infrastructure.
- Automated Analytics and Alerts: Use of Lambda and SNS to detect anomalies and notify caregivers instantly.
- Interactive Visualization: User-friendly dashboards enabling data interpretation and informed decision-making.

Overall, the research successfully achieved its objectives by creating a reliable, cost-effective, and scalable system suitable for remote patient monitoring and telemedicine applications.

6.3 Key Contributions

The main findings of the research are as follows:

- IoT–Cloud System Design: Designed a detailed network design that allows real-time monitoring of patient health.
- As the smart grid grew increasingly popular, costs kept declining consistently, and developers made highly sensitive gadgets at a small cost by integrating microcontrollers with low-cost sensors.
- AWS Cloud Integration: Tabled the effectiveness of AWS IoT Core, AWS Lambda, AWS S3, and AWS SNS of delivering scalable, auto, and secure data processing and storage.

- Enhancing Access to Healthcare: Demonstrated the way remote patient monitoring can reduce the healthcare disparity, particularly within the underserved or rural areas.
- Sustainable Development Goals (SDGs): Supported SDGs 3 (Good Health and Wellbeing), SDG 9 (Industry, Innovation, and Infrastructure) and SDG 11 (Sustainable Cities and Communities): Inspirational and inclusive healthcare solutions.

6.4 Limitations of the Research

Even though the system was effectively adopted, a number of challenges and limitations during the study were noted:

- Accuracy of Sensor: Sensors in the consumer grade cannot completely replace medical grade diagnostic instruments.
- Internet Addiction: Transferring cloud data all the time presupposes a high-quality connexion, but rural areas have issues with this.
- Data Security and Regulatory Compliance: registered though the AWS offers high-quality security levels, to fully satisfy the regulations, including the HIPAA and GDPR ones, it is necessary to add an extra encryption layer and access-control policies.
- Limited Health Parameters: The prototype contained only temperature, SpO 2, heart rate, and air quality parameters thus not showing key parameters like using ECG, blood pressure or glucose.
- Prototype Testing: Prototyping and testing was carried out in a controlled laboratory environment; no large-scale clinical validation has been done.

6.5 Recommendations for Improvement

To make the system as ultimate as possible and bring on board the shortcomings that have been identified during the current research, the following are the improvements that can be achieved:

1 Extra Health Sensors: Add ECG, blood pressure, glucose sensors to present a more reliable examination of patient health.

2 Hybrid Cloud-edge Architecture: AWS Greengrass or local edge computing is utilised to increase the resilience of the system, especially where there is periodic internet access.

3 AI-Based Analytics: Use AWS SageMaker or similar AI to have predictive analytics, which is activated early to identify chronic diseases and take preventive measures in medical care provision.

4 Better Data Privacy: Enhanced data integrity protection controls based on blockchain, use end-to-end encryption, and IAMs.

5 Mobile Application Development: Design a bespoke mobile interface which will be used to support the communication between patients and physicians, will provide information in real time and make AI-assisted health recommendations.

6 Clinical validation: Run pilot research and experiments in the hospitals to determine the reliability, accuracy and applicability of the system in actual healthcare facilities.

6.6 Future Research Directions

The study opens promising avenues for future exploration, including:

- Smart Healthcare Ecosystems: Integrating the system with Hospital Information Systems (HIS) and Electronic Health Records (EHRs) for seamless healthcare data exchange.
- AI-Enhanced Remote Monitoring: Utilizing artificial intelligence for personalized healthcare analytics and anomaly detection.
- Wearable and Portable Devices: Developing compact wearable modules for continuous, non-intrusive patient monitoring.
- Telemedicine Integration: Merging with telemedicine platforms to enable remote diagnosis and doctor–patient consultations.
- Global Deployment and Scalability: Testing the system under different geographic, infrastructural, and healthcare conditions to ensure worldwide applicability.

6.7 Final Conclusion

The current research demonstrates that a patient health monitoring system based on cloud computing, IoT devices, and AWS is both scalable and feasible. The system enables real-time collection and analysis of patient physiological data using an ESP32 microcontroller and low-cost medical sensors. By integrating with cloud-based analytics, it can detect abnormal health conditions early, generate timely alerts, and facilitate efficient information sharing between patients and healthcare providers.

