МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп’ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувачка кафедри,
д-р техн. наук, проф.

_____Ірина ЖУРАВСЬКА

« __ » _____ 202__ р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

IoT-система збору та первинної обробки біомедичних показників пацієнтів на базі Raspberry Pi та ESP32

Спеціальність 123 Комп’ютерна інженерія

Освітня програма «Комп’ютерна інженерія»

Здобувач _____Кирил ЗАВГОРОДНІЙ
підпис

«__» _____ 202__ р.

Керівник PhD, доцент (б.в.з) _____ Євген ДАРНАПУК
підпис

«__» _____ 202__ р.

Миколаїв – 2025

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри комп’ютерної інженерії
_____Ірина ЖУРАВСЬКА
«_____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Завгородній Кирил Сергійович
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи

IoT-система збору та первинної обробки біомедичних показників пацієнтів на базі Raspberry Pi та ESP32

Затверджена наказом по ЧНУ ім. Петра Могили від 23.06.2025 № 165/1.

2. Строк представлення кваліфікаційної роботи «_10_» __грудня___ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Діючий апаратно-програмний прототип IoT-системи на базі мікроконтролера ESP32 та одноплатного комп’ютера Raspberry Pi, що створює умови для збір, первинну фільтрацію (SMA, медіанний фільтр), передачу (MQTT) та візуалізацію (Grafana) біомедичних показників (пульс, сатурація, температура) у реальному часі; програмне забезпечення для вбудованої та серверної частин; пояснювальна записка.
Технічна документація на апаратні платформи ESP32 та Raspberry Pi 4; специфікації сенсорів MAX30102 та DS18B20; медичні нормативи щодо діапазонів фізіологічних показників людини; стандарти протоколів передачі даних MQTT та HTTP.

4. Перелік питань, що підлягають розробці:

Аналіз сучасного стану технологій IoT у медицині та огляд існуючих аналогів систем моніторингу здоров’я.
Розробка математичних моделей фільтрації сенсорних сигналів та інфологічної моделі структури даних для їх зберігання.
Проєктування архітектури системи, вибір апаратних компонентів та обґрунтування програмного стека (Python, InfluxDB, Grafana).
Розробка принципової електричної схеми та алгоритмів взаємодії компонентів системи.
Програмна реалізація вбудованого ПЗ для ESP32 та серверної частини на Raspberry Pi.
Тестування системи, перевірка метрологічних характеристик та оцінка ефективності розробки.

5. Перелік графічних матеріалів
Структурна схема системи; схема електрична принципова; блок-схема алгоритму роботи; скріншоти програмного інтерфейсу.

6. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ Євген ДАРНАПУК
Особистий підпис *Власне ім’я ПРІЗВИЩЕ*

Здобувач _____ Кирил ЗАВГОРОДНІЙ
Особистий підпис *Власне ім’я ПРІЗВИЩЕ*

Дата видачі завдання «_____»_____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної магістерської роботи

Тема: IoT-система збору та первинної обробки біомедичних показників пацієнтів на базі Raspberry Pi та ESP32

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КМР	01.09.2025	04.09.2025	Виконано
2	Огляд літератури за темою роботи	05.09.2025	16.09.2025	Виконано
3	Складання календарного плану КМР	17.09.2025	25.09.2025	Виконано
4	Аналіз предметної області	25.09.2025	30.09.2025	Виконано
5	Розробка проєктних рішень	01.10.2025	14.10.2025	Виконано
6	Моделювання та конструювання АПЗ	15.10.2025	20.10.2025	Виконано
7	Перевірка працездатності, тестування та апробація розробленого АПЗ, аналіз результатів тестування	21.10.2025	13.11.2025	Виконано
8	Оформлення КМР та презентації	14.11.2025	24.11.2025	Виконано
9	Попередній захист	25.11.2025	25.11.2025	Виконано
10	Відгук керівника КМР	26.11.2025	04.12.2025	Виконано
11	Рецензування	05.12.2025	08.12.2025	Виконано
12	Завершення оформлення КМР та презентації	09.12.2025	11.09.2025	Виконано
13	Захист кваліфікаційної магістерської роботи	17.12.2025	17.12.2025	Виконано

Керівник роботи _____ Євген ДАРНАПУК

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Кирил ЗАВГОРОДНІЙ

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

«____» _____ 20__ р

АНОТАЦІЯ
до кваліфікаційної магістерської роботи
«IoT-система збору та первинної обробки біомедичних показників пацієнтів на базі Raspberry Pi та ESP32»
Студент 605 гр.: Завгородній Кирил Сергійович
Керівник: PhD, доцент (б. в. з) Дарнапук Є. С.

Актуальність теми. Розвиток технологій Інтернету речей (IoT) створює нові можливості для автоматизованого збору та аналізу біомедичних даних, що має важливе значення для сучасної медицини. Потреба у безперервному моніторингу життєвих показників пацієнтів, особливо у віддалених або домашніх умовах, зумовлює необхідність розробки доступних, гнучких і надійних систем дистанційного контролю. Використання недорогих апаратних платформ, таких як Raspberry Pi та ESP32, надає змогу створювати енергоефективні рішення, придатні для практичного застосування у сфері охорони здоров'я.

Об'єкт дослідження – процес збору та первинної обробки біомедичних даних із використанням технологій Інтернету речей.

Предмет дослідження – методи, математичні моделі, засоби та програмно-апаратна реалізація IoT-системи для моніторингу біомедичних показників пацієнтів.

Мета роботи – розроблення IoT-системи для збору, алгоритмічної обробки та візуалізації біомедичних даних пацієнтів у режимі реального часу на базі мікроконтролера ESP32 і одноплатного комп'ютера Raspberry Pi.

Для досягнення поставленої мети вирішено такі основні **завдання**:

- проведено аналіз сучасних IoT-рішень та апаратних платформ у сфері медичного моніторингу;
- розроблено математичні моделі фільтрації шумів та структуру даних для ефективного зберігання часових рядів;
- обґрунтовано вибір архітектури системи та інструментів розробки;
- спроєктовано апаратну частину комплексу та програмну логіку взаємодії компонентів;
- реалізовано функціональний прототип системи та виконано його тестування.

Практична значимість полягає у можливості застосування створеної системи для організації дистанційного медичного нагляду в клініках або в домашніх умовах, а також у навчальному процесі під час вивчення дисциплін IoT і комп'ютерної інженерії.

Апробація роботи. Основні положення роботи доповідались на XXVIII Всеукраїнській науково-практичній конференції «Могилянські читання – 2025» (м. Миколаїв). Результати дослідження також пройшли апробацію під час лабораторних випробувань на кафедрі комп'ютерної інженерії ЧНУ ім. Петра Могили.

Основні результати роботи за розділами:
У першому розділі проведено аналіз предметної області, досліджено існуючі комерційні й відкриті IoT-рішення, обґрунтовано вибір апаратної бази (ESP32, Raspberry Pi) та сформовано вимоги до системи.
У другому розділі виконано математичну формалізацію задачі моніторингу, обґрунтовано вибір методів цифрової фільтрації сигналів та розроблено інфологічну модель структури даних для їх ефективного зберігання.
У третьому розділі спроєктовано архітектуру трирівневої IoT-системи, розроблено електричні схеми підключення сенсорів (MAX30102, DS18B20) та алгоритми взаємодії компонентів через протокол MQTT.
У четвертому розділі реалізовано програмне забезпечення (Firmware для ESP32, серверна частина на Python/Flask), налаштовано середовище візуалізації Grafana та проведено комплексне тестування системи, яке підтвердило її працездатність та метрологічну точність.

Кваліфікаційна робота містить: 109 сторінок, 6 таблиці, 20 рисунків та 62 джерел посилання.

Ключові слова: IoT, ESP32, Raspberry Pi, MQTT, InfluxDB, Grafana, цифрова фільтрація, біомедичні показники, дистанційний моніторинг.

ABSTRACT
of the Master's Thesis
" IoT System for Collection and Primary Processing of Patient Biomedical Indicators
Based on Raspberry Pi and ESP32"
Student: Zavgorodniy Kyrylo Serhiyovych
Supervisor: PhD, Associate Professor (b. v. z) Darnapuk Ye. S.

Relevance of the topic. The development of Internet of Things (IoT) technologies creates new opportunities for automated collection and analysis of biomedical data, which is of great importance for modern medicine. The need for continuous monitoring of patients' vital signs, especially in remote or home settings, necessitates the development of accessible, flexible, and reliable remote monitoring systems. The use of low-cost hardware platforms, such as Raspberry Pi and ESP32, allows for the creation of energy-efficient solutions suitable for practical application in the healthcare sector.

Object of research – the process of collecting and primary processing of biomedical data using Internet of Things technologies.

Subject of research – methods, mathematical models, tools, and hardware-software implementation of an IoT system for monitoring patient biomedical indicators.

Aim of the work – to develop an IoT system for collecting, algorithmically processing, and visualizing patient biomedical data in real-time based on the ESP32 microcontroller and the Raspberry Pi single-board computer.

To achieve this goal, the following main tasks were solved:

- an analysis of modern IoT solutions and hardware platforms in the field of medical monitoring was conducted;
- mathematical models for noise filtering and a data structure for efficient storage of time series were developed;
- the choice of system architecture and development tools was justified;
- the hardware part of the complex and the software logic of component interaction were designed;
- a functional prototype of the system was implemented and tested.

Practical significance lies in the possibility of applying the created system for organizing remote medical supervision in clinics or at home, as well as in the educational process when studying IoT and computer engineering disciplines.

Approbation of the work. The main provisions of the work were reported at the XXVIII All-Ukrainian Scientific and Practical Conference "Mohyla Readings – 2025" (Mykolaiv). The research results were also tested during laboratory trials at the Department of Computer Engineering of Petro Mohyla Black Sea National University.

Main results of the work by chapters:
In the first chapter, an analysis of the subject area was conducted, existing

commercial and open IoT solutions were investigated, the choice of hardware base (ESP32, Raspberry Pi) was justified, and system requirements were formulated. **In the second chapter**, a mathematical formalization of the monitoring task was performed, the choice of digital signal filtering methods was justified, and an infological model of the data structure for their efficient storage was developed. **In the third chapter**, the architecture of the three-tier IoT system was designed, electrical connection schematics for sensors (MAX30102, DS18B20) were developed, as well as algorithms for component interaction via the MQTT protocol. **In the fourth chapter**, the software was implemented (Firmware for ESP32, server part on Python/Flask), the Grafana visualization environment was configured, and complex system testing was conducted, confirming its operability and metrological accuracy.

The qualification work contains: 109 pages, 6 tables, 20 figures, and 62 references.

