



Project with Embedded System

Remote Patient Monitoring



Author: Etkä Kocak
Semester: Spring 2025
Subject: Computer Science
Course code: 2DT304

Abstract

This project presents the design and implementation of an Internet of Medical Things-based remote patient monitoring system using embedded hardware and full-stack web technologies.

The system allows patients to measure and transmit vital health data using a device equipped with biomedical sensors capable of capturing necessary data to calculate blood oxygen saturation, body temperature, ECG signals, and systolic blood pressure. A Raspberry Pi Pico W acts as the microcontroller within the device.

A Node.js based backend processes incoming data from the device, stores it in a MongoDB database, and presents the results to patients and healthcare professionals through a web interface. Role-based access control ensures secure data handling, while real-time communication over HTTP using API endpoints enables interactive testing. Overall, the system provides an affordable and extensible solution for remote healthcare monitoring.

Note: This report was written as part of the course 2DT304 – Project with Embedded System at Linnaeus University. It was approved by the course examiner, but has not been officially published by the university. The content, conclusions, and opinions expressed are solely those of the author.

Keywords: IoMT, IoT, Embedded System, Raspberry Pi Pico, Biomedical Sensors, RFID, NFC, RBAC, Node.js, HTTP, REST API, MongoDB, Full Stack Development, Patient, Healthcare Professional

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem definition	1
1.3	Purpose and research question / hypothesis	2
1.4	Scope / Limitation	2
1.5	Target group	2
1.6	Outline	3
2	Related Work	4
2.1	Similar Commercial Systems	4
2.2	Open-Source and Academic Projects	4
2.3	Professional Medical Devices	4
3	Hardware	6
3.1	Embedded Board	6
3.2	Input Device	6
3.2.1	Sensors	7
3.3	Output Device	9
3.4	Schematics	10
3.5	Power Consumption	11
4	Software	12
4.1	Software Preference	12
4.1.1	GitHub	13
4.1.2	System Architecture	13
4.2	External Dependencies	14
4.3	Real-Time	15
4.3.1	Error Handling	15
4.4	Security	15
5	Results and Discussion	22
5.1	System Validation	22
5.2	Medical Accuracy	22
5.3	Security	22
6	Conclusion	26
6.1	Challenges	26
6.2	Future Work	26
	References	27
A	Appendix 1	A

1 Introduction

This chapter will describe relevant background information for the project, and this introduction will also clarify the contents of this project.

1.1 Background

Healthcare technologies worldwide have advanced and also increasingly shifted toward remote solutions, especially in response to the growing number of elderly patients and individuals with chronic illnesses. Regular monitoring of vital signs is crucial for early diagnosis and continuous treatment, but frequent hospital visits are not always practical or accessible. This has led to increased interest in Remote Patient Monitoring (RPM) systems.

RPM refers to technologies that enable patients to measure and transmit health data from home to healthcare providers in real time. These are usually comprehensive systems, but there are also simpler systems that just allow people to track and monitor some of their health metrics, such as the recently released smartwatches.

This project explores the development of such a system by integrating biomedical sensors with a Raspberry Pi Pico W microcontroller to collect key health metrics. The goal is to enable patients to easily perform tests such as blood oxygen level, body temperature, electrocardiogram (ECG), and systolic blood pressure, and to transmit this data to healthcare professionals to monitor and evaluate. But patients may also track and monitor their own test results. The system aims to be both user-friendly and technically extensible, serving as a prototype for modern and comprehensive healthcare solutions.

1.2 Problem definition

While RPM systems are growing in popularity, many of the available solutions are expensive, complex to operate, and not well-suited for elderly or non-technical users. Many commercial solutions require patients to remember login credentials, navigate complicated interfaces, which makes these systems inaccessible to a large portion of the population, particularly older adults or those with limited digital literacy.

There's a gap in the availability of low-cost and user-friendly solutions. This project was initiated to address this gap by building a simple and secure RPM system that uses a more basic but secure login for patients, provides a clean and guided user interface for elderly patients, and offers a cost-effective, open-source alternative for real-time health monitoring.

The problem this project aims to solve is not just technological, but also usability-related: how to make health monitoring more accessible and inclusive for those who need it most.

1.3 Purpose and research question / hypothesis

The main purpose of this project is to develop a cost-effective and user-friendly Remote Patient Monitoring system that allows patients, especially elderly or non-technical individuals, to measure vital health data and share it with healthcare professionals in real time. The system aims to reduce complexity by providing a clear and guided web interface, and simplifying the login process, but keeping it still secure.

The project also focuses on enabling real-time interaction between the device and the server using embedded hardware, biomedical sensors, and full-stack web technologies. In addition to collecting vital signs such as blood oxygen saturation, body temperature, ECG signals, and systolic blood pressure, the system processes and visualizes the data for both the patient and healthcare professionals.

Since the goal is primarily technical and practical, no formal research hypothesis is proposed. Instead, the system is built and evaluated based on its ability to deliver a secure, accessible, and extensible platform for remote health monitoring.

1.4 Scope / Limitation

This project focuses on the design and development of a Remote Patient Monitoring system that collects and transmits those health metrics: blood oxygen saturation, body temperature, ECG signals, and systolic blood pressure using a Raspberry Pi Pico W and several biomedical sensors. The scope of this work is limited to the technical implementation, real-time data communication, and user interface development of the system.

It is important to emphasize that the system is not intended for clinical or diagnostic use. While considerable effort has been made to obtain accurate results within the constraints of the available hardware, the sensors used are low-cost, non-medical grade components, and the signal processing algorithms are meant for demonstration and technical validation purposes only. Therefore, the system does not claim to provide medically certified or diagnostically reliable health data.

This work should be viewed as a technological prototype that demonstrates how embedded systems, IoT (including IoMT), and web technologies can be used to build a functioning RPM platform. Not as a homemade professional medical system.

1.5 Target group

This report is primarily intended for individuals and researchers in the fields of computer science, particularly in areas such as embedded systems engineering, signal processing, backend development, and database management. It may also be of interest to researchers and developers working on health-related technologies.

1.6 Outline

This report is organized into six main chapters. Following this introduction, chapter 2 gives an overview about similar systems, related work, and studies in the area of this project, remote patient monitoring. After that, chapter 3 will focus on all the details of hardware implementation, while chapter 4 details the software of the system. Chapter 5 presents an analysis of the results of the project and reflects on how well the solution meets its goals. Finally, chapter 6 concludes the report by summarizing the project, outlining challenges faced during development, and proposing ideas for future improvements.

2 Related Work

This chapter explores existing work related to remote patient monitoring systems, including commercial products, academic projects, and professional medical devices.

2.1 Similar Commercial Systems

Today, there are several commercial solutions available for health tracking and remote patient monitoring. Major tech companies, for example, like Apple and Samsung, have developed smartwatches that enable users to monitor key health metrics. Both Apple's smartwatch series [1] and Samsung Galaxy Watch [2] series offer some common health-tracking features with our project, such as blood oxygen saturation and ECG recording.

2.2 Open-Source and Academic Projects

In the realm of remote patient monitoring, several open-source and academic initiatives have been developed to provide solutions. Below are some notable similar projects that align with our work:

- **HomeICU: Remote Patient Monitor for COVID-19** [3]

HomeICU is an open-source patient monitoring system that utilizes wearable sensors to measure vital health metrics. The project encompasses firmware and a base station for comprehensive monitoring.