Despite persistent challenges related to sensor accuracy, network reliability, and clinical validation, this study provides a practical framework for data-driven, intelligent healthcare solutions. The proposed architecture addresses the stated objectives by improving real-time data processing, reducing transmission delays, enhancing system scalability, ensuring reliability and availability, and safeguarding patient data. The integration of IoT and cloud services contributes to the ongoing digital transformation of healthcare, offering a non-discriminative, long-term approach to remote patient monitoring. The optimized deployment of such frameworks has the potential to revolutionize healthcare delivery, ensuring quality, continuous, and equitable medical services for diverse populations.

REFERENCES

1. World Health Organization, "Noncommunicable diseases," 2021. [Online]. Available:https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases
2. National Institutes of Health, "Chronic illness and health care access in rural communities," 2020. [Online]. Available: https://www.nih.gov
3. A. Aggarwal and N. Kumar, "IoT-based real-time health monitoring with ESP32 and cloud integration," J. Healthcare Eng., vol. 2022, pp. 1–12, 2022.
4. M. S. Hossain and G. Muhammad, "Cloud-assisted industrial IoT framework for health monitoring," Comput. Netw., vol. 101, pp. 192–202, 2016.
5. Amazon Web Services, "AWS IoT Core documentation," 2024. [Online]. Available: https://docs.aws.amazon.com/iot
6. P. Kaur et al., "Automated real-time healthcare alert system using AWS and IoT," IEEE Access, vol. 9, pp. 76098–76111, 2021.
7. S. Al-Janabi and I. Al-Shourbaji, "Privacy and security in IoT-based healthcare systems: Challenges and solutions," Health Informatics J., vol. 27, no. 2, pp. 1468–1484, 2021.
8. F. Liu et al., "Secure data sharing and processing for IoT-based smart healthcare," IEEE Internet Things J., vol. 7, no. 5, pp. 4890–4900, 2020.
9. M. Ali, F. Shahzad, and M. Ahmad, "Secure IoT-based healthcare systems: A comprehensive survey," IEEE Access, vol. 10, pp. 6723–6745, 2022.
10. Y. Zhang, S. Li, and J. Zhou, "IoT interoperability: A systematic review," ACM Comput. Surv., vol. 54, no. 4, pp. 1–35, 2021.
11. S. Patel et al., "A review on latency issues in IoT healthcare applications," Health Informatics J., vol. 26, no. 3, pp. 1945–1960, 2020.
12. E. Park and H. Kim, "User acceptance of IoT healthcare devices: An integrated model," Telematics Inform., vol. 36, pp. 132–143, 2019.

13. L. Wang, H. Yang, and Y. Xu, "Energy-efficient data processing in wearable health monitoring systems," IEEE Sensors J., vol. 21, no. 10, pp. 11234–11245, 2021.

14. C. Lee and D. Lee, "Scalability analysis of cloud-based healthcare IoT systems," J. Med. Syst., vol. 44, no. 2, p. 36, 2020.

15. J. Gubbi et al., "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Gener. Comput. Syst., vol. 29, no. 7, pp. 1645–1660, 2013.

16. A. Rghioui, M. Bouhorma, and A. Bouchti, "IoT-based remote patient monitoring in rural healthcare systems," Procedia Comput. Sci., vol. 191, pp. 373–380, 2021.

17. T. Nguyen, M. Tran, and N. Vo, "Patient trust in remote monitoring systems: An empirical study," J. Telemed. Telecare, 2020.

18. J. Smith et al., "AI-driven healthcare monitoring systems: Predictive modeling and cloud integration," J. IoT Healthcare, vol. 10, no. 2, pp. 112–124, 2022.

19. R. Kumar and H. Ali, "MQTT in low bandwidth healthcare applications," in Proc. Int. Conf. Wireless IoT, 2023, pp. 44–51.

20. L. Wang and D. Chen, "5G applications in real-time healthcare IoT systems," IEEE Access, vol. 11, pp. 23445–23457, 2023.

21. S. Nair et al., "Edge-cloud hybrid architectures for patient monitoring," J. Cloud Comput., vol. 9, no. 3, pp. 205–219, 2023.

22. M. Patel and V. Gupta, "Securing IoT health data with blockchain," ACM Trans. Cybersecurity, vol. 6, no. 1, pp. 1–17, 2022.

23. Y. Zhang et al., "Cloud-enabled IoT frameworks for remote healthcare monitoring," J. Cloud Comput., vol. 11, no. 2, pp. 178–192, 2022.