Keywords: IoT, ESP32, Raspberry Pi, MQTT, InfluxDB, Grafana, digital filtering, biomedical indicators, remote monitoring.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ADC – Analog-to-Digital Converter

API – Application Programming Interface

BLE – Bluetooth Low Energy

CPU – Central Processing Unit

CSV – Comma-Separated Values

DB – Database

ESP32 – Embedded System Platform 32

GPIO – General Purpose Input/Output

GUI – Graphical User Interface

HTTP – Hypertext Transfer Protocol

IoMT – Internet of Medical Things

IoT – Internet of Things

JSON – JavaScript Object Notation

LCD – Liquid Crystal Display

LED – Light Emitting Diode

MQTT – Message Queuing Telemetry Transport

PCB – Printed Circuit Board

RAM – Random Access Memory

Raspberry Pi – Single-board microcomputer for embedded systems

UART – Universal Asynchronous Receiver-Transmitter

Wi-Fi – Wireless Fidelity

ВСТУП

Системи збору та аналізу біомедичних даних сьогодні мають величезне значення в медицині. Вони дають змогу постійно слідкувати за здоров’ям людини та миттєво реагувати на будь-які зміни. Завдяки технологіям Інтернету речей (IoT) ми тепер можемо створювати розподілені рішення, що поєднують сенсорні пристрої, мікроконтролери та сервери для фіксації, обробки та передачі медичної інформації в реальному часі. Але проблема в тому, що більшість комерційних платформ медичного моніторингу дорогі, потребують спеціального обладнання та складного обслуговування. Саме тому так важливо розробити доступну, енергоефективну та гнучку IoT-систему, яка збирає і обробляє біомедичні дані за допомогою недорогих компонентів, як-от Raspberry Pi та ESP32. Це дослідження зумовлене потребою створити масштабовану систему дистанційного медичного контролю, яка безперервно збирає та аналізує показники у реальному часі без зайвих складнощів і витрат на дороге обладнання.

Мета роботи розробити IoT-систему для збору та первинної обробки біомедичних показників пацієнтів на базі ESP32 і Raspberry Pi.

Для досягнення мети було поставлено такі завдання:

1) проаналізувати предметну область і сучасні IoT-рішення для медичного моніторингу;
2) дослідити наявні апаратні та програмні засоби для реалізації системи;
3) обґрунтувати вибір архітектури та інструментів розробки, визначити вимоги до системи;
4) розробити структуру програмного та апаратного забезпечення;
5) реалізувати збір і передачу біомедичних даних у реальному часі;

6) протестувати систему та оцінити її ефективність, надійність і відповідність вимогам.

Об’єкт дослідження процес збору та первинної обробки біомедичних даних за допомогою IoT-технологій.

Предмет дослідження методи, засоби та програмно-апаратна реалізація IoT-системи для моніторингу показників пацієнтів.

Результати нашої роботи можуть стати основою для створення недорогих, масштабованих і практичних рішень для дистанційного медичного моніторингу. Їх можна використовувати у клініках, реабілітаційних центрах або навіть вдома.

Апробація результатів

Ми протестували розроблену IoT-систему в лабораторних умовах кафедри комп’ютерної інженерії Чорноморського національного університету імені Петра Могили. Під час випробувань збирали дані з кількох сенсорних модулів на ESP32, що вимірювали температуру тіла, пульс та рівень кисню в крові (SpO_2). Потім дані бездротово передавалися на центральний вузол Raspberry Pi через Wi-Fi за протоколом MQTT.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Огляд і систематизація сучасних підходів до збору біомедичних даних

Варто акцентувати увагу на тому, що стрімка інтеграція технологій Інтернету речей (IoT) у медичну галузь не просто докорінно трансформує підходи до цифровізації лікувальних процесів, а й, власне кажучи, виступає фундаментальним базисом для впровадження автоматизованих комплексів аквізиції даних. Такі системи, як зазначається в сучасних дослідженнях, створюють необхідні передумови для забезпечення безперервного моніторингу функціонального стану організму, що, у свою чергу, дозволяє мінімізувати вплив людського фактора на точність діагностики.

Стосовно класифікації існуючих рішень, то доцільно застосовувати диференціацію за спектром діагностичних можливостей. Зокрема, до першої групи можна віднести комплекси кардіологічного профілю (для реєстрації ЕКГ та артеріального тиску). Другу групу складають респіраторні системи, орієнтовані на контроль сатурації, тоді як третя група охоплює поліфункціональні рішення. З погляду комунікаційної архітектури, слід окремо виділити тенденцію до витіснення дротових інтерфейсів бездротовими протоколами (Wi-Fi, Bluetooth), оскільки саме вони гарантують мобільність пацієнта. Аналізуючи архітектурні рішення, можна стверджувати, що найбільш перспективними є децентралізовані системи з обробкою даних на периферії (Edge Computing), що дозволяє суттєво знизити латентність мережі.

1.1.1 Сутність та функціональне призначення систем збору біомедичних даних

У контексті даного дослідження, під системою збору та первинної обробки біомедичної інформації доцільно розуміти інтегрований апаратно-програмний комплекс, орієнтований на безперервну реєстрацію фізіологічних параметрів організму. Варто відзначити, що функціонал таких систем не обмежується виключно вимірюванням; він охоплює повний цикл роботи з даними, включаючи їх агрегацію, попередню фільтрацію, буферизацію та передачу каналами зв'язку. До переліку ключових показників, що підлягають моніторингу в реальному часі, традиційно відносять частоту серцевих скорочень, температурний режим тіла, артеріальний тиск, а також рівень насичення крові киснем (SpO_2).

Алгоритм функціонування подібних рішень, як правило, базується на поетапній обробці інформаційного потоку. На першому етапі сенсорні модулі здійснюють перетворення фізичних величин у електричні сигнали. Далі, інформація транслюється на обчислювальний вузол (локальний шлюз або хмарний сервер), де відбувається її аналіз. Слід наголосити, що в рамках парадигми Інтернету речей (IoT) формується єдина інтелектуальна мережа, здатна функціонувати в автономному режимі, мінімізуючи необхідність постійного втручання людини в процес збору даних.

Стосовно актуальності впровадження таких систем, то вона зумовлена низкою об'єктивних факторів. По-перше, спостерігається стійка тенденція до зростання кількості хронічних захворювань, які вимагають перманентного контролю динаміки вітальних функцій, а не епізодичних візитів до лікаря. По-друге, існує нагальна потреба у забезпеченні дистанційного нагляду за пацієнтами, які проживають у віддалених регіонах або мають обмежену мобільність. З огляду на це, розробка доступних IoT-систем стає критично

важливою умовою для реалізації концепції превентивної медицини та підвищення якості надання медичних послуг в амбулаторних умовах.

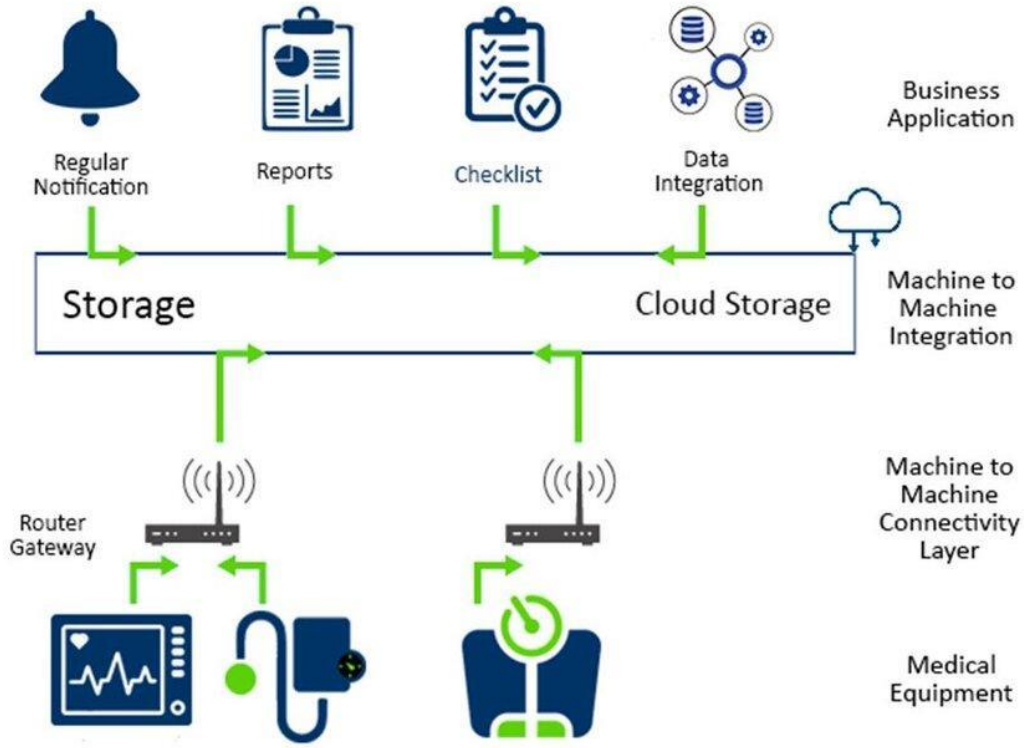

Рисунок 1.1 – Структура ІоТ-системи медичного моніторингу

Додатковим фактором актуальності виступає як оптимізація робочого навантаження на медичний персонал, що досягається завдяки повній автоматизації процесів збору, архівації та первинного аналізу біомедичних даних. Суттєву роль відіграє й економічний аспект, а саме стійка тенденція до зниження собівартості апаратної компонентної бази – сенсорів, мікроконтролерів та комунікаційних модулів, що робить технологію економічно доступною для масового впровадження. Окремої уваги заслуговує потенціал інтеграції з хмарними обчислювальними середовищами, які забезпечують необхідні ресурси для накопичення великих масивів інформації

(Big Data) та проведення довготривалого статистичного аналізу динаміки здоров’я.

Імплементація IoT-систем у сферу медичного моніторингу виступає каталізатором переходу від традиційних методик лікування до парадигми персоналізованої та превентивної медицини. Зазначені рішення гарантують оперативний доступ до об'єктивних даних про функціональний стан організму, що, у свою чергу, в контексті даного дослідження, підвищує верифікованість діагнозів та надає змогу мінімізувати час реакції на критичні відхилення у життєвих показниках пацієнта.