- **Raspberry Pi Controlled Health Monitor System** [4]

Developed as a final project at Cornell University, this system measures health metrics using an inflatable cuff and sensors. It integrates hardware components with a Raspberry Pi 3 for data processing and display.

- **Vital-RPM: Remote Patient Monitoring System** [5]

Vital-RPM is a system designed to monitor patients' vital signs with automated health status assessments using machine learning techniques. technologies.

2.3 Professional Medical Devices

Hospitals and clinics use professional-grade medical devices to monitor patients' vital signs. These devices offer highly accurate results and are operated by trained professionals.

For example, one of the commonly used professional ECG systems in hospitals is the CARDIOVIT AT-180, a high-performance 16-channel device. Despite its high cost, it is widely adopted in clinical settings due to its advanced diagnostic capabilities. [6] Additionally, major technology companies such as Philips also develop professional-grade patient monitoring systems, offering comprehensive solutions for healthcare providers. [7]

It would not be entirely fair to directly compare this project to professional patient monitoring systems or products developed by major technology companies. However, several academic and open-source projects were examined for reference. What sets this

project apart is the integration of a device-linked web application designed for both patients and healthcare professionals. While some other projects include web or mobile interfaces, not all offer a fully functional, role-based platform that is ready for real-world usage. Moreover, many of the examined systems were intended for single-user use, whereas the system developed here supports multiple users via NFC-based authentication, without requiring configuration of the device to a single user.

3 Hardware

This chapter explains the hardware used in the system, including the embedded board, connected sensors, circuit schematic, and power management considerations.

3.1 Embedded Board

The Raspberry Pi Pico W was used as the main embedded board and microcontroller in the device part of this project. It is based on the RP2040 dual-core Arm Cortex-M0+ processor, running at 133 MHz, with 264 KB of SRAM and 2 MB of onboard flash memory. It includes a variety of I/O options such as GPIO, I2C, SPI, UART, and ADC, which makes it suitable for interfacing with multiple sensors simultaneously. [8]

The Pico W also features integrated Wi-Fi connectivity, allowing wireless communications. HTTP protocol is used in this project for the wireless communications between the device and the backend server. Pico W operates without an operating system but instead runs firmware written in MicroPython, which allows efficient control over GPIO and sensor operations while maintaining a lightweight execution model suitable for resource-constrained devices. [8]

This microcontroller was selected because of its low power consumption, affordable price, and sufficient performance for handling sensor data acquisition and basic processing before transmitting it to the server.

3.2 Input Device

The system includes four input devices, three of which are biomedical sensors that collect data from the patient. These include:

- **MAX30102:** An optical sensor that is used in this device to measure SpO₂ (blood oxygen saturation) and heart rate. It communicates with the Raspberry Pi Pico W via the I2C protocol, allowing digital data transfer. In the software, it is accessed through a MicroPython library that controls initialization, data reading, and basic filtering.
- **ICQUANZX heart rate:** An analog sensor that is used in this device to capture the electrical activity of the heart to measure ECG (electrocardiography) and heart rate. It is connected to the ADC (Analog-to-Digital Converter) pin of the Pico, and real-time analog signals are read using the ADC module of MicroPython.
- **DS18B20:** A digital temperature sensor used to measure body temperature. It communicates over the 1-Wire protocol. The sensor is accessed through a MicroPython library that handles device initialization and temperature reading functions.
- **RC532 NFC Module:** This module is used for user authentication by reading NFC tags or cards. It operates over the SPI protocol, and is accessed using a custom MicroPython library adapted for Pico boards. When a valid NFC tag that is connected to a patient is detected, the patient is securely logged into the system.

All sensor data is collected by the Raspberry Pi Pico W and transmitted to the server via HTTP POST requests. This modular sensor design enables the device to support a variety of health monitoring tasks with reliable data acquisition and communication.

3.2.1 Sensors

This section explains how each sensor in the system is used to collect meaningful data from the patient. Rather than focusing on software implementation details, the emphasis here is on the functional role of each sensor and how it contributes to the overall monitoring process. Specifics about the sensor algorithms, data processing, and code logic will be discussed in the next chapter.

- **Patient Login:** The RC532 NFC module is used to authenticate patients before starting any measurement. When a patient taps their NFC tag or card on the reader, the device identifies the user and initiates a session securely linked to their ID. This method eliminates the need for usernames or passwords and simplifies access, especially for elderly users.
- **Body Temperature:** The DS18B20 sensor measures the patient's body temperature via skin contact and provides accurate digital readings using the 1-Wire protocol. When the sensor is triggered, the Raspberry Pi Pico W collects temperature values using this sensor and sends the result directly to the server after processing. The server then stores the value in the database and makes it available in the test history of the patient.
- **Blood Oxygen Saturation (SpO2):** The MAX30102 uses infrared and red light absorption to estimate blood oxygen saturation via skin contact. The Raspberry Pi Pico W reads raw PPG (photoplethysmography) data and locally applies a filtering and ratio-based algorithm to calculate SpO2. The final processed value is then transmitted to the server.
- **Electrocardiography (ECG):** The ICQUANZX analog sensor captures the electrical signals of the heart when the patient places their finger on the contact pad. The Pico reads the analog voltage through its ADC pin and collects a stream of raw values, and sends this list of data to the server. Due to the limited memory and processing power of the Pico W, the data is not processed locally. Instead, it is transmitted to the server, where the waveform is visualized and analyzed.
- **Systolic Blood Pressure:** The system estimates systolic blood pressure using the Pulse Transit Time (PTT) method. The ICQUANZX sensor measures the pulse at the fingertip, while the MAX30102 sensor measures the pulse at the fingertip of the other hand. The Raspberry Pi sends the raw data in real time from both sensors to the server. The server calculates the time difference between these two pulse waves. The server then calculates PTT and applies a predefined mathematical model to compute the systolic blood pressure.

To ensure ease of use, especially for elderly or non-technical users, each sensor interaction is accompanied by clear visual instructions provided through the web interface. These images guide the patient on how to correctly place or interact with each sensor before initiating a test. These instructional visuals are also included in this report for reference, illustrating the simplicity and accessibility of the system.



Figure 3.1: Instruction Image: Body Temperature Measurement



Figure 3.2: Instruction Image: ECG Measurement

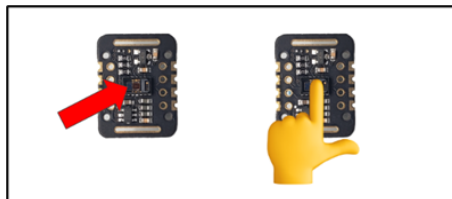


Figure 3.3: Instruction Image: SpO2 Measurement

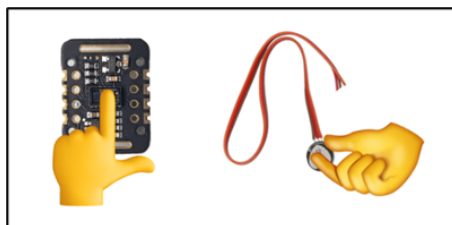


Figure 3.4: Instruction Image: Blood Pressure Measurement

3.3 Output Device

The main output interface of the system is the web-based user interface, which presents the processed test results to both patients and healthcare professionals. After the Raspberry Pi Pico W collects the sensor data, it sends it to the server, where it is stored and processed. The server then generates visual and textual representations of the results and displays them via the web interface.