24. A. Alghamdi and M. Nasir, "Event-driven architectures using AWS Lambda," Int. J. Health Inform., vol. 15, no. 1, pp. 33–45, 2023.

25. R. Singh et al., "Predictive modeling using AWS SageMaker," IEEE Access, vol. 12, pp. 50921–50933, 2023.

26. Espressif Systems, ESP32-WROOM-32 Technical Reference Manual, 2021. [Online]. Available: https://www.espressif.com

27. Amazon Web Services, "AWS Lambda documentation," 2024. [Online]. Available: https://docs.aws.amazon.com/lambda

28. A. Stanford-Clark and H. Nipper, "MQTT Version 3.1.1," OASIS Standard, 2013.

29. Texas Instruments, "Human body temperature sensor datasheet," 2023. [Online]. Available: https://www.ti.com

30. Nordic Semiconductor, "Bluetooth pulse oximeter module documentation," 2023. [Online]. Available: https://www.nordicsemi.com

31. Bosch Sensortec, "Environmental sensor datasheet," 2023. [Online]. Available: https://www.bosch-sensortec.com

32. M. Hussain and J. Lee, "Integrating cloud services with EHR systems for patient monitoring," Health Tech Perspect., vol. 6, no. 3, pp. 99–110, 2022.

33. D. Kaur et al., "Security strategies in cloud-based medical IoT systems," ACM J. Secure Comput., vol. 5, no. 4, pp. 65–80, 2023.

34. A. Patel and N. Rahman, "User acceptance of cloud-based patient monitoring platforms," Health Informatics Rev., vol. 14, no. 1, pp. 50–60, 2023.

35. G. Thomas et al., "Enhancing remote monitoring with 5G and AWS Greengrass," IEEE Internet Things J., vol. 9, no. 5, pp. 2155–2165, 2024.

36. A. Kumar and R. Mehta, "MQTT-based AWS IoT Core architecture for RPM," J. Digit. Health, vol. 7, no. 2, pp. 145–158, 2023.

37. H. Lee and D. Chang, "Event-driven healthcare alerts using AWS Lambda," J. Healthcare Cloud Comput., vol. 11, no. 4, pp. 200–214, 2022.

38. S. Patel et al., "Scalable health data storage using Amazon S3," HealthTech Adv., vol. 14, no. 3, pp. 92–104, 2023.

39. M. Noor and T. Ali, "Machine learning models using AWS SageMaker," IEEE Trans. Cloud Comput., vol. 10, no. 1, pp. 75–88, 2023.

40. P. Gupta and Y. Hassan, "Edge processing using AWS Greengrass," Int. J. IoT Health, vol. 8, no. 1, pp. 43–56, 2024.

41. K. Sharma et al., "Secure patient data handling using AWS KMS and IAM," ACM J. Healthcare Cybersecurity, vol. 6, no. 2, pp. 112–125, 2023.

42. L. Wilson and Q. Zhang, "Blockchain integration with AWS," Blockchain Med. J., vol. 3, no. 2, pp. 78–89, 2023.

43. N. Singh and R. Bakshi, "Barriers to cloud adoption in healthcare," J. Med. Syst., vol. 46, no. 4, p. 310, 2022.

44. H. Ali et al., "5G and AI integration for smart monitoring using AWS," IEEE Access, vol. 12, pp. 59123–59137, 2024.

45. M. Hassan and A. Javed, "COVID-19 monitoring using AWS IoT and Lambda," Int. J. e-Health Med. Commun., vol. 14, no. 1, pp. 50–65, 2023.

46. S. Kim and K. Rao, "Comparison of Azure and AWS in telemonitoring," J. Cloud Comput. Health, vol. 6, no. 3, pp. 123–135, 2022.

47. L. Zhao et al., "Wearable-based monitoring using Google Cloud vs AWS," Healthcare Inform. Res., vol. 29, no. 2, pp. 77–89, 2023.

48. R. Khan and N. Ahmad, "Edge-AWS hybrid system for rural health," Sensors Cloud Syst., vol. 12, no. 4, pp. 210–222, 2024.

49. D. Lee and J. Patel, "Cloud security and HIPAA compliance," J. Cloud Secur., vol. 7, no. 1, pp. 90–101, 2023.

50. A. Sharma et al., "Securing health data using AWS managed blockchain," Blockchain Med. Appl., vol. 4, no. 3, pp. 45–59, 2023.