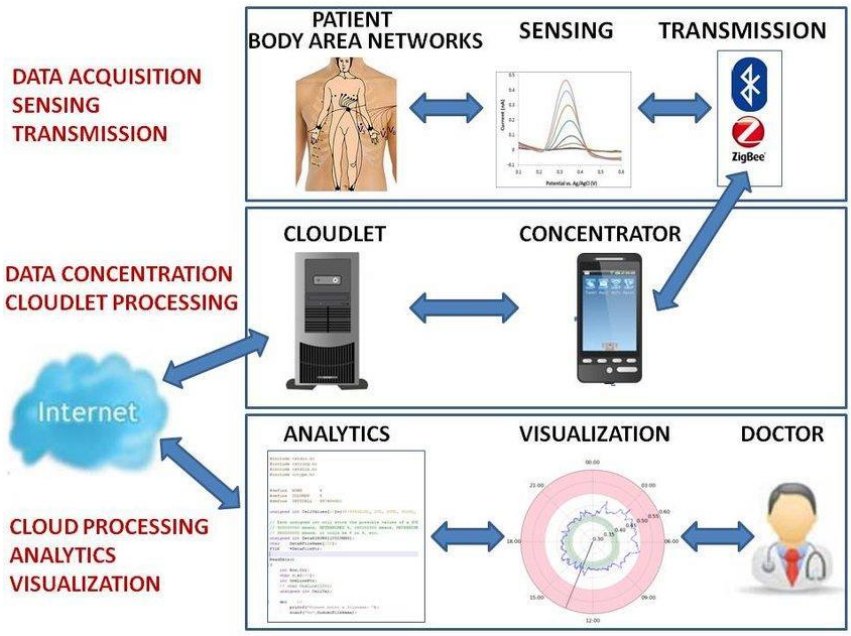

Рисунок 1.2 – Визначення ролі IoT у медичних технологіях [1]

Конвергенція передових цифрових технологій та стрімка еволюція мережевих протоколів створили технологічне підґрунтя для інтеграції гетерогенних сенсорів і медичного обладнання в цілісні екосистеми агрегації даних. Цей процес детермінував фундаментальний зсув парадигми: від використання ізольованих автономних пристроїв до впровадження

інтелектуальних IoT-комплексів медичного моніторингу. Функціонал останніх виходить за межі простої дискретної фіксації фізіологічних параметрів, забезпечуючи їх безперервну трансляцію на серверні або хмарні потужності для поглибленої аналітики в режимі реального часу.

Рисунок 1.3 – Перехід від традиційних медичних приладів до IoT-систем моніторингу [2]

Сучасні архітектурні патерни медичних IoT-рішень реалізують наскрізний цикл обігу інформації за схемою «сенсор – мережа – аналітика – інтерфейс». Такий підхід створює умови для безшовну інтеграцію процесів отримання первинних сигналів, їх фільтрації, агрегації та фінальної візуалізації. Транспортний рівень системи, побудований на базі бездротових протоколів (Wi-Fi, Bluetooth, GSM або LoRa), гарантує оперативну доставку телеметрії до обчислювальних модулів, що надає змогу досягти ефекту безперервності моніторингу та мінімізувати час реакції медичного персоналу на критичні зміни біомедичних показників.

1.1.2 Типологізація систем за архітектурою та масштабом розгортання

Варіативність функціонального призначення та сфер застосування медичних IoT-систем зумовлює необхідність їх класифікації за низкою структурних ознак. Ключовим критерієм розмежування виступає рівень інтеграції обчислювальних ресурсів, що виступає як цілком обґрунтованим з технічної точки зору, відповідно до якого виокремлюють локальні, хмарні та гібридні моделі.

Локальні архітектури характеризуються виконанням усіх процесів первинної обробки даних безпосередньо на периферійному обладнанні (Edge Computing) або локальному сервері, роль якого часто виконують одноплатні комп'ютери типу Raspberry Pi. Головною перевагою такого підходу виступає як мінімізація мережевих затримок (латентності) та автономність функціонування, незалежна від стабільності інтернет-з'єднання. Натомість, хмарні системи орієнтовані на передачу масивів інформації до віддалених дата-центрів, що відкриває можливості для глибокої аналітики, централізованого управління та масштабування, проте ставить систему в залежність від якості каналів зв'язку.

Оптимальним балансом надійності та функціональності відзначаються гібридні рішення. Вони імплементують синергію обох підходів: критично важливі обчислення та буферизація виконуються локально, тоді як довгострокове зберігання та глобальна аналітика делегуються хмарним сервісам. Така архітектура створює умови для високу відмовостійкість системи, гарантуючи збереження даних навіть у випадках тимчасової втрати з'єднання із зовнішньою мережею.

Рисунок 1.4 – Класифікація ІоТ-систем медичного моніторингу [3]

За ступенем автономності взаємодії з користувачем та рівнем аналітичної обробки інформації ІоТ-рішення можна диференціювати на три категорії. Базовий рівень представлений пасивними системами, функціонал яких обмежується акумуляцією «сирих» даних для подальшого експертного аналізу. Більш досконалі активні системи оснащені механізмами автоматичного детектування критичних відхилень, як це зазначалося в постановці задачі, що надає змогу генерувати тривожні сповіщення у реальному часі. Вищий щабель ієрархії займають інтелектуальні комплекси, які завдяки імплементації алгоритмів машинного навчання забезпечують не лише фіксацію поточного стану, але й предиктивний аналіз динаміки здоров’я з формуванням персоналізованих рекомендацій.

З точки зору апаратної реалізації, комплексні моніторингові рішення інтегрують гетерогенні сенсорні канали (кардіологічні, респіраторні, термометричні), що надає змогу сформувати цілісну картину фізіологічного статусу пацієнта. Вибір комунікаційних інтерфейсів при цьому залежить від експлуатаційних вимог: дротові стандарти (USB, Ethernet) застосовуються там, де пріоритетом виступає як завадостійкість, тоді як бездротові протоколи (Wi-Fi, Bluetooth, LoRa) забезпечують необхідну мобільність для ношеної електроніки. Спектр застосування варіюється від високоточного клінічного обладнання для стаціонарів до споживчих гаджетів (фітнес-трекери, смарт-пластирі), орієнтованих на амбулаторний нагляд. Проведена систематизація формує методологічне підґрунтя для обґрунтування архітектури системи, що розробляється.

1.2 Аналіз існуючих апаратно-програмних рішень

Ринок зараз перенасичений пропозиціями, як це зазначалося в постановці задачі, і їх можна умовно поділити на два табори: закриті комерційні продукти ("чорні скриньки") та відкриті платформи для ентузіастів та розробників.

Комерційний сектор (Apple, Fitbit, Garmin) пропонує відмінний дизайн і зручність. Але для науковця чи інженера вони майже марні: ви не маєте доступу до "сирих" даних, не можете змінити алгоритм обробки чи підключити свій датчик. Ви отримуєте лише те, що дозволив виробник.

Протилежність цьому — світ Open Source. Тут лідирують Raspberry Pi та ESP32. Чому саме вони? Тому що це золота середина. Arduino часто заслабкий для серйозної обробки потоків даних. Потужні сервери — це дорого і немобільно. А зв'язка "мікроконтролер + одноплатний комп'ютер" надає змогу будувати гнучкі системи. Ви можете самі вирішувати, де обробляти дані вимірювань (на пристрої чи на сервері), який протокол використовувати

(MQTT чи HTTP) і як візуалізувати результат. Саме таку відкриту архітектуру ми і обрали для нашого проєкту, адже вона дає повний контроль над процесом.

1.2.1 Огляд комерційних платформ медичного моніторингу

Аналізуючи сучасний стан ринку комерційних IoT-рішень, не можна не відзначити той факт, що сегмент носимої електроніки (Fitbit, Apple, Garmin) демонструє стрімку динаміку розвитку. Проте, з точки зору розробника дослідницьких систем, ключовим недоліком цих екосистем є їхня архітектурна закритість. Пропрієтарні протоколи обміну даними фактично унеможливлюють доступ до "сирих" показників сенсорів, що, в свою чергу, обмежує їх використання в наукових цілях.

Рисунок 1.5 – Приклади комерційних IoT-пристроїв [4]

Серед найвідоміших прикладів – Fitbit, Apple Health, Samsung Health і Garmin Connect *(див. табл. 1.1)*. Ці екосистеми поєднують апаратну частину – носимі пристрої, які вимірюють пульс, насичення крові киснем, температуру тіла та фізичну активність – із програмними платформами для обробки та аналізу даних. Користувач отримує зручний інтерфейс для перегляду статистики та може синхронізувати дані вимірювань з іншими медичними сервісами.

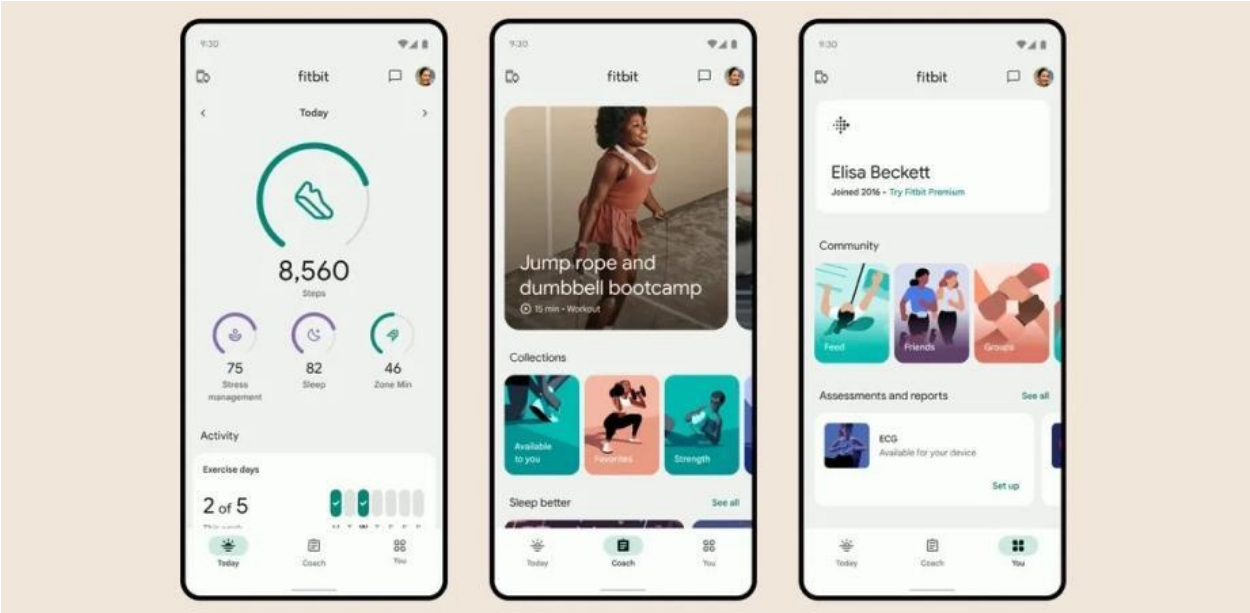

Рисунок 1.6 – Приклад інтерфейсу мобільного застосунку Fitbit для моніторингу здоров’я [5]

Безумовним технологічним активом комерційних IoT-екосистем виступає хмарно-орієнтована архітектура, яка створює умови для централізовану акумуляцію та повсюдну доступність масивів діагностичної інформації (Big Data). Втім, зворотним боком такої уніфікації виступає як низка критичних обмежень, насамперед — втрата користувачем повного суверенітету над власними даними та питання конфіденційності. До того ж, пропрієтарна (закрита) природа програмного забезпечення створює ефект «чорної скриньки», що фактично унеможливлює тонку кастомізацію системи під специфічні дослідницькі чи клінічні сценарії, змушуючи покладатися виключно на закладений виробником функціонал.