Output elements include real-time feedback such as successful login confirmation, test initiation status, numerical values for temperature, SpO2, and blood pressure, as well as a graphical representation of the ECG waveform. The interface is designed to be clear and accessible, making the data easy to interpret even for users with limited technical knowledge.

Although the Raspberry Pi Pico W technically is the output device of the circuit by sending data out from the sensors, the actual output for the user is the web interface, where the final results are visualized and interpreted. The device uses the HTTP protocol to communicate with the backend server through RESTful API endpoints. For each test, the server triggers the Raspberry Pi via an HTTP GET request, and the Raspberry Pi responds with an HTTP POST request containing the measured data. This data is then processed, stored in the database, and presented to the user through the web interface.

3.4 Schematics

The schematic below illustrates the complete wiring diagram of the Remote Patient Monitoring device. The Raspberry Pi Pico W is the central microcontroller, interfacing with all sensors and modules through various communication protocols.

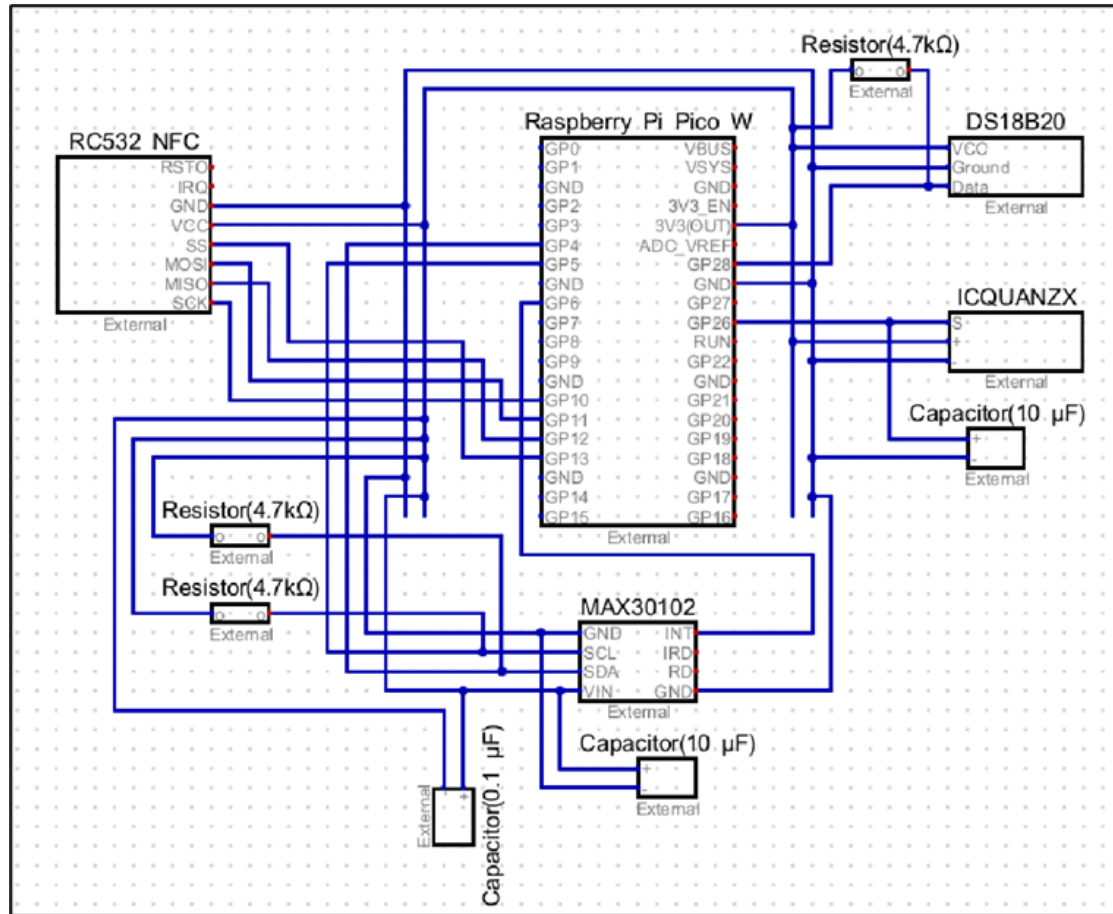


Figure 3.5: Circuit Schematic: Remote Patient Monitoring Device

MAX30102 is connected via the I2C protocol (SCL and SDA). Datasheet:[9]

ICQUANZX is connected to the ADC pin of Pico to capture analog voltage signals, with filtering capacitors used to reduce noise. No datasheet available.

PN532 NFC Module is wired over SPI protocol (SCK, MOSI, MISO, and SS). Datasheet:[10]

DS18B20 uses the 1-Wire protocol and requires a $4.7k\Omega$ pull-up resistor on the data line. Datasheet:[11]

Resistors and capacitors are used where necessary to ensure proper voltage regulation, signal stability, and noise suppression.

3.5 Power Consumption

Power efficiency was a key consideration in the design of this system, especially given the continuous nature of remote health monitoring. To reduce unnecessary power usage, the Raspberry Pi Pico W does not keep biomedical sensors continuously active.

Instead, it operates in an event-driven mode, listening for HTTP GET requests from the server. Once a request is received, the Pico selectively powers on only the required sensor(s) to perform the measurement. After the data is collected and transmitted via HTTP POST, the sensors are deactivated.

This approach ensures that no component is running when not needed and optimizes energy consumption.

4 Software

This chapter explains the software components developed for the system, including the backend logic and frontend interface of the web application, Raspberry Pi firmware, and overall system communication protocols.

4.1 Software Preference

Multiple programming languages were used in different layers of the system. The back-end server was developed using JavaScript with Node.js, and the frontend interface used EJS (Embedded JavaScript) for dynamic HTML rendering, along with CSS for styling. Additionally, Python was used on the server side to process raw signals, such as PTT calculation, which requires numerical computation and final result calculation on the server side instead, because of the memory limitation of the microcontroller.

However, the primary focus of this section is the language used on the microcontroller. The Raspberry Pi Pico W was programmed using MicroPython, a version of Python optimized to run on microcontrollers. [12]

MicroPython was chosen for several practical and technical reasons: [12]

- **Development:** MicroPython allows for rapid prototyping and testing, which is crucial in projects involving multiple sensors and communication protocols.
- **Readability and maintainability:** Its syntax is clean and easy to understand, making it highly suitable for academic projects and collaborative development.
- **Hardware-level access:** MicroPython provides direct access to GPIO, I2C, SPI, ADC, and UART interfaces, fulfilling all hardware control needs of this project.
- **Community support:** MicroPython has an active community and rich documentation, which makes debugging and troubleshooting significantly faster.
- **Safety and reliability:** One of the key advantages of using MicroPython is its built-in safety and error-handling features. Unlike low-level languages such as C or Assembly, which can cause undefined behavior or even physical hardware damage when misused, MicroPython provides a protected runtime environment. For example, in cases of excessive memory usage or invalid operations, MicroPython raises runtime exceptions instead of crashing the device. It also abstracts hardware interactions through safe APIs, reducing the risk of damaging GPIO pins or misconfiguration peripherals. This makes MicroPython a more reliable option, especially in prototyping and educational environments where safety is critical. But of course, MicroPython doesn't abstract away the responsibility, it just simply allows you to fail safely.