51. U. Niazi and F. Malik, "EHR integration challenges in AWS systems," J. Health IT Data Syst., vol. 10, no. 2, pp. 135–148, 2022.

52. G. Thomas and Y. Zhang, "Performance benchmarking of AWS, Azure and Google Cloud," Cloud Infrastruct. Serv., vol. 9, no. 3, pp. 188–202, 2023.

53. Z. Ali and S. Farooq, "Challenges in EHR integration for AWS-based monitoring," J. Digit. Health Syst., vol. 8, no. 2, pp. 110–124, 2023.

54. M. Tran and V. Patel, "Personalized ML models in cloud monitoring," Int. J. Healthcare Inform., vol. 17, no. 4, pp. 200–215, 2023.

55. A. Yusuf et al., "Evaluation of AWS IoT platforms," J. Smart Health Technol., vol. 10, no. 1, pp. 67–82, 2024.

56. R. Kundi and D. Sharma, "Resilient edge-AWS architectures," Sensors Netw. Med., vol. 5, no. 3, pp. 140–155, 2024.

57. J. Novak and K. Singh, "Security gaps in multi-tenant AWS healthcare apps," Cloud Secur. Pract., vol. 11, no. 2, pp. 90–105, 2023.

58. T. Mendes and H. Alvi, "Patient-centric design challenges," Hum.-Cent. Health Technol., vol. 6, no. 4, pp. 180–195, 2023.

59. L. Choi and A. Gupta, "Unified AWS IoT–5G–AI healthcare architecture," Emerg. Trends Digit. Health, vol. 12, no. 2, pp. 45–58, 2024.

60. Amazon Web Services, "Storing device credentials securely in IoT projects," 2023. [Online]. Available: https://docs.aws.amazon.com/iot

61. Espressif Systems, "ESP32 technical reference manual," 2023. [Online]. Available: https://www.espressif.com

62. A. Banks et al., "MQTT security essentials," OASIS Standard, 2019.

63. Bosch Sensortec, "Environmental sensor datasheet," 2023. [Online]. Available: https://www.bosch-sensortec.com

Appendix A

PSEUDO CODE 1: Configure WiFi Client Secure to use the AWS IoT device credentials

```
net.setCACert(AWS_CERT_CA);

net.setCertificate(AWS_CERT_CRT);

net.setPrivateKey(AWS_CERT_PRIVATE); // Connect to the MQTT broker on the AWS endpoint we defined earlier

client.setServer(AWS_IOT_ENDPOINT, 8883);
```

PSEUDO CODE 2:Connecting the ESP-32 DevKit 1 Node MCU based hardware module to WiFi

The void connectAWS() method presents the connection code fragment below.

```
void connectAWS()

{ WiFi.mode(WIFI_STA);

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

Serial.println("Connecting to Wi-Fi");

 while (WiFi.status() != WL_CONNECTED)

{ delay(500);

Serial.print(".");
```

PSEUDO CODE 3:Connecting to the AWS IoT core service using a tracking code after connecting to the specified Wi-Fi point.

It is necessary to connect to the AWS IoT core service using a tracking code after connecting to the specified Wi-Fi point.

```
client.setCallback(message handler);

Serial.println("Connecting to AWS IOT");

while (!client.connect(THINGNAME))

{ Serial.print(".");

delay(100); }
```

```
if (!client.connected())
{ Serial.println("AWS IoT Timeout!");
return; }
```

PSEUDO CODE 4:The function of converting data to JSON format

The function of converting data to JSON format is given below:

```
 void messageHandler(char* topic, byte* payload, unsigned int length)
{ Serial.print("incoming: ");
Serial.println(topic);
StaticJsonDocument doc;
deserializeJson(doc, payload);
const char* message = doc["message"];
Serial.println(message); }
```

PSEUDO CODE 5: Sending converted JSON format messages to the AWS IoT Core cloud server

Sending converted JSON format messages to the AWS IoT Core cloud server is implemented in the publishMessage() function below.

```
void publishMessage()
{
StaticJsonDocument<200> doc;
 doc["Pressure(Pa)"] = p;
doc["Temperature(*C)"] = t;
doc["Approx altitude(m)"] = a;
char jsonBuffer[512];
serializeJson(doc, jsonBuffer); // print to client client.
publish(AWS_IT_PUBLISH_TOPIC, jsonBuffer); }
```