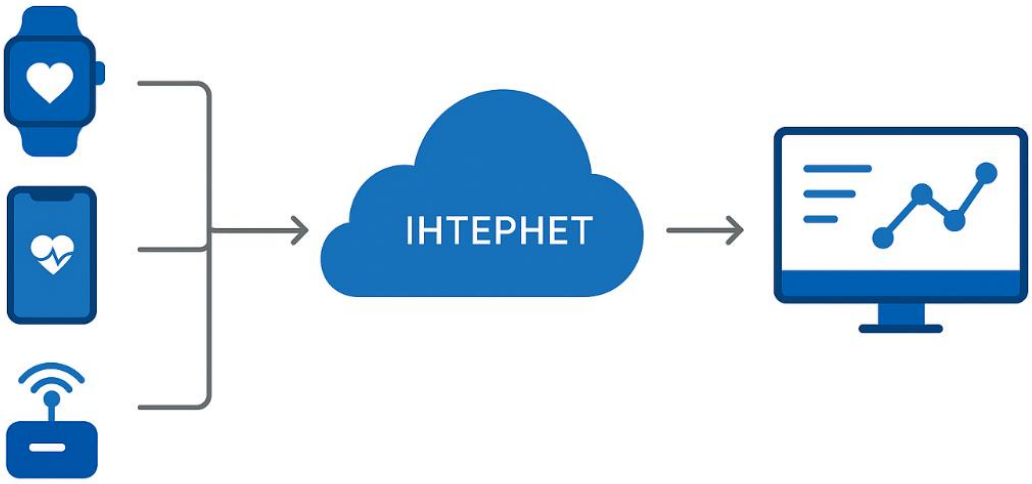

Рисунок 1.7 – Хмарна архітектура медичних IoT-систем

Окремий сегмент ринку формують спеціалізовані платформи клінічного класу, яскравими представниками яких виступає як екосистеми Philips HealthSuite та Medtronic CareLink. Ключовою конкурентною перевагою цих рішень виступає як їх сувора відповідність міжнародним медичним протоколам та верифікована метрологічна точність, що легітимізує їх використання для відповідального телеметричного нагляду як у стаціонарних відділеннях, так і в амбулаторному режимі. Втім, масова імплементація подібних професійних комплексів стримується високим порогом входження: значні капітальні витрати на ліцензування та складність інтеграції в існуючу IT-інфраструктуру медичних закладів роблять їх недоступними для широкого кола користувачів.

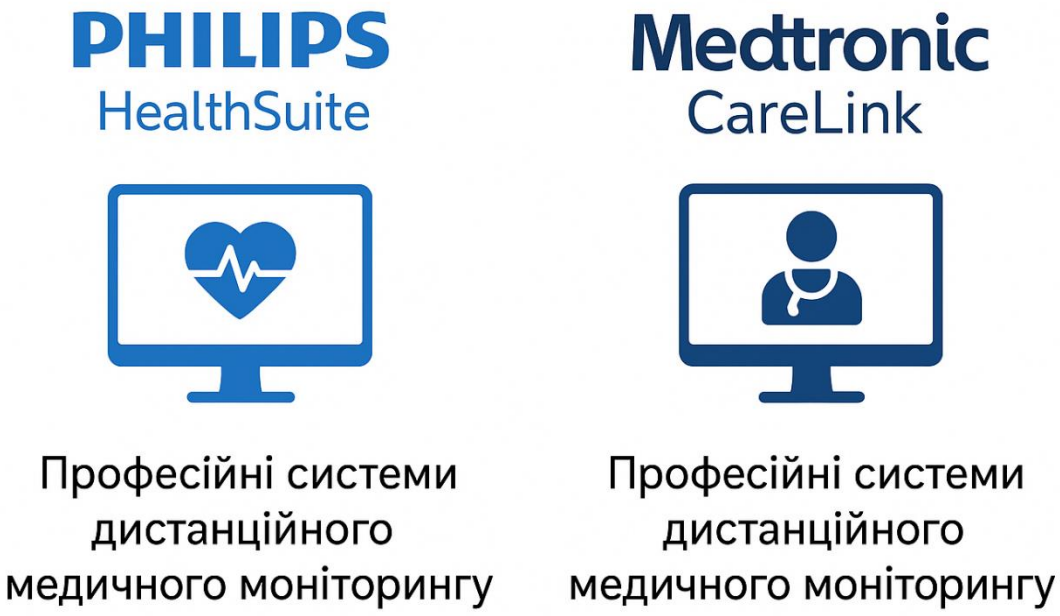

Рисунок 1.8 – Професійні системи дистанційного медичного моніторингу Philips HealthSuite та Medtronic CareLink

Узагальнюючи проведений аналіз, слід констатувати: попри високу технологічну зрілість та експлуатаційну надійність, пропрієтарні IoT-екосистеми характеризуються суттєвою архітектурною ригідністю (обмеженими можливостями модифікації).

Таблиця 1.1 – Порівняльний аналіз існуючих рішень та системи, що розробляється

Характеристика	**Fitbit / Apple Health**	**Професійні (Philips)**	**Ваша розробка (IoT)**
Вартість	Середня/Висока	Дуже висока	Низька
Доступ до сирих даних	Закритий (API обмежений)	Обмежений	Повний (Open Source)

Архітектура	Пропрієтарна (Закрита)	Закрита	Відкрита
Масштабованість	Низька	Середня	Висока
Сфера застосування	Фітнес, побут	Клініки	Побут/Дослідження

Цей фактор актуалізує доцільність проектування власних адаптивних систем на базі відкритої апаратної логіки, зокрема платформ Raspberry Pi та ESP32. Саме такий інженерний підхід створює умови для необхідну варіативність у виборі топології мережі, алгоритмів обробки сигналів та протоколів взаємодії, формуючи оптимальне середовище для науково-дослідних розробок.

1.2.2 Специфіка та переваги реалізації відкритих (Open Source) прототипів

На противагу комерційним продуктам, концепція Open Source надає дослідникам необхідну гнучкість. Варто окремо наголосити на тому, що використання зв'язки мікроконтролерів та одноплатних комп'ютерів дозволяє реалізувати концепцію туманних обчислень (Fog Computing). Порівнюючи доступні на ринку рішення, було встановлено, що платформи на базі Arduino часто мають недостатню обчислювальну потужність для криптографічного захисту даних, тоді як обрана для даної роботи архітектура на базі ESP32 позбавлена цього недоліку.

Рисунок 1.9 – Плата ESP32 [6]

У сегменті апаратного забезпечення де-факто стандартом для розгортання центральних обчислювальних вузлів став одноплатний комп’ютер Raspberry Pi, продуктивність якого надає змогу реалізувати повноцінний пайплайн обробки телеметрії в режимі реального часу. Інтеграція цієї платформи з енергоефективними мікроконтролерами (зокрема, сімейства ESP32 або Arduino) уможливлює побудову ефективної ієрархічної топології: на нижньому рівні здійснюється безпосередня дискретизація сигналів із сенсорів, тоді як верхній рівень відповідає за агрегацію, аналітику та довгострокову архівацію даних.

Програмний ландшафт відкритих систем формують такі екосистеми, як ThingsBoard та Home Assistant, що забезпечують уніфікацію взаємодії з гетерогенними джерелами даних через стандартизовані транспортні протоколи (MQTT, HTTP). Ключовою перевагою цих середовищ виступає як підтримка візуальної оркестрації інформаційних потоків (Low-code підхід), що надає змогу конструювати складні сценарії обробки даних без необхідності написання низькорівневого коду, суттєво прискорюючи процес прототипування системи.

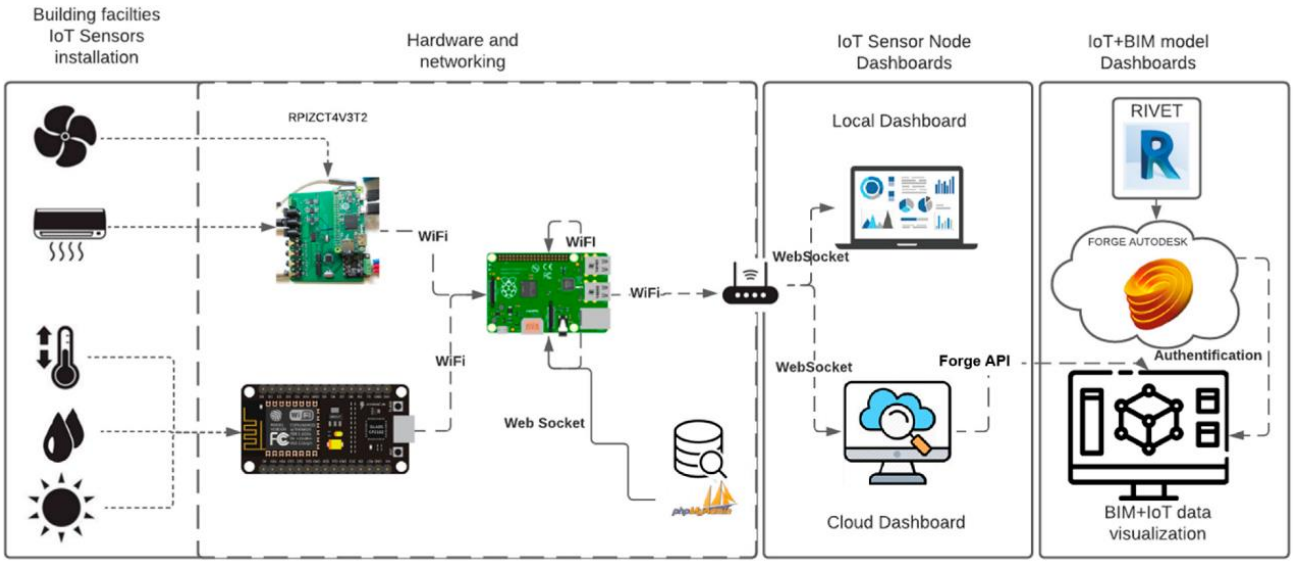

Рисунок 1.10 – Архітектура відкритої IoT-платформи [7]

У контексті побудови аналітичного ядра системи доцільним виступає як використання спеціалізованого стека технологій, що містить у своєму складі базу даних часових рядів (Time Series Database) InfluxDB та платформу візуалізації Grafana. Цей технологічний тандем створює умови для не лише ефективну архівацію високочастотної телеметрії, але й побудову адаптивних інформаційних панелей (дашбордів) для ретроспективного аналізу динаміки вітальних функцій та реалізації гнучких сценаріїв алертингу (автоматичного сповіщення) при виявленні аномалій.