While lower-level languages like C/C++ or assembly offer developers more control and possibly better performance, they also require much more development time and are more prone to memory-related bugs. In contrast to MicroPython, lower-level languages typically require setting up complex build systems, such as configuring build files or toolchains, before you can even blink an LED. In contrast, MicroPython enabled us

to focus on the functionality and integration of our system rather than spending excessive time on low-level hardware handling. In short, MicroPython was the most efficient, reliable, accessible, and productive choice for this application.

4.1.1 GitHub

The entire software developed for this project is hosted in a public GitHub repository. This includes not only the code used in the final web application and Raspberry Pi Pico W, but also the scripts used during sensor testing and early prototyping. Hosting the code on GitHub enables version control, collaborative development, and full transparency into the project's evolution. The repository contains a detailed commit history, providing clear insight into how the system was built and improved over time. This makes the project easy to maintain, extend, or replicate. The GitHub Repository link is available in references [13] and appendices, see Appendix 1.

4.1.2 System Architecture

The figure below presents the overall system architecture of the remote patient monitoring platform developed in this project. The system combines embedded hardware, server-side software, and a web-based frontend to create a fully integrated full-stack application. A web interface communicates with the hardware device Raspberry Pi Pico W in real time via HTTP requests, allowing users to initiate tests and view results directly through the application.

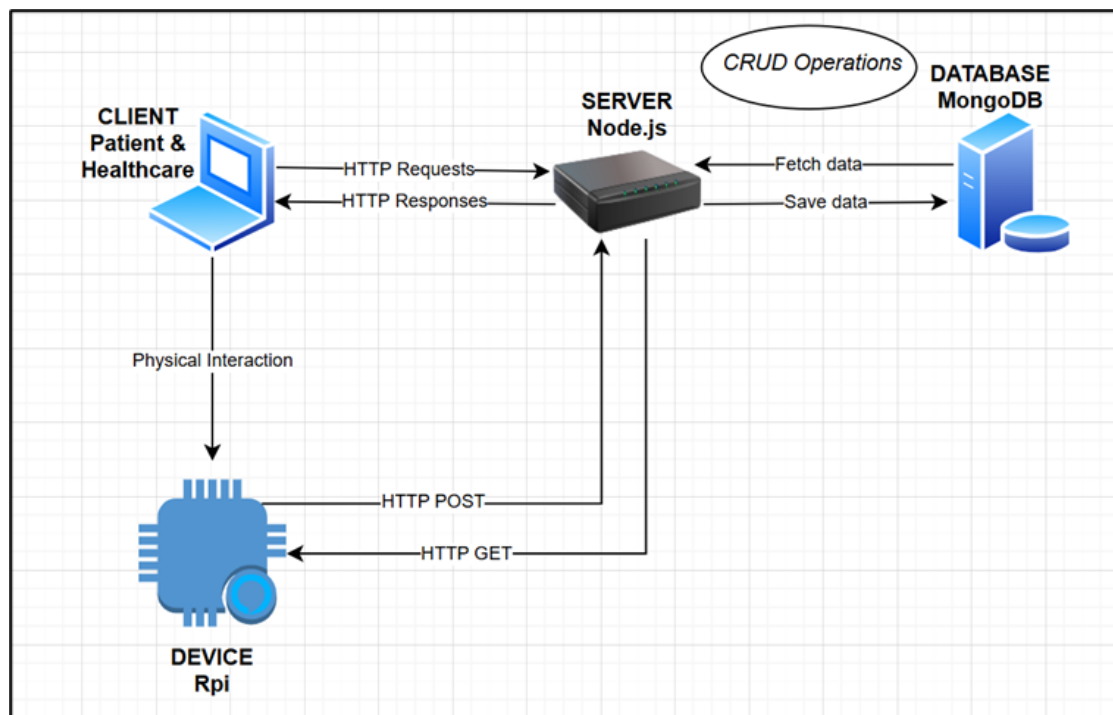


Figure 4.6: Architectural Diagram: Remote Patient Monitoring

The system architecture is composed of four main components: the client interface, the backend server, the embedded device Raspberry Pi Pico W, and the MongoDB database. The interaction between these components is illustrated in Figure 4.6.

The Raspberry Pi collects sensor data from the patient and sends it to the backend server using HTTP POST requests. The server, developed using Node.js and the Express.js framework, handles routing and API operations. Upon receiving data from the device, the server processes and stores it in the MongoDB database using standard CRUD operations (create, read, update, and delete).

To initiate a test, the patient interacts with the web application, which sends an HTTP GET request to the Raspberry Pi. This triggers the device to activate the corresponding sensor and begin data collection. After gathering the data, the Pi transmits it back to the server, where it is processed and made available for viewing through the user interface.

Communication between the device and the server is structured via a RESTful API, and the HTTP protocol is used consistently for all device-server interactions.

4.2 External Dependencies

In line with ethical development practices, all external dependencies used in this project are transparently declared here.

While most of the libraries used were already available within the default MicroPython environment, two external libraries were additionally installed to enable certain sensor integrations. Codes of these two libraries were not included in our GitHub repository; only the codes developed by our team are present in the repository to maintain a clean separation between original and external code.

Built-in MicroPython libraries used:

- **machine:** Provides control over GPIO pins, ADCs, and other hardware peripherals.
- **time / utime:** Used for delays and time-based operations.
- **ujson:** JSON handling for encoding/decoding data structures.
- **urequests:** Allows HTTP requests (GET/POST) from the microcontroller to the server.
- **network:** Used to connect the device to Wi-Fi networks.
- **math:** Used for mathematical calculations such as exponentials and filtering.
- **onewire / ds18x20:** Used to interface with the DS18B20 sensor over the 1-Wire protocol.

External libraries used:

- **max30102:** A driver module for the MAX30102 sensor. Library:[14]
- **NFC_PN532:** A driver module for the PN532 NFC modules. Library:[15]

4.3 Real-Time

The system supports real-time communication between the Raspberry Pi Pico W and the server. When a patient initiates a test via the web interface, the server immediately sends an HTTP GET request to the Pi to trigger the relevant sensor. The Pi activates the sensor, collects the data, and transmits it back to the server via an HTTP POST request. This workflow creates a responsive and interactive testing experience.

However, "real-time" in this context does not imply instant test results. Measurements, for example as ECG, require time to gather meaningful data. For instance, ECG acquisition takes approximately 20–30 seconds of continuous signal collection. During this period, the device streams raw data, which is processed server-side to generate meaningful results and visualizations.

4.3.1 Error Handling

In the case of unexpected delays, failed sensor readings, network issues, or server-side timeouts, the system has been designed to fail gracefully. For example, if a sensor measurement fails or times out, the server-side logic ensures that:

- A clear message is returned to the user indicating failure.
- No corrupt or partial data is stored.

The Raspberry Pi also uses exception handling (try/except) in its MicroPython scripts to catch hardware errors and handle them without crashing the system. Instead of halting execution, it provides error messages or retry logic, ensuring stability and robustness.

This combination of responsive communication, timeout handling, and graceful degradation allows the system to maintain usability even under imperfect conditions.

4.4 Security

Security is a critical aspect of any healthcare-related system. To ensure both data privacy and proper access control, this system implements several important security mechanisms.

RBAC: To ensure secure access to sensitive health data and system operations, a Role-Based Authorization Control mechanism has been implemented. This system restricts access based on predefined user roles: patients and healthcare professionals. Patients are allowed to log in, perform tests, and view only their own test history. Healthcare professionals, on the other hand, can view all registered patients' data and add new patients to the system.