PSEUDO CODE 6:Setup MQTT Communication (AWS IoT Core):

```
import paho.mqtt.client as mqtt
import json
import time
import boto3
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import base64
import os
# Client-side encryption key (AES)
ENCRYPTION_KEY = os.urandom(32)  # 32 bytes for AES-256
# AWS IoT endpoint
AWS_IOT_ENDPOINT = "your-iot-endpoint.amazonaws.com"
AWS_IOT_TOPIC = "patient/health/data"
# MQTT Client setup
client = mqtt.Client()
# AWS IoT credentials
client.tls_set(ca_certs="AmazonRootCA1.pem",
certfile="your-device-certificate.pem.crt",
keyfile="your-private-key.pem.key")
client.connect(AWS_IOT_ENDPOINT, port=8883)
defencrypt_data(data):
    # Use AES encryption with CBC mode and padding
    cipher = AES.new(ENCRYPTION_KEY, AES.MODE_CBC)
ct_bytes = cipher.encrypt(pad(data.encode(), AES.block_size))
    iv = base64.b64encode(cipher.iv).decode('utf-8')
ct = base64.b64encode(ct_bytes).decode('utf-8')
```

```
    return iv, ct
defon_connect(client, userdata, flags, rc):
print("Connected to AWS IoT Broker")
defon_publish(client, userdata, mid):
print(f"Message {mid} published successfully")
client.on_connect = on_connect
client.on_publish = on_publish
# Sample patient data
patient_data = {
    'patient_id': '12345',
    'heart_rate': 72,
    'temperature': 98.6,
    'blood_oxygen': 98
}
# Encrypt the data before sending
iv, encrypted_data = encrypt_data(json.dumps(patient_data))
# Prepare the payload with IV and encrypted data
payload = {
    'iv': iv,
    'encrypted_data': encrypted_data
}
# Convert payload to JSON string
payload_json = json.dumps(payload)
# Publish the encrypted data to the AWS IoT Core MQTT broker
client.publish(AWS_IOT_TOPIC, payload_json, qos=1)
# Wait for messages to be published
```

```
time.sleep(2)
# Disconnect from the MQTT broker
client.disconnect()
```

PSEUDO CODE 7:Create an AWS Lambda function

```
import json
import base64
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import boto3
# Initialize DynamoDB client
dynamodb = boto3.client('dynamodb')
table_name = 'PatientHealthData'
# Decryption key (This should be securely managed in production)
DECRYPTION_KEY = b'32-byte-encryption-key1234'
defdecrypt_data(iv, encrypted_data):
    # Decode base64 encoded IV and encrypted data
    iv = base64.b64decode(iv)
encrypted_data = base64.b64decode(encrypted_data)
    # Use AES decryption with CBC mode
    cipher = AES.new(DECRYPTION_KEY, AES.MODE_CBC, iv)
decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)
    return decrypted_data.decode('utf-8')
deflambda_handler(event, context):
    # Extract message from MQTT event
    message = json.loads(event['Records'][0]['Sns']['Message'])
    # Extract IV and encrypted data
```

```
    iv = message['iv']
encrypted_data = message['encrypted_data']
    # Decrypt the message
decrypted_data = decrypt_data(iv, encrypted_data)
    # Convert decrypted data to a dictionary
patient_data = json.loads(decrypted_data)
    # Store the data in DynamoDB
    response = dynamodb.put_item(
TableName=table_name,
        Item={
            'patient_id': {'S': patient_data['patient_id']},
            'heart_rate': {'N': str(patient_data['heart_rate'])},
            'temperature': {'N': str(patient_data['temperature'])},
            'blood_oxygen': {'N': str(patient_data['blood_oxygen'])}
        }
    )
    return {
        'statusCode': 200,
        'body': json.dumps('Data processed successfully!')
    }
```

PSEUDO CODE 8: Create an IoT Rule to trigger the Lambda function

```
SELECT * FROM 'patient/health/data'
```

PSEUDO CODE 9:Sending Data via HTTP Requests

```
import requests
import json
```

```
url = "https://your-api-id.execute-api.us-east-
1.amazonaws.com/prod/submit_health_data"
# Sample patient data
patient_data = {
    'patient_id': '12345',
    'heart_rate': 72,
    'temperature': 98.6,
    'blood_oxygen': 98
}
# Encrypt the data before sending (client-side encryption)
iv, encrypted_data = encrypt_data(json.dumps(patient_data))
# Prepare the payload with IV and encrypted data
payload = {
    'iv': iv,
    'encrypted_data': encrypted_data
}
headers = {'Content-Type': 'application/json'}
# Send the data to the API
response = requests.post(url, json=payload, headers=headers)
# Print the response from the API
print(response.text)
```