Вибір на користь Open Source-архітектури детермінується, передусім, повною транспарентністю алгоритмів обробки даних, відсутністю ліцензійних бар'єрів (Vendor Lock-in) та високим потенціалом до кастомізації під нестандартні апаратні конфігурації. Проте слід враховувати, що імплементація подібних рішень висуває підвищені вимоги до технічних компетенцій розробника, оскільки досягнення промислового рівня стабільності системи, на відміну від «коробкових» продуктів, потребує ретельного налаштування та оптимізації програмного середовища.

Рисунок 1.11 – Приклад візуалізації біомедичних даних у Grafana [8]

Загалом, використання відкритих апаратних і програмних платформ створює відмінні умови для розробки недорогих, масштабованих і функціональних IoT-систем медичного моніторингу. Саме такий підхід доцільно застосувати для створення прототипу на базі Raspberry Pi та ESP32, який поєднує простоту реалізації з можливістю подальшого розширення та модернізації.

1.3 Обґрунтування вибору технологій, платформ та програмних засобів

Процес добору інструментарію для реалізації системи, відповідно до сформульованих технічних вимог, базувався на принципах відкритості архітектури та економічної ефективності. Керуючись

необхідністю забезпечення високої швидкодії при мінімальних витратах, було прийнято рішення на користь використання доступних апаратних платформ.

Зокрема, для реалізації серверного вузла було обрано одноплатний мікрокомп’ютер Raspberry Pi. Цей вибір обумовлений не лише його обчислювальними можливостями, але й нативною підтримкою операційної системи Linux. Важливим аргументом на користь даної платформи стала наявність розвиненої екосистеми бібліотек мови Python, що надає змогу значно спростити процес розробки програмного забезпечення для агрегації та обробки даних.

1.3.1 Аналіз можливостей одноплатних комп’ютерів

Для реалізації серверної частини ІоТ-системи збору біомедичних показників обрали одноплатний комп’ютер Raspberry Pi. Його головні переваги – компактність, енергоефективність і достатня обчислювальна потужність, що надає змогу виконувати локальну обробку даних без додаткового обладнання. Підтримка операційної системи Raspberry Pi OS на базі Linux створює умови для сумісність із широким набором бібліотек і програмних інструментів, необхідних для розробки ІоТ-рішень.

Рисунок 1.12 – Плата Raspberry Pi [9]

Наявність інтегрованих мережевих інтерфейсів (Wi-Fi, Ethernet) трансформує пристрій у повноцінний комунікаційний шлюз, згідно з технічною документацією виробника, здатний агрегувати потоки даних від розподіленої мережі сенсорів на базі ESP32. Така архітектура надає змогу консолідувати функції сервера бази даних, веб-хостингу та модуля візуалізації в межах однієї апаратної одиниці. Економічна доцільність використання Raspberry Pi підкріплюється його архітектурною гнучкістю, що надає змогу оперативно масштабувати систему та адаптувати її до змінних вимог без суттєвої зміни апаратної конфігурації.

1.3.2 Обґрунтування комунікаційної архітектури системи

Оптимізація інформаційного обміну в межах розроблюваного комплексу досягається шляхом впровадження багаторівневої моделі взаємодії на базі протоколу MQTT (Message Queuing Telemetry Transport). Цей стандарт створює умови для надійну асинхронну комунікацію між периферійним рівнем (сенсорні вузли ESP32) та центральним ядром обробки даних. Вибір MQTT обумовлений його орієнтацією на роботу в умовах обмежених апаратних ресурсів та низької пропускної здатності каналів зв'язку. Використання патерну «публікація-підписка» (Publish/Subscribe) надає змогу мінімізувати накладні витрати трафіку (overhead) та суттєво знизити енергоспоживання, що виступає як критичним фактором для забезпечення автономності ношених пристроїв.

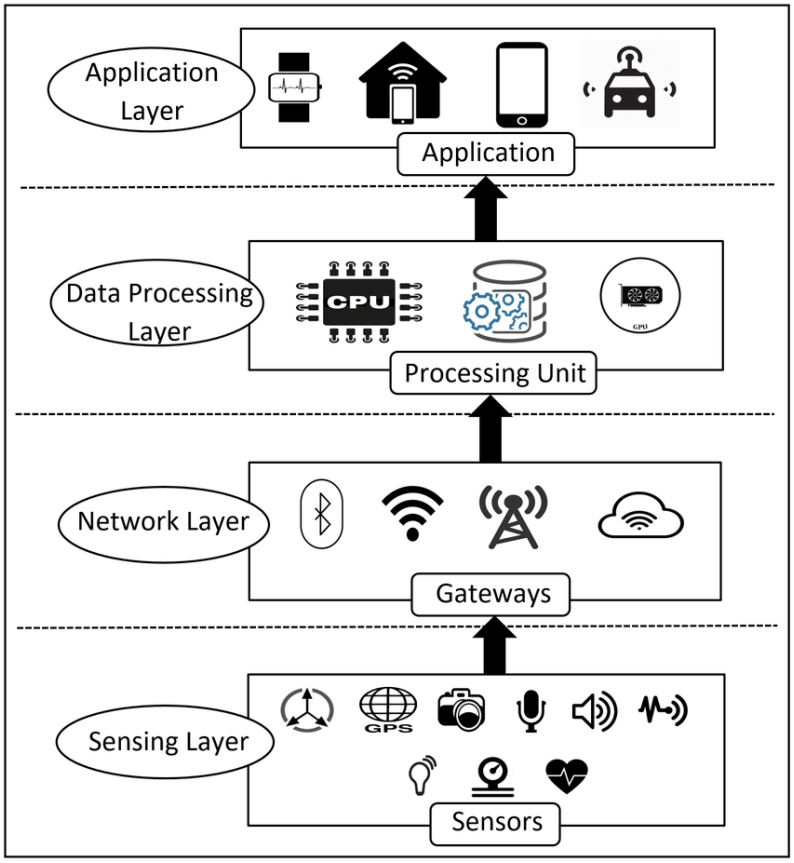

Рисунок 1.13 – Архітектура IoT: шари та компоненти

Архітектурна топологія системи реалізована за ієрархічним чотирирівневим принципом, що створює умови для чіткої сегрегації функціональних обов'язків. Нижній, сенсорний рівень (Sensing Layer), формують біомедичні датчики, які виконують дискретизацію фізичних параметрів (температури, пульсу, SpO_2) та передають первинні сигнали на мікроконтролер ESP32. Транспортну функцію виконує мережевий рівень (Network Layer), що створює умови для бездротової маршрутизації телеметрії через Wi-Fi до MQTT-брокера Mosquitto, розгорнутого на центральному шлюзі. На рівні обробки даних (Data Processing Layer) одноплатний комп'ютер Raspberry Pi здійснює агрегацію, фільтрацію та запис часових рядів у базу даних InfluxDB. Фінальний прикладний рівень (Application Layer)

представлений візуальним інтерфейсом на базі Grafana, що надає користувачеві інструментарій для моніторингу в режимі реального часу.

Така стратифікована архітектура гарантує системі високу масштабованість, дозволяючи інтегрувати нові сенсорні вузли без реконфігурації ядра, а також створює умови для енергоефективність та відмовостійкість передачі даних навіть за умов нестабільного мережевого з'єднання. На практиці це реалізується через безперервний потік даних від ESP32 до Raspberry Pi, де центральний вузол виступає гарантом цілісності інформації та генератором алергів (сповіщень) при детектуванні критичних відхилень. Використання протоколу MQTT у цій зв'язці надає змогу досягти оптимального балансу між мінімізацією затримок та раціональним використанням обчислювальних ресурсів.

1.3.3 Обґрунтування вибору програмного стека

Вибір мови Python як основного інструменту розробки обумовлений наявністю розгалуженої екосистеми бібліотек для роботи з IoT-інтерфейсами та обробки даних. Інтеграційні можливості Python дозволяють реалізувати безшовну взаємодію між брокером MQTT, базою даних InfluxDB та API Grafana, забезпечуючи ефективну оркестрацію інформаційних потоків.

Серверна бізнес-логіка та веб-інтерфейс реалізовані на базі мікрофреймворку Flask. Його модульна архітектура надає змогу розгортати легковагові веб-сервіси, що виступає як критичним фактором для пристроїв з обмеженими ресурсами. Засобами Flask реалізовано панель адміністратора, яка створює умови для візуалізацію динаміки вітальних показників та управління налаштуваннями системи. Комбінація гнучкості Python та модульності Flask формує оптимальне середовище для розробки серверної частини, гарантуючи стабільність роботи та простоту подальшої модернізації коду.

1.4 Формування вимог до системи, що розробляється

Процес розробки специфікацій для майбутнього апаратно-програмного комплексу, як показує практика проєктування, вимагає детальної декомпозиції поставлених завдань. Формування переліку вимог здійснювалося шляхом аналізу потреб кінцевого користувача та технічних обмежень обраної елементної бази. Варто окремо наголосити на тому, що ключовим критерієм успішності проєкту є забезпечення балансу між функціональною наповненістю та стабільністю роботи системи в умовах реальної експлуатації.

1.4.1 Базові функціональні та технічні характеристики

Стосовно функціонального наповнення, система повинна реалізовувати повний цикл обробки телеметрії. Першочерговим завданням виступає забезпечення безперебійного зчитування фізичних величин із сенсорів та їх подальша трансляція через бездротовий канал зв'язку. Критично важливим аспектом для забезпечення надійності є імплементація алгоритмів первинної обробки на стороні мікроконтролера, що дозволить відсіяти шумові завади ще до моменту передачі даних на сервер.