Safe routing: Access control is enforced using Express middleware, which validates the user's role before granting access to protected routes. This implementation adheres to the principle of least privilege, ensuring users are only able to perform actions appropriate to their role.

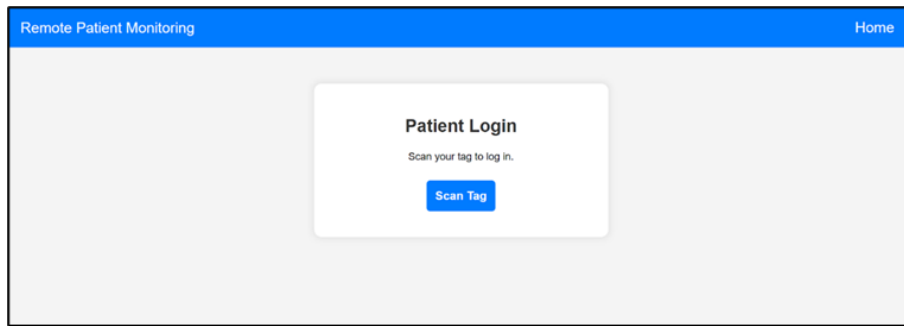
Passwords: Healthcare professionals log in with a username and password, while patients log in using an NFC card assigned to them by a healthcare professional during

registration. All user passwords are securely hashed using “bcrypt” before being stored in the database. This ensures that raw passwords are never stored or transmitted. To register as a healthcare professional, a valid authorization code (configured by us during setup) must be entered. This prevents unauthorized users from registering as medical staff.

Session-Based Authentication: Once a user is logged in, their session is securely maintained using “express-session”. Sensitive operations check the session to verify authentication and authorization status before proceeding. API endpoints consistently check for login status every time they are called.

Every effort has been made to ensure the system adheres to current security standards and remains robust for practical use, even in settings such as modern institutions. But it is important to acknowledge that no digital system can be considered entirely immune to exploitation. Although no known vulnerabilities are currently present, a thorough security audit by a cybersecurity professional might uncover weaknesses not apparent during development. For this reason, security should be viewed as an ongoing process rather than a one-time implementation.

In summary, are there any known security issues in this system? As of now, no such issues have been identified. But is the system entirely immune to exploitation? Certainly not, like any digital system, it may still contain unknown vulnerabilities.



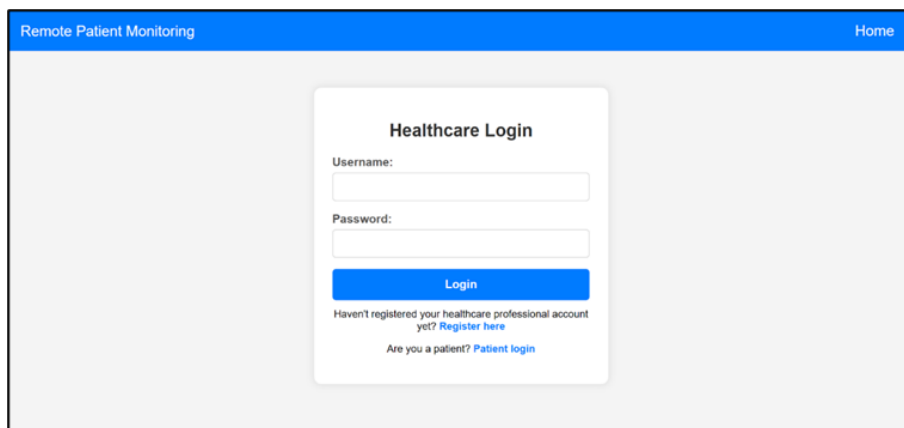
Remote Patient Monitoring Home

Patient Login

Scan your tag to log in.

[Scan Tag](#)

Figure 4.7: Website Screenshot: Patient Login Interface



Remote Patient Monitoring Home

Healthcare Login

Username:

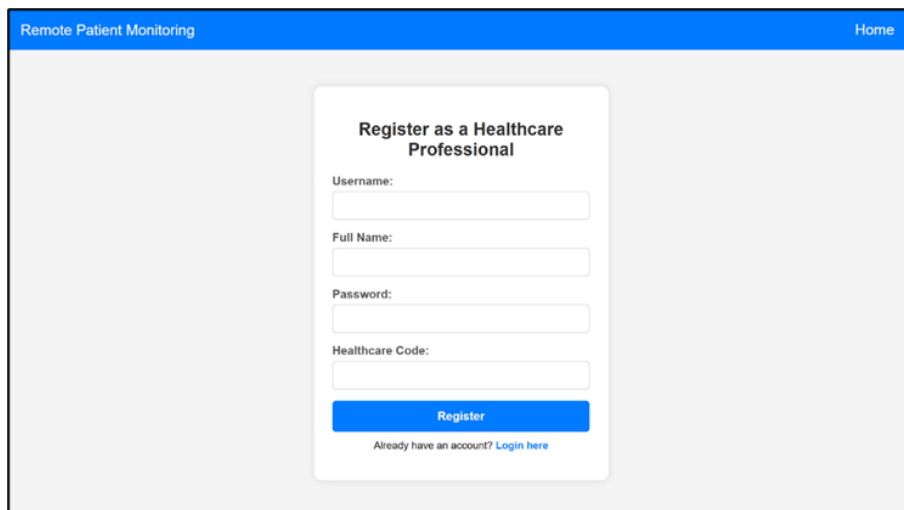
Password:

[Login](#)

Haven't registered your healthcare professional account yet? [Register here](#)

Are you a patient? [Patient login](#)

Figure 4.8: Website Screenshot: Healthcare Login Interface



Remote Patient Monitoring Home

Register as a Healthcare Professional

Username:

Full Name:

Password:

Healthcare Code:

[Register](#)

Already have an account? [Login here](#)

Figure 4.9: Website Screenshot: Healthcare Register Interface

Remote Patient Monitoring [← Back to Dashboard](#)

Add New Patient

Full Name:

Patient ID (RFID UID):

Press 'Scan Tag' to get UID

Age:

Gender:

Select Gender

Smoking:

Does the patient smoke?

Regular Exercise:

Does the patient exercise regularly?

Hypertension Condition:

Select Hypertension Condition

Blood Pressure Condition:

Select Blood Pressure Condition

Figure 4.10: Website Screenshot: Healthcare Dashboard - Register a patient

```

_id: ObjectId('67ced500fbdb315c5fd5eeba')
username : "131-49-215-12"
password : "$2b$10$LyCQNUtEY07SuK.ulwy00eA3
fullname : "Etkan K"
role : "patient"
age : 21
gender : "Male"
smoking : "Yes"
exercise : "No"
hypertension : "No"
bloodpressure : "Normal"
__v : 0

```

Figure 4.11: Database Screenshot: Users – Patient

```
_id: ObjectId('67cec1457d0d1740e8912c26')
username : "etka"
password : "$2b$10$4pwUv1DaP7zme5HUw3XcZeP
fullname : "Etka Kocak"
role : "healthcare"
__v : 0
```

Figure 4.12: Database Screenshot: Users – Healthcare

```
_id: ObjectId('67d48611bb3f8a1e3501ee6e')
thepatient : "131-49-215-12"
result : 96.1
testType : "spo2"
createdAt : 2025-03-14T19:40:01.116+00:00
__v : 0
```