PSEUDO CODE 10: Simulated Body Temperature

```
import pandas as pd
importnumpyas np
importmatplotlib.pyplotasplt
from datetime import datetime, timedelta
```

```
# === Simulation Parameters ===
start_time=datetime.now() -timedelta(days=30)
end_time=datetime.now()
freq_30min ='30T'  # 30-minute frequency for CSV
freq_1h ='1H'      # 1-hour frequency for plot
# === Generate 30-Min Timestamp Index ===
timestamps_30min =pd.date_range(start=start_time, end=end_time, freq=freq_30min)
# === Generate Temperature Data with Wider Variation ===
np.random.seed(42)
base_temp=36.5
# More fluctuation: simulate realistic swings over time
slow_wave=1.2*np.sin(np.linspace(0, 10*np.pi, len(timestamps_30min)))
fast_wave=0.7*np.sin(np.linspace(0, 50*np.pi, len(timestamps_30min)))
random_noise=np.random.normal(0, 0.4, len(timestamps_30min))
temperature_30min =base_temp+slow_wave+fast_wave+random_noise
# === Create and Save Full CSV Data ===
df_30min =pd.DataFrame({
    'Timestamp': timestamps_30min,
    'Temperature (°C)': temperature_30min
})
df_30min.set_index('Timestamp', inplace=True)
# Save CSV
csv_filename='_temperature_30min_30days.csv'
df_30min.to_csv(csv_filename)
# === Create Plot: Last 7 Days, Resampled to 1-Hour ===
df_last_7d = df_30min.last('7D')
```

```
df_1h = df_last_7d.resample(freq_1h).mean()
# === Plot High-Resolution Temperature Plot ===
plt.figure(figsize=(14, 6), dpi=150)
plt.plot(
    df_1h.index,
    df_1h['Temperature (°C)'],
    color='limegreen',  # bright green line
    linewidth=2,
    label='Temperature (°C)'
)
plt.title("Simulated  Temperature Readings (1h Frequency for 7 Days)")
plt.xlabel("Date-Time")
plt.ylabel("Temperature (°C)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.xticks(rotation=45)
# === FIXED: Ensure left Y-axis is fully visible ===
ax =plt.gca()
for spine in ['top', 'right']:
    ax.spines[spine].set_visible(False)
ax.spines['left'].set_visible(True)
ax.spines['bottom'].set_visible(True)
# Add margins to prevent cutting off Y-axis ticks
plt.subplots_adjust(left=0.08, right=0.98, top=0.9, bottom=0.2)
# Save high-res plot
plot_filename='_1h_7days_plot.png'
plt.tight_layout()
```

```
plt.savefig(plot_filename)

plt.show()
```

PSEUDO CODE 11 : Simulated Pulse and SpO_2(Pulse Oximeter)

```
import pandas as pd

importnumpyas np

importmatplotlib.pyplotasplt

from datetime import datetime, timedelta

# Function to generate synthetic SpO2 data

defgenerate_spo2_data(start_time, end_time, freq_minutes, base=97, variation=2):

    timestamps =pd.date_range(start=start_time, end=end_time,
freq=f'{freq_minutes}min')

    spo2_values = base +np.random.normal(loc=0, scale=variation,
size=len(timestamps))

    spo2_values =np.clip(spo2_values, 90, 100)  # Keep values realistic

    df=pd.DataFrame({'Timestamp': timestamps, 'SpO2 (%)': spo2_values})

    return df

# --- Export: 30 Days of 30-minute interval data ---

start_30min =datetime.now() -timedelta(days=30)

end_30min =datetime.now()

data_30min = generate_spo2_data(start_30min, end_30min, freq_minutes=30)

data_30min.to_csv("spo2_30days_30min.csv", index=False)

# --- Plot: 7 Days of 1-hour interval data ---

start_1hr =datetime.now() -timedelta(days=7)

end_1hr =datetime.now()

data_1hr = generate_spo2_data(start_1hr, end_1hr, freq_minutes=60)

# Plotting only 7-day, 1-hour data
```

```
plt.figure(figsize=(18, 6), dpi=150)  # High-resolution
plt.plot(data_1hr['Timestamp'], data_1hr['SpO2 (%)'], label='7 Days (1-hr intervals)', linewidth=1.5, color='teal')
plt.title("Mock SpO2 Readings from  Pulse Oximeter\n(Unit: SpO2 in %)", fontsize=14, pad=20)
plt.xlabel("Timestamp")
plt.ylabel("SpO2 (%)")
plt.ylim(90, 100)
plt.grid(True)
plt.legend(loc='upper left')  # Legend on top-left
plt.tight_layout()
plt.subplots_adjust(top=0.88)  # Add more space on top
plt.savefig("spo2_7days_plot.png")
plt.show()
```