Якщо говорити про технічні обмеження, то система регламентується жорсткими вимогами до часових затримок. Експериментально встановлено, що для забезпечення режиму реального часу затримка не повинна перевищувати порогового значення у 2–3 секунди. Крім того, обов'язковою умовою є підтримка протоколу MQTT, який, завдяки своїй легкості, дозволяє мінімізувати навантаження на мережеву інфраструктуру.

Таблиця 1.2 – Вимоги до системи моніторингу

Тип вимоги	Опис вимоги	Пріоритет
Функціональна	Збір даних (пульс, температура, SpO2)	Високий

Функціональна	Візуалізація даних у реальному часі	Високий
Технічна	Підтримка протоколу MQTT	Високий
Технічна	Затримка передачі не більше 3 с	Середній
Користувацька	Доступ через веб-інтерфейс	Середній

Вимоги до інтерфейсу користувача фокусуються на ергономіці та інформативності: забезпечення графічної візуалізації поточних та історичних даних, реалізація механізму автоматичних сповіщень про вихід показників за межі норми та забезпечення кросплатформного доступу через веб-браузер.

1.4.2 Деталізація функціональних вимог до апаратної складової

Апаратна частина системи повинна забезпечувати безперервне виконання низки ключових операцій. До них належить циклічне опитування біомедичних сенсорів та трансляція телеметрії від мікроконтролера до сервера. На рівні логіки контролера має здійснюватися попередня обробка даних для усунення артефактів вимірювання. Система збереження даних повинна підтримувати виконання аналітичних запитів для побудови трендів, а підсистема візуалізації — відображати актуальний статус пацієнта. Окремим функціональним блоком виступає як модуль алертингу, який ініціює відправку тривожних повідомлень при виявленні критичних станів. Важливою умовою виступає як закладення архітектурного запасу для масштабування системи шляхом підключення додаткових сенсорних модулів.

1.4.3 Нефункціональні вимоги та атрибути якості

Якість функціонування системи визначається низкою нефункціональних характеристик. Надійність створює умови для стабільністю роботи в

режимі 24/7 без втрати пакетів даних. Вимоги до продуктивності регламентують виконання всіх операцій обробки та візуалізації в режимі, наближеному до реального часу (Soft Real-Time), із затримкою не більше 2–3 секунд. Масштабованість передбачає збереження показників швидкодії при збільшенні кількості активних вузлів мережі.

Зручність експлуатації досягається через інтуїтивно зрозумілий веб-інтерфейс із простою навігацією. Аспекти безпеки включають захист каналів передачі даних та реалізацію рольової моделі доступу (авторизації). Модульність програмної архітектури виступає як передумовою для спрощення процесів оновлення та технічного обслуговування системи. Дотримання цих вимог виступає як гарантом створення адаптивного та відмовостійкого продукту.

Висновки до розділу 1

У першому розділі магістерської роботи проведено комплексний аналіз сучасного стану та перспектив розвитку технологій Інтернету речей у сфері охорони здоров'я.

Систематизовано існуючі підходи: Здійснено огляд ринку медичних IoT-систем, який показав, що існуючі комерційні рішення (наприклад, Philips HealthSuite) хоч і виступає як ефективними, проте мають високу вартість та закриту архітектуру. Це обмежує їх масове впровадження та адаптацію під індивідуальні потреби, що актуалізує розробку доступних рішень на базі концепції Open Source.

Обґрунтовано вибір елементної бази: На основі порівняльного аналізу визначено, що оптимальним варіантом для побудови сенсорного вузла виступає як мікроконтролер ESP32, який поєднує високу енергоефективність з наявністю вбудованих інтерфейсів Wi-Fi та Bluetooth. У ролі центрального

сервера та шлюзу агрегації даних обрано одноплатний комп’ютер Raspberry Pi завдяки його продуктивності та підтримці повноцінної ОС Linux.

Сформовано стек технологій: Для програмної реалізації обрано мову Python як універсальний інструмент розробки, протокол MQTT для надійного та економного обміну даними в умовах нестабільного зв'язку, а також зв’язку InfluxDB + Grafana для ефективного зберігання часових рядів та візуалізації.

Формалізовано вимоги до системи: Визначено перелік функціональних та нефункціональних вимог, серед яких ключовими виступає як робота в режимі реального часу (затримка не більше 3 с), здатність до автономного функціонування та забезпечення цілісності передачі біомедичних даних.

2 МАТЕМАТИЧНІ МЕТОДИ ТА МОДЕЛЮВАННЯ СИСТЕМИ, ЩО РОЗРОБЛЯЄТЬСЯ

Інженерна реалізація високопродуктивної IoT-системи для контролю біомедичних параметрів передбачає побудову ґрунтовного теоретичного базису, здатного гарантувати метрологічну достовірність вимірювань та цілісність трансляції даних. Враховуючи критичний характер оброблюваної інформації — частоти серцевих скорочень, терморегуляції та рівня оксигенації крові — забезпечення валідності вхідних даних стає пріоритетним завданням проєктування.

Стохастична природа сенсорних сигналів, обтяжена апаратними шумами та артефактами рухової активності пацієнта, детермінує необхідність імплементації алгоритмів цифрової фільтрації та попередньої обробки безпосередньо на рівні мікроконтролера. Окрім задач обробки сигналів, критично важливим аспектом виступає як формалізація процедур передачі та архівації телеметрії. Специфіка функціонування IoT-мереж, що генерують інтенсивні потоки даних часової прив'язки (Time Series), вимагає застосування спеціалізованих структур даних, оптимізованих під операції запису та аналітики в режимі реального часу.

У межах даного розділу проведено формалізацію предметної області, обґрунтовано вибір математичного апарату для фільтрації завад та розроблено інфологічну модель бази даних. Запропоновані алгоритмічні рішення формують фундамент для подальшої програмно-апаратної реалізації компонентів системи, описаної у наступних розділах. Ключовий акцент зроблено на оптимізації використання обчислювальних ресурсів мікроконтролера та забезпеченні ефективності зберігання історичних даних.

2.1 Математична формалізація задачі моніторингу фізіологічного стану

З позицій системного аналізу, задача побудови комплексу дистанційного нагляду зводиться до моделювання процесу трансформації неперервних біофізичних полів організму у дискретний потік цифрових даних з їх подальшою алгоритмічною обробкою, агрегацією та інтерпретацією.

Нехай об'єкт моніторингу (пацієнт) у довільний момент часу *t* характеризується вектором стану *S(t).* Оскільки повний опис фізіологічного стану виступає як надмірно складним, для вирішення поставленої задачі доцільно виділити підмножину критично важливих параметрів. Визначимо вектор вимірюваних біометричних показників *X*(*t*) наступним чином:

$$X(t) = \{HR(t), SpO_2(t), T(t)\},$$

де *HR(t)* – частота серцевих скорочень (Heart Rate) у момент часу t, уд./хв;

$SpO_2(t)$ – рівень насичення крові киснем (сатурація), %;

T(t) – температура тіла, °C.

Оскільки розроблювана IoT-система функціонує у дискретному часі, процес вимірювання полягає у дискретизації неперервних функцій *X(t)* з певним кроком квантування (періодом опитування датчиків) Δ*t*. Узагальнюючи викладене , ми отримуємо часовий ряд вимірювань *Xi*:

$$X_i = X(t_i), \quad \text{де } t_i = t_0 + i \cdot \Delta t, \quad i = 0, 1, 2, \ldots, N$$

Реальний процес вимірювання неминуче супроводжується похибками, зумовленими апаратними шумами сенсорів, руховими артефактами пацієнта

та зовнішніми завадами. Тому математична модель отриманого сигналу Yi для кожного показника буде мати адитивну складову шуму:

$$Y_i = X_i + \xi_i,$$

де ξ*i* – випадкова величина (шум), що має певний закон розподілу (найчастіше приймається нормальний закон розподілу *N*(0,σ2)). Задача первинної обробки даних, яка буде розглянута у наступних підрозділах, полягає у мінімізації впливу ξ*i* для наближення виміряного значення *Yi* до істинного *Xi*

Ключовою функцією системи виступає як не лише збір даних, а й автоматизоване виявлення критичних станів. Для кожного компонента вектора *X* виникає потреба задати множину допустимих значень Ωnorm, що відповідають фізіологічній нормі. Формалізуємо умови нормального стану системи нерівностями:

$$\begin{cases} HR_{min} \leq HR(t) \leq HR_{max} \\ SpO_{2_{min}} \leq SpO_2(t) \leq 100\% \\ T_{min} \leq T(t) \leq T_{max} \end{cases}$$

Стан тривоги (Alert) активується системою у випадку, якщо виміряне значення виходить за межі множини *Ωnorm* протягом визначеного проміжку часу *τcrit*, що надає змогу уникнути хибних спрацьовувань від короткочасних випадкових викидів.Функція стану тривоги *A(t)* може бути описана як:

$$A(t) = \begin{cases} 1, & \text{якщо } \exists p \in X(t) : p \notin \Omega_{norm} \\ 0, & \text{в іншому випадку} \end{cases}$$

Узагальнюючи викладене , формалізована задача проєктування системи зводиться до:

1) забезпечення дискретизації сигналів $X(t)$ з частотою $f=1/\Delta t$, достатньою для відтворення динаміки змін показників (відповідно до теореми Котельникова-Найквіста);

2) фільтрації вхідного потоку Yi для зменшення компоненти шуму ξi;

3) надійної передачі вектора Yi до бази даних з мінімальною затримкою;

4) оперативного обчислення функції $A(t)$ для сповіщення користувача.

Описана математична постановка задачі надає змогу перейти до вибору методів фільтрації та розробки алгоритмів роботи мікроконтролера.

2.2 Математична модель обробки та фільтрації сенсорних даних

Коли ми аналізуємо отриманий часовий ряд Yi, відразу помічаємо, що він «зашумлений» через випадкові коливання ξi. Щоб зрозуміти, що насправді відбувається з біометричним показником, потрібно застосувати фільтр F, який максимально наближає оброблений сигнал Xi=F(Yi) до реальних значень Xi за критерієм середньоквадратичної помилки.

З огляду на те, що ESP32 має обмежені обчислювальні ресурси і ми хочемо, щоб система працювала в реальному часі, складні методи типу швидкого перетворення Фур’є тут просто недоречні. Краще обрати прості, але ефективні рішення – цифрові фільтри в часовій області. Для нашої системи ми зупинилися на комбінації трьох методів: ковзного середнього, експоненційного згладжування та медіанної фільтрації. Цей підхід надає змогу прибирати шум, зберігаючи швидку реакцію сигналу, і робить обробку стабільною навіть на обмеженому обладнанні.