Figure 4.13: Database Screenshot: A test result - SpO2

```
_id: ObjectId('67d4aa212d2fcb0cc9e4458d')
thepatient : "131-49-215-12"
▶ result : Array (200)
testType : "ekg"
createdAt : 2025-03-14T22:13:53.156+00:00
__v : 0
```

Figure 4.14: Database Screenshot: A test result – ECG

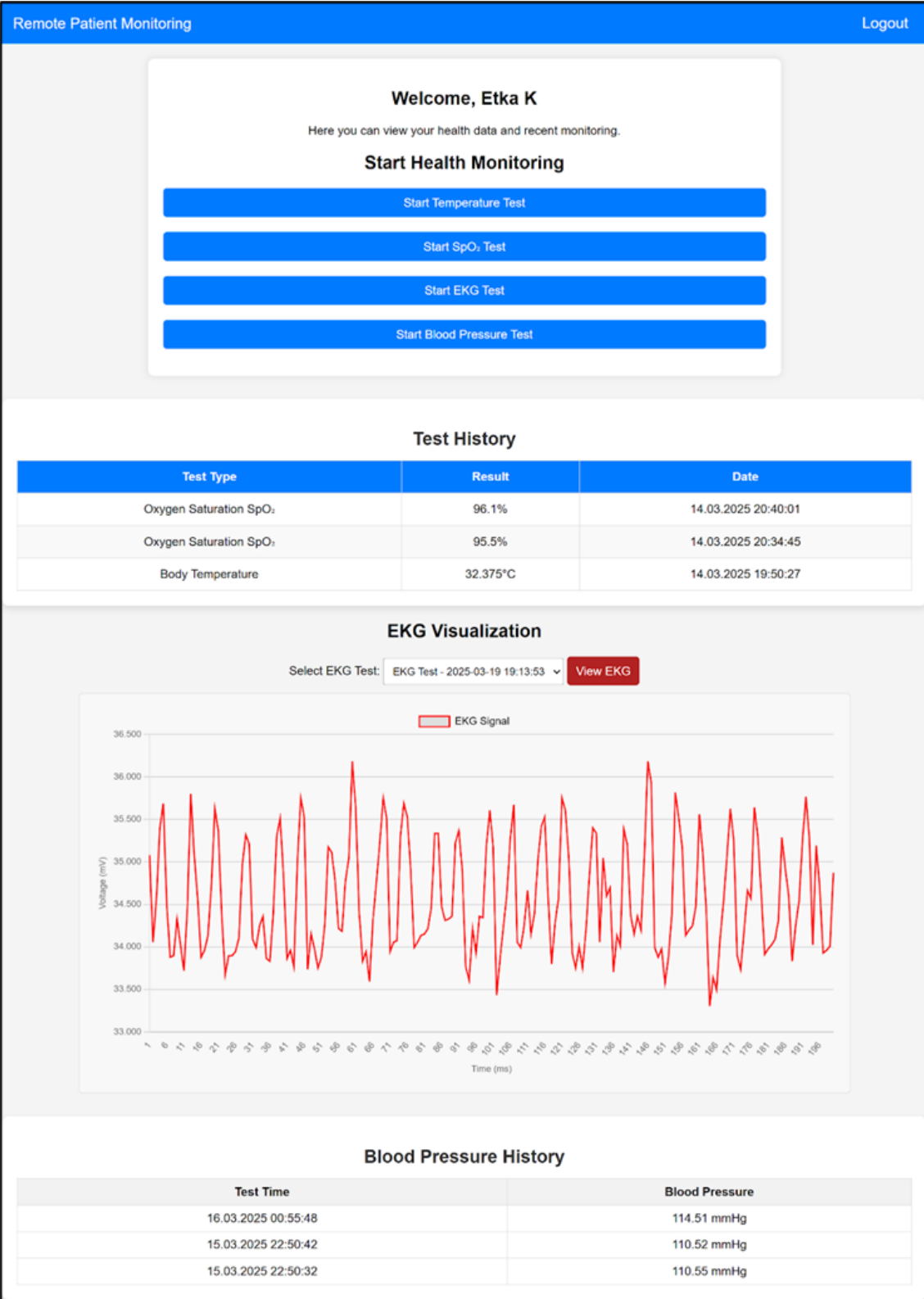


Figure 4.15: Website Screenshot: Patient Dashboard

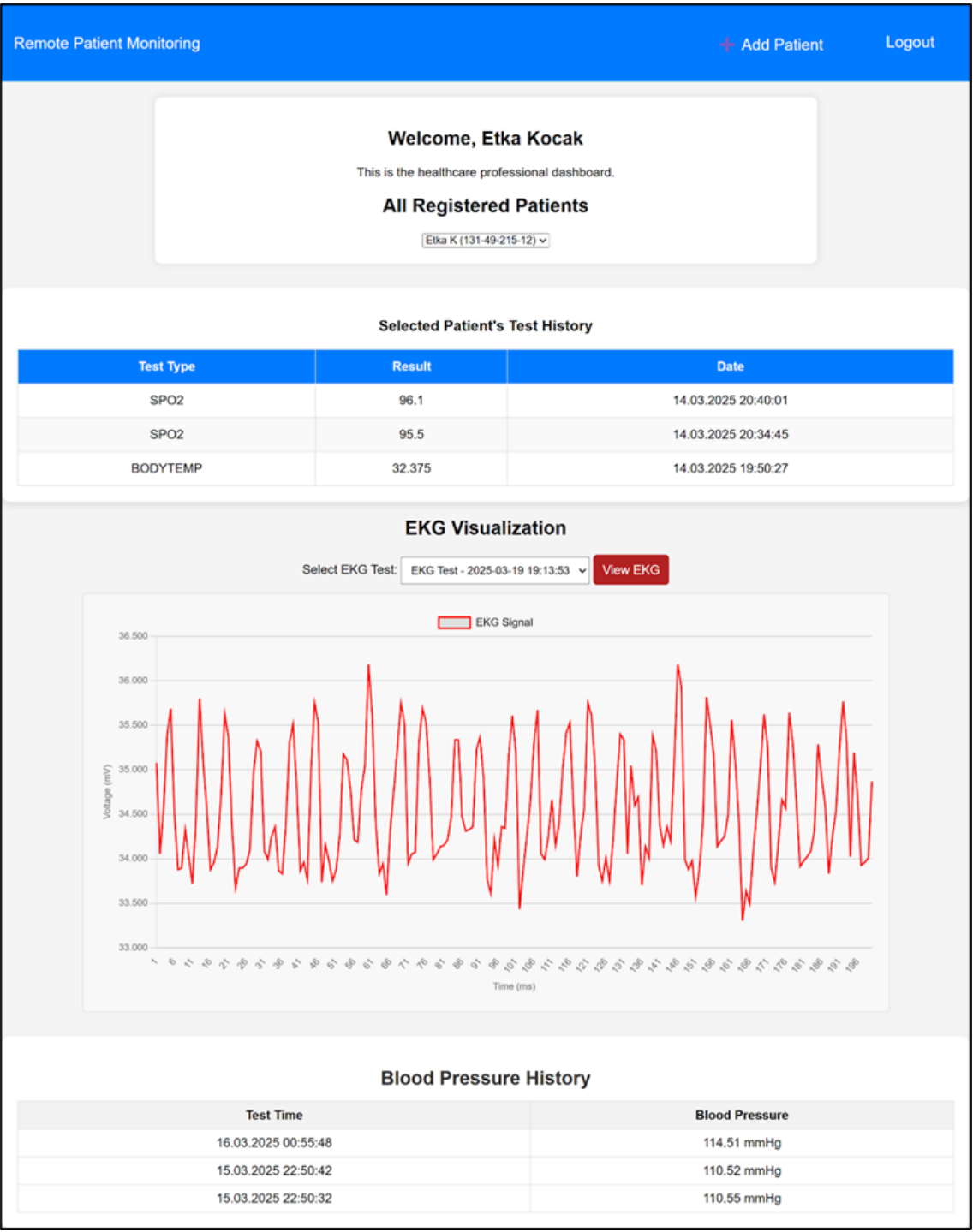


Figure 4.16: Website Screenshot: Healthcare Dashboard

5 Results and Discussion

This chapter presents the outcomes of the developed system, discusses its functionality and limitations. Images of the electronic circuit setup are included to visually show the implementation results.