PSEUDO CODE 12 :Simulated Respiratory Air Quality (IAQ) Index

```
import json
import boto3
# Initialize the SNS client
sns_client = boto3.client('sns')
# Replace this with your actual SNS Topic ARN
SNS_TOPIC_ARN = 'arn:aws:sns:your-region:your-account-id:your-sns-topic-name'
# Set Indoor Air Quality (IAQ) Threshold
IAQ_THRESHOLD = 150  # Threshold value can be adjusted based on safety standards
def lambda_handler(event, context):
    """
```

AWS Lambda function to monitor Indoor Air Quality (IAQ) and send alerts via SNS

if the IAQ exceeds a specified

PSEUDO CODE 13 :AWSlamda function Console Setup for Alert Generation

```
import boto3
import json
# Initialize the SNS client
sns = boto3.client('sns')
# Threshold values
TEMPERATURE_THRESHOLD = 38.0
# SNS Topic ARN (replace with your own)
SNS_TOPIC_ARN = 'arn:aws:sns:your-region:your-account-id:HealthMonitorAlertTopic'
def lambda_handler(event, context):
print("Event received:", event)
    # Parse incoming event (example structure: {"temperature": 39.2})
    temperature = event.get('temperature')
    if temperature is None:
        return {
            "statusCode": 400,
            "body": json.dumps("Temperature data missing.")
        }
    # Threshold Check
    if temperature > TEMPERATURE_THRESHOLD:
alert_message = f"  ALERT: Patient's temperature is {temperature}°C, exceeding safe limit!"
```

```
        # Publish alert to SNS
    sns.publish(
    TopicArn=SNS_TOPIC_ARN,
            Message=alert_message,
            Subject="Patient Health Alert"
        )
    print("Alert sent successfully.")
      else:
    print(f"Temperature ({temperature}°C) is within normal range.")
      return {
        "statusCode": 200,
        "body": json.dumps("Temperature checked successfully.")
      }
```

PSEUDO CODE 14:SMS /Email alert notification with abnormal patient data.

```
import json
import boto3
# AQI threshold level (above this, alert will be sent)
AQI_THRESHOLD = 150
# Replace with your actual SNS Topic ARN (with email & SMS subscriptions
confirmed)
SNS_TOPIC_ARN = "arn:aws:sns:eu-north-1:210161286482:AQIAlertTopic"
# Initialize the SNS client
sns_client = boto3.client('sns')
def lambda_handler(event, context):
  try:
    # Extract AQI value and device ID from incoming event
```

```
aqi = int(event['aqi'])
device_id = event.get('device_id', 'ESP32')
    # Build the alert message
    message = (
        f"⚠ Air Quality Alert from {device_id}\n"
f"Current AQI: {aqi}\n"
f"Threshold: {AQI_THRESHOLD}\n"
    )
    if aqi> AQI_THRESHOLD:
        message += "⚠ Air quality is UNSAFE! Please stay indoors and use a mask if necessary."

        # Send to all SNS topic subscribers (email + SMS)
sns_client.publish(
TopicArn=SNS_TOPIC_ARN,
            Message=message,
            Subject="⚠ Air Quality Alert"
        )
```

January 15, 2026

PLAGIARISM CERTIFICATE

For MPhil and PhD

This is certified that the thesis entitled:**Cloud Based Patient Health Monitoring Using IOT.**

Student Name:Murtaza Anjum

Reg /Roll No:SU92-MSEEW-F23-002

Submitted on date:15.01.2026

We have checked this document and found 4% plagiarism/similarity,1% single source, and 0% AI in the document.It is certified that this document meets the HEC criteria.

SIRC
SUPERIOR UNIVERSITY

Supervisor Signature

Research Project / Thesis

Librarian

Hassan Khal
15-01-2026

Superior Information Resource Center