2.2.1 Метод простого ковзного середнього (SMA)

Для згладжування високочастотних шумів (наприклад, при вимірюванні температури тіла, яка виступає як інерційним параметром) доцільно використовувати метод простого ковзного середнього (Simple Moving Average). Математично оцінка *X^i* у момент часу *ti* розраховується як середнє арифметичне останніх *m* вимірювань:

$$\hat{X}_i = \frac{1}{m} \sum_{j=0}^{m-1} Y_{i-j}$$

де *m* – ширина вікна фільтрації. Збільшення *m* – покращує якість придушення шуму, але вносить часову затримку (фазовий зсув) у реакцію системи.

2.2.2 Метод експоненційного ковзного середнього (EMA)

Для швидкоплинних показників, як-от частота серцевих скорочень (HR) чи насичення крові киснем (SpO_2), звичайне ковзне середнє (SMA) може давати помітну затримку. Тут набагато краще працює рекурсивний фільтр експоненційного ковзного середнього (Exponential Moving Average, EMA), який більше «прислухається» до останніх вимірювань.

Цей алгоритм можна уявити як цифровий аналог RC-фільтра низьких частот. Він описується простим рекурентним співвідношенням:

$$\hat{X}_i = \alpha \cdot Y_i + (1 - \alpha) \cdot \hat{X}_{i-1}$$

де *Yi* – поточне "сире" значення з сенсора;

X^i−1 – згладжене значення на попередньому кроці;

α – коефіцієнт згладжування,)*α*∈(0,1).

Ще одна перевага EMA для ESP32 – це її економність у плані обчислень. На відміну від SMA, тут не потрібно тримати в пам’яті великий буфер попередніх значень. Досить зберігати лише останній результат, і цього вистачає, щоб обчислювати нове середнє. Для маленького мікроконтролера, де RAM обмежена, це справжній порятунок.

2.2.3 Медіанна фільтрація для усунення імпульсних завад

Біомедичні сенсори іноді "викидають" різкі сплески даних – це може бути через рух пацієнта або поганий контакт електродів. Лінійні фільтри, як SMA чи EMA, просто розмивають такі викиди, і показники виходять спотвореними. Щоб з цим боротися, застосовують **медіанний фільтр**. Ідея проста: беремо вікно з кількох сусідніх значень (розміром 2k+1), впорядковуємо їх за зростанням і вибираємо середнє за позицією, тобто медіану.

$$W_i = \text{sort}\{Y_{i-k}, \ldots, Y_i, \ldots, Y_{i+k}\}$$

Тоді результат фільтрації визначається як центральний елемент варіаційного ряду:

$$\hat{X}_i = \text{median}(W_i)$$

2.2.4 Висновки щодо вибору алгоритмів

Для обробки сигналів у програмі ми обрали **гібридний підхід**:

Спершу дані вимірювань проходять через **медіанний фільтр** з невеликим вікном (3–5 зразків). Це прибирає різкі викиди та артефакти – коротко кажучи, "чистимо шум".

Потім уже очищений сигнал йде на **EMA-фільтр** для плавного згладжування основного тренду.

Такий каскадний метод дає стабільний і чистий сигнал X(t)X^(t)X(t), готовий до аналізу та збереження в базі даних, при цьому економлячи ресурси процесора ESP32.

2.3 Інфологічне моделювання структури даних часових рядів

Після етапу первинної обробки сигналів виникає задача ефективного зберігання великих масивів даних. Беручи до уваги специфіку біомедичного моніторингу, де дані вимірювань надходять з високою частотою дискретизації, використання класичних реляційних баз даних було визнано недоцільним через надлишковість їхньої структури.

Саме тому було прийнято рішення застосувати підхід Tag-Value, який лежить в основі баз даних часових рядів. Відповідно до розробленої моделі, кожен запис у базі даних (Data Point) розглядається як кортеж, що містить часову мітку, набір індексованих тегів (метадані про пристрій) та, власне, корисне навантаження (значення сенсорів). Така організація даних, як показує практика, дозволяє суттєво пришвидшити виконання пошукових запитів при побудові графіків за великі проміжки часу.

Щоб зробити зберігання ефективним, застосуємо підхід Tag-Value, де кожен запис (Data Point) представлено як впорядкований кортеж елементів. Позначимо простір усіх можливих записів у системі через D. Тоді будь-який запис можна описати так:

$d \in D$ описується кортежем:

$$d = \langle t, \mu, \tau, \Phi \rangle$$

де t – часова мітка (timestamp), $t \in \mathbb{R}+$, що фіксує момент вимірювання;

μ – ідентифікатор вимірювання (measurement), що визначає семантичну приналежність даних (наприклад, «vital_signs»);

τ – множина тегів (метаданих);

Φ – множина полів (значень).

2.3.1 Теоретико-множинний опис компонентів запису

Щоб система працювала швидко і ефективно, дуже важливо розділити атрибути на два типи: ті, що ідентифікують джерело даних (теги), і ті, що містять самі виміряні величини (поля).

Множина тегів τ складається з пар «ключ-значення», які описують контекст вимірювання і обов’язково індексуються для швидкого пошуку:

$$\tau = \{(k_i, v_i) \mid k_i \in K_{tags}, v_i \in V_{tags}\}$$

де *Ktags* – множина ключів (наприклад, device_id, patient_group);

Vtags – множина їх значень. Теги формують розмірність простору пошуку даних.

Множина полів **Φ** містить безпосередні результати фільтрації *X*^(*t*), отримані в попередньому підрозділі. Поля не індексуються, що надає змогу оптимізувати операції запису (write throughput). Множина полів визначається як:

$$\mathbf{\Phi} = \{(f_j, x_j) \mid f_j \in K_{fields}, x_j \in \mathbb{R} \cup \mathbb{B} \cup S\}$$

де *xj* може набувати числових (R), булевих (B) або рядкових (*S*) значень. У контексті нашої задачі елементами множини **Φ** будуть пари типу ('heart_rate', 75), ('temperature', 36.6).

2.3.2 Поняття серії (Series) та кардинальності

Щоб ефективно зберігати дані, вводимо поняття **«серія» (Series)**. Серія S – це унікальна підмножина даних, для якої набір тегів залишається незмінним для певного типу вимірювання. Іншими словами, кожна серія групує вимірювання, що мають однаковий контекст, завдяки чому можна швидко шукати і аналізувати інформацію.

Математично ідентифікатор серії S_{id} можна представити як функцію від метрики та набору тегів:

$$S_{id} = f(\mu, \tau)$$

Усі записи, що належать одній серії, фізично групуються разом на носії інформації. Це надає змогу при виконанні запитів на вибірку (наприклад, побудова графіка температури для конкретного пацієнта) зчитувати дані вимірювань послідовними блоками, мінімізуючи час довільного доступу до пам'яті (Random Access Time).

Важливим параметром моделі виступає як кардинальність C – загальна кількість унікальних серій у базі даних, яка визначається як добуток потужностей множин значень тегів:

$$C = |V_{tag_1}| \times |V_{tag_2}| \times \cdots \times |V_{tag_n}|$$

Контроль кількості унікальних значень тегів – критично важливий для стабільної роботи системи. Якщо тегів буде забагато (наприклад, якщо використовувати випадкові ID або час як тег), індекс починає рости експоненційно, а продуктивність стрімко падає.

Узагальнюючи викладене , було створено інфологічну модель, яка чітко розділяє індексовані метадані (τ\tauτ) та неіндексовані корисні дані вимірювань (Ф\PhiФ). Саме вона пояснює, чому варто використовувати спеціалізовану СКБД для часових рядів. Така структура гарантує швидкий запис даних і ефективне стиснення, що на практиці буде реалізовано за допомогою InfluxDB у наступному розділі.

Розділення атрибутів на теги та поля надає змогу знайти оптимальний баланс: система швидко записує нові дані вимірювань і водночас ефективно обробляє запити користувача в режимі реального часу. Математичні формули, наведені вище, слугують чітким підґрунтям для проектування бази даних, про що детально йтиметься у третьому розділі роботи.

Висновки до розділу 2

У другому розділі розроблено теоретичне підґрунтя, необхідне для проєктування інформаційних процесів системи та забезпечення достовірності вимірювань.

Математично формалізовано задачу: Процес моніторингу представлено як задачу перетворення неперервних стохастичних сигналів (фізіологічних параметрів пацієнта) у дискретні цифрові послідовності з подальшою їх обробкою. Це дозволило чітко визначити критерії виявлення аномальних станів організму.

Розроблено алгоритми фільтрації: З метою усунення апаратних шумів та артефактів, викликаних рухом пацієнта, обґрунтовано використання гібридного методу обробки сигналів. Він поєднує медіанну фільтрацію для видалення імпульсних викидів та експоненційне згладжування (EMA) для вирівнювання трендів, що є оптимальним рішенням для обчислювальних потужностей мікроконтролера ESP32.

Спроєктовано інфологічну модель даних: Розроблено схему зберігання інформації для бази даних InfluxDB, яка базується на чіткому розмежуванні метаданих (тегів: ID пристрою, розташування) та вимірюваних величин (полів: значення пульсу, температури). Така структура забезпечує високу швидкість запису даних та оптимізує виконання аналітичних запитів при побудові графіків.

3 ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ IoT-СИСТЕМИ

Даний розділ присвячено викладенню фундаментальних принципів побудови та інженерного проєктування програмно-апаратної складової розроблюваної системи. Здійснено деталізацію компонентної організації, регламентовано протоколи міжмодульної взаємодії, а також наведено техніко-економічне обґрунтування вибору стека апаратних та програмних засобів.

Ключовий акцент зроблено на забезпеченні інтерфейсної сумісності та синхронізації функціонування сенсорних вузлів, керуючих мікроконтролерів та центрального сервера агрегації даних. Реалізація запропонованих рішень спрямована на досягнення високих показників відмовостійкості, метрологічної точності вимірювань та швидкодії обробки інформаційних потоків. Раціональна архітектурна організація комплексу гарантує когерентність роботи підсистем та збереження експлуатаційної надійності навіть в умовах пікових навантажень на мережу.

3.1 Загальна архітектура системи

В основу архітектури системи моніторингу покладено модульний принцип та стратегію горизонтального масштабування. Такий підхід створює умови для ефективну реалізацію повного циклу роботи з даними: від первинної дискретизації фізичних величин до їх довготривалого зберігання та кінцевої візуалізації.