5.1 System Validation

The system was successfully implemented and tested. All key components including the Raspberry Pi Pico W device, biomedical sensors, NFC module, and web application functioned as expected. The device was able to perform all four measurements (SpO₂, body temperature, ECG, and systolic blood pressure), authenticate users via NFC, and communicate data to the server in real time.

The web application also operated without issue. Patients were able to log in by scanning their own tag, perform tests, and view their results. Healthcare professionals could log in with their credentials, register a new patient, and access all patient records with their test results data. The backend, built with Node.js and MongoDB, reliably handled all data storage, retrieval, and display. The user authentication and role-based access control mechanisms also performed correctly during testing.

5.2 Medical Accuracy

While the system performed technically well, it is important to clarify that it has not undergone any formal medical validation. The sensors used are consumer-grade, and the algorithms implemented for calculating final results are based on general use models, not certified professional grade clinical algorithms.

Any data collected and visualized should be used only for technical demonstration or educational purposes, unless further clinical testing is conducted under professional supervision. Therefore, it is important to emphasize once again that this project is a computer science initiative, not a medical study, and was not developed in collaboration with any medical institution or department.

5.3 Security

The functionality application is illustrated with multiple web interface screenshots in Chapter 4 (see Figures 4.7 to 4.16). These figures show the actual output produced by the system after real test sessions. All test results displayed in those screenshots were collected during live usage of the device by Etkä Kocak, who performed the tests on himself using the developed system.

The architecture of the device was explained in Chapter 3 (see Figure 3.5), but this chapter will also contain real images of the device and used hardware components.

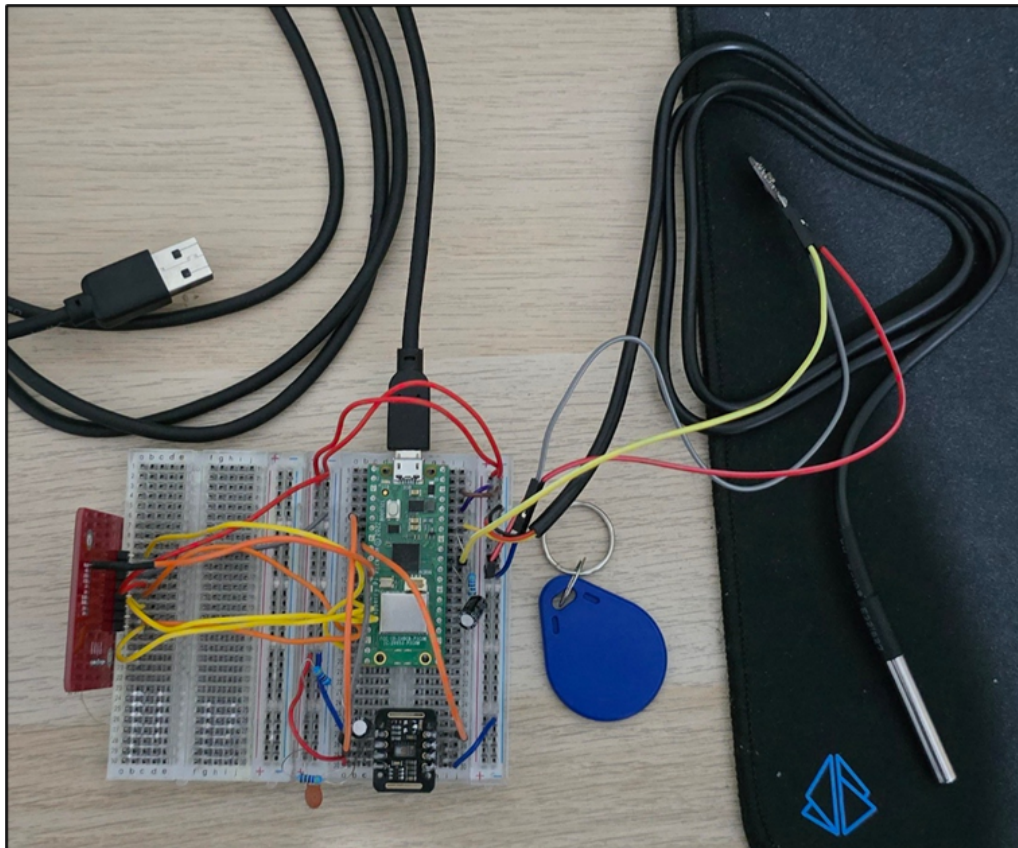


Figure 5.17: Full View: Remote Patient Monitoring Device

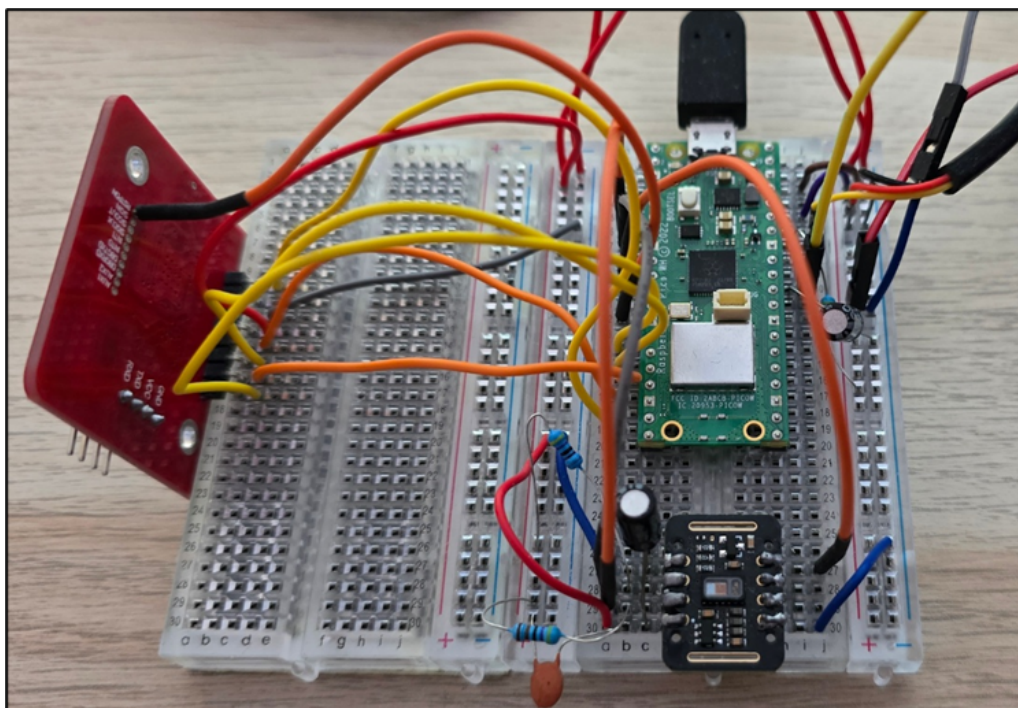


Figure 5.18: Detail Shot: Remote Patient Monitoring Device

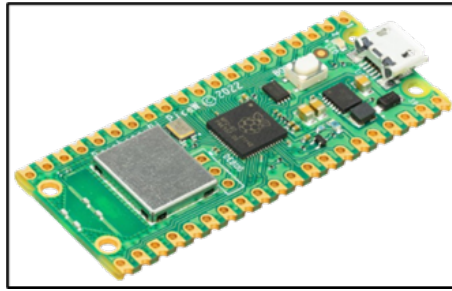


Figure 5.19: Component: Raspberry Pi Pico W

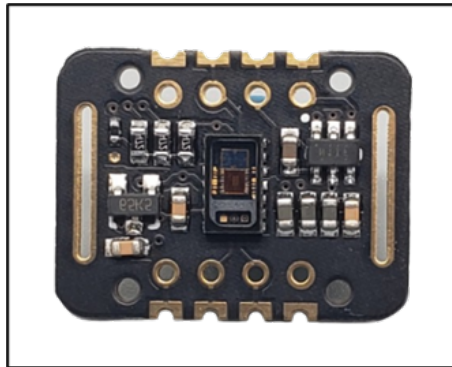


Figure 5.20: Component: MAX30102



Figure 5.21: Component: DS18B20



Figure 5.22: Component: ICQUANZX heart rate



Figure 5.23: Component: PN532 NFC module and NFC tag

6 Conclusion

This chapter explains the key outcomes of the development process, highlighting challenges and outlines potential future improvements if the project were to continue development with more resources.

6.1 Challenges

Throughout the project, two major challenges were encountered, one of which was successfully resolved, while the other was ultimately deemed unfeasible under the current circumstances.

The first challenge was the limited processing capability of the Raspberry Pi Pico W. It was sufficient for handling calculations and transmitting final results for body temperature and SpO2 measurements. But it lacked the computational power and memory to process to calculate final results for ECG and systolic blood pressure measurements. To overcome this, the device was adapted to send raw data directly to the server for ECG and blood pressure measurements, where algorithms on the server handle the processing, visualization, and result generation. This approach enabled the system to retain real-time interaction while offloading heavier computations to a more capable environment.

The second challenge revolved around the initial plan to include glucose level monitoring. The idea had to be abandoned due to both technical and ethical limitations. Commercial glucose sensors require a prescription[16] and are not intended for use by individuals without diagnosed diabetes or blood sugar disorders.[17] Moreover, even if a sensor device had been acquired, its data is fully encrypted and incompatible with custom integration.[18] Rather than simply discarding this part of the project, it was replaced with ECG measurement, which not only added clinical value but also demonstrated the team's commitment to maintaining project scope and innovation.

6.2 Future Work

If the project were to continue development, as for future improvements, one step could be to deploy the Node.js server on a cloud platform and assign it a public domain name, making the website easily accessible online for demonstration and interaction.

Additionally, if the project were to continue development, it could be further validated through professional medical testing, ensuring the accuracy of all health measurements and potentially qualifying it as a certified medical device.

References

- [1] Apple Inc. (2025) Compare Apple Watch models. Apple Sverige. [Online]. Available: <https://www.apple.com/se/watch/compare/?modelList=watch-se-gen2,watch-series-10,watch-ultra-2>
- [2] Samsung Electronics. (2025) Galaxy Watch7 44mm Green Bluetooth. Samsung Sverige. [Online]. Available: <https://www.samsung.com/se/watches/galaxy-watch/galaxy-watch7-44mm-green-bluetooth-sm-l310nzaeub/>
- [3] ZWang. (2020) HomeICU: Remote Patient Monitor for COVID-19. GitHub repository. [Online]. Available: <https://github.com/hellozed/homeicu>. [Accessed: April 2, 2025].
- [4] J. Dong and H. Ye, “Raspberry Pi Controlled Health Monitor System,” Cornell University ECE 5725 Final Projects, 2017. [Online]. Available: https://courses.ece.cornell.edu/ece5990/ECE5725_Spring2017_Projects/5725_final/index.htm
- [5] Hammadh Arquil. (2023) Vital-RPM: Remote Patient Monitoring System. GitHub repository. [Online]. Available: <https://github.com/hammvdh/vitalrpm>. [Accessed: April 2, 2025].
- [6] SCHILLER AG. (2025) CARDIOVIT AT-180. [Online]. Available: <https://www.schiller.ch/en/products/cardiovit-at-180-p34>
- [7] Philips Healthcare. (2025) Patient Monitoring. Philips Sverige. [Online]. Available: <https://www.philips.se/healthcare/solutions/patient-monitoring>
- [8] Raspberry Pi Ltd. (2025) Raspberry Pi Pico and Pico W Documentation. [Online]. Available: <https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html>
- [9] Maxim Integrated. (2018) MAX30102: Pulse Oximeter and Heart-Rate Sensor for Wearable Health. Analog Devices. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/max30102.pdf>
- [10] NXP Semiconductors. (2017, Nov. 28) PN532: NFC Controller. NXP. [Online]. Available: https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf
- [11] Maxim Integrated. (2019) DS18B20: Programmable Resolution 1-Wire Digital Thermometer. Analog Devices. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf>
- [12] MicroPython Project. (2025) MicroPython - Python for microcontrollers. MicroPython.org. [Online]. Available: <https://micropython.org/>
- [13] Etkä Kocak. (2025) IoMT Remote Patient Monitoring. GitHub repository. [Online]. Available: <https://github.com/etkakocak/IoMT-Remote-Patient-Monitoring>.
- [14] Nicola. (2024) MAX30102 MicroPython Driver. GitHub repository. [Online]. Available: <https://github.com/n-elias/MAX30102-MicroPython-driver>. [Accessed: March, 2025].

- [15] Carglglz. (2020) NFC PN532 SPI. GitHub repository. [Online]. Available: https://github.com/Carglglz/NFC_PN532_SPI. [Accessed: March, 2025].
- [16] Region Stockholm. (2025, Apr. 1) Kontinuerlig glukosmätare. Vårdgivarguiden. [Online]. Available: <https://vardgivarguiden.se/kunskapsstod/hjalpmedelsguiden/behovstrappor/vard-och-behandling/livsuppehallande-behov/medicinska-funktionsnedsattningar/diabeteshjalpmedel/kontinuerlig-glukosmatare>
- [17] VISS – Vårdinformation Stockholms län. (2023, Jan.) Diabetes hos vuxna – vårdprogram. Viss.nu. [Online]. Available: <https://viss.nu/kunskapsstod/vardprogram/diabetes-hos-vuxna>
- [18] Abbott Diabetes Care. (2022) System specifications for FreeStyle Libre 3. FreeStyleServer.com. [Online]. Available: https://freestyleserver.com/Payloads/IFU/2022/q2/DOC44922_Rev-B/Assets/mi_system_specifications.html?direction=forward

A Appendix 1

Source Code

The full source code for the project is available on GitHub at the following link:
<https://github.com/etkakocak/IoMT-Remote-Patient-Monitoring>