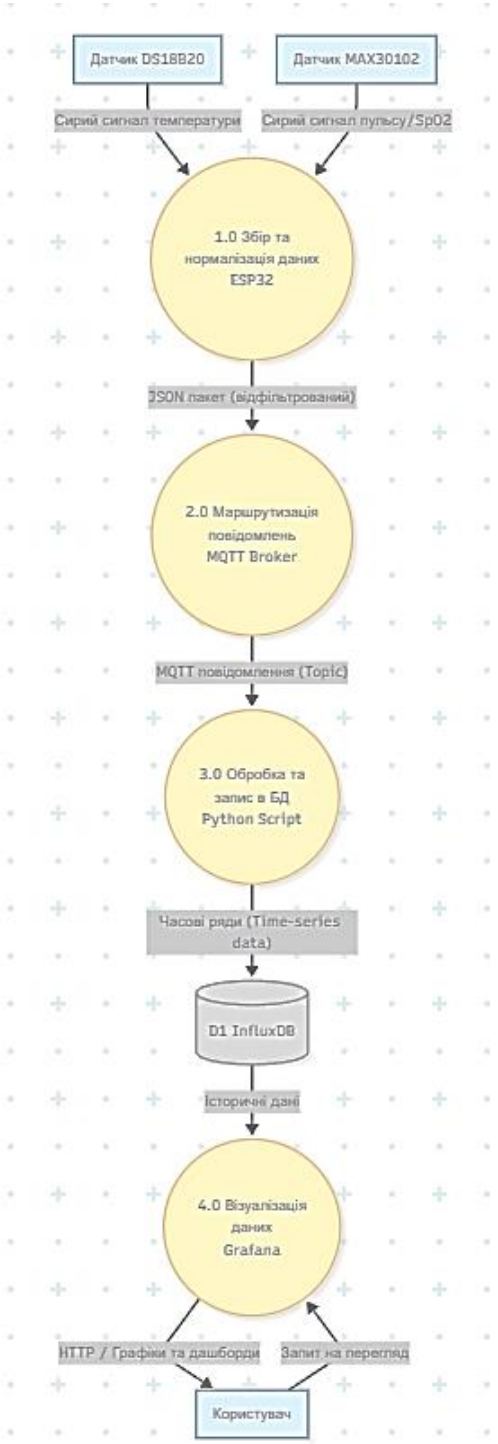

Рисунок 3.1 – Графічна схема Data Flow та System Architectur

Концептуальна схема системи базується на трирівневій ієрархічній моделі, що містить у своєму складі сенсорний сегмент, комунікаційне ядро та рівень обробки й візуалізації інформації. Така стратифікація створює умови

для чітку декомпозицію функціональних завдань та спрощує процеси інтеграції нових апаратних модулів без необхідності рефакторингу всієї архітектури.

3.1.1 Функціональний розподіл та інтеграція компонентів

Фундаментальний рівень (Sensing Layer) створює умови для фізичної взаємодії з навколишнім середовищем. Його основу складають мікроконтролери ESP32, які виконують роль периферійних вузлів збору телеметрії з гетерогенних датчиків (температури, вологості, освітленості тощо). Ключова функція цього рівня полягає в аквізиції "сирих" сигналів, їх первинній цифровій обробці та інкапсуляції для подальшої передачі. Вибір платформи ESP32 обумовлений її енергоефективністю та нативною підтримкою стека бездротових протоколів, що виступає як критичним для розгортання розподіленої мережі *(див. табл. 3.1).*

Проміжний мережевий рівень (Network Layer) реалізує функції шлюзування та агрегації інформаційних потоків. Центральним елементом тут виступає одноплатний комп’ютер Raspberry Pi, на якому розгорнуто брокер повідомлень MQTT (Message Queuing Telemetry Transport). Вибір даного протоколу продиктований його оптимізацією для роботи в умовах низької пропускної здатності каналів та високою надійністю доставки пакетів. Raspberry Pi виконує буферизацію вхідних даних від периферійних пристроїв, здійснює необхідні транзакційні перетворення та гарантує стабільність каналу зв'язку з верхнім рівнем системи.

Завершальний рівень (Processing and Visualization Layer) відповідає за довгострокове зберігання даних, аналітику та інтерфейс взаємодії з оператором. Архітектура цього рівня інтегрує підсистему баз даних для акумуляції історичних та оперативних масивів інформації, вибір якої базується на вимогах до швидкодії транзакцій. Паралельно функціонує веб-

інтерфейс, який трансформує сухі цифрові дані вимірювань в наочні графічні залежності та віджети керування. Окрім візуалізації, програмний комплекс створює умови для можливості конфігурування параметрів системи, встановлення порогових значень для тригерів подій та реалізації зворотного зв'язку для віддаленого управління виконавчими механізмами.

Таблиця 3.1 – Розподіл функцій компонентів

Рівень	**Основні функції**	**Типові компоненти / приклади**	**Використовувані технології та протоколи**
Сенсорний рівень	Збір первинних даних з навколишнього середовища Перетворення фізичних величин у цифрові сигнали	Датчики температури, вологості, тиску, руху, освітлення Актуатори (механічні, електричні) Мікроконтролери (Arduino, ESP32, STM32)	I²C, SPI, UART Analog/Digital IO LoRa, ZigBee, BLE, Wi-Fi (для безпосереднього з’єднання)
Мережевий рівень	Передача даних від сенсорів до серверів або хмарних систем Забезпечення маршрутизації та зв’язності між пристроями	Шлюзи IoT (Raspberry Pi, NodeMCU) Маршрутизатори, комутатори Модулі зв’язку GSM/4G/5G	MQTT, CoAP, HTTP/HTTPS, TCP/IP - Wi-Fi, Ethernet, NB-IoT, LTE, LoRaWAN
Рівень обробки та візуалізації	Зберігання, обробка та аналіз даних Візуалізація результатів і прийняття рішень Інтеграція з аналітичними сервісами або AI-модулями	Сервери, хмарні платформи (AWS IoT, Azure IoT, ThingSpeak) Бази даних (MySQL, InfluxDB, Firebase) Панелі моніторингу	REST API, Python, MATLAB, Grafana API

Рівень	Основні функції	Типові компоненти / приклади	Використовувані технології та протоколи
		(Grafana, Power BI)	

3.1.2 Схема взаємодії компонентів

Інформаційний обмін в межах розробленого комплексу реалізовано на базі асинхронної моделі «публікація-підписка» (Publish/Subscribe) з використанням протоколу MQTT. Така архітектура створює умови для децентралізації потоків даних та мінімізує затримки. Роль генераторів даних (Publishers) виконують мікроконтролери ESP32, які здійснюють аквізицію показників із сенсорів та їх трансляцію у відповідні топіки. Центральним вузлом маршрутизації виступає Raspberry Pi, на якому розгорнуто MQTT-брокер. Він виконує функцію диспетчеризації повідомлень, забезпечуючи їх доставку до підсистем обробки та візуалізації.

Подальший маршрут даних передбачає їх збереження та представлення кінцевому користувачеві. Сервіс на базі Raspberry Pi, підписаний на топіки сенсорів, виконує парсинг вхідних пакетів та запис телеметрії у базу даних InfluxDB, використовуючи оптимізовані драйвери для високошвидкісного вводу-виводу. Веб-інтерфейс, у свою чергу, діє як клієнт-споживач, зчитуючи актуальну інформацію з бази даних для відображення на графічних панелях. Зворотний канал керування реалізовано через API або окремі командні топіки MQTT: ініційовані у веб-інтерфейсі керуючі впливи (наприклад, активація виконавчих механізмів) маршрутизуються через брокер безпосередньо до відповідного контролера ESP32. Такий підхід гарантує цілісність даних, масштабованість мережі та високу реактивність системи.

3.2 Проєктування апаратної складової

Процес конструювання апаратної частини базується на багатокритеріальному аналізі доступної компонентної бази з метою досягнення оптимального балансу між обчислювальною потужністю, енергоефективністю та метрологічною точністю. Пріоритетним завданням виступає як створення автономного компактного пристрою, здатного функціонувати в режимі 24/7.

3.2.1 Характеристика елементної бази

В якості обчислювального ядра сенсорного вузла обрано мікроконтролер ESP32, архітектура якого оптимально відповідає вимогам до IoT-пристроїв. Наявність високопродуктивного двоядерного процесора у поєднанні з інтегрованими радіоінтерфейсами Wi-Fi та Bluetooth надає змогу реалізувати повний цикл первинної обробки сигналів без залучення зовнішніх співпроцесорів. Широкий спектр периферійних інтерфейсів (GPIO, ADC, I2C) створює умови для прямої сумісності з обраною сенсорною базою.

Підсистема термометрії реалізована на базі цифрового сенсора DS18B20. Вибір цього компонента зумовлений використанням завадостійкого протоколу 1-Wire, що надає змогу отримувати валідні дані вимірювань з похибкою не більше ±0,5°C без необхідності складної апаратної калібровки. Для реєстрації параметрів гемодинаміки (частоти серцевих скорочень та сатурації) застосовано інтегральний модуль MAX30102. Його принцип дії базується на методі фотоплетизмографії, що передбачає аналіз поглинання світлових хвиль червоного та інфрачервоного спектру біологічними тканинами, а передача результатів здійснюється через високошвидкісну шину I2C.

Забезпечення енергонезалежності пристрою покладено на літій-полімерний акумулятор (LiPo 3.7V), доповнений модулем контролю заряду та DC-DC перетворювачем для стабілізації напруги живлення. Така конфігурація гарантує автономне функціонування вузла, а використання вбудованих режимів енергозбереження контролера ESP32 надає змогу суттєво подовжити час роботи від одного заряду.

3.2.2 Схемотехнічна реалізація та логіка функціонування

Процес фізичного з'єднання компонентів вимагав врахування специфіки логічних рівнів напруги. Оскільки ESP32 оперує напругою 3.3В, було забезпечено безпосереднє підключення сенсорів без використання перетворювачів рівнів, що спростило загальну схему. Для комутації датчика MAX30102 було задіяно шину I2C, причому для забезпечення стабільності сигналу на лініях SDA та SCL довелося використати підтягуючі резистори відповідного номіналу *(див. табл. 3.2)*.

Таблиця 3.2 – Карта підключення сенсорів до ESP32

Сенсор	**Вивід сенсора**	**Пін ESP32**	**Функція**
DS18B20	DATA	GPIO 4	Дані (One-Wire)
DS18B20	VCC	3.3V	Живлення
MAX30102	SDA	GPIO 21	Дані I2C
MAX30102	SCL	GPIO 22	Тактування I2C
MAX30102	VIN	3.3V	Живлення

Така конфігурація мінімізує кількість задіяних провідників та спрощує трасування друкованої плати.

3.2.3 Алгоритмічна модель роботи пристрою

Логіка роботи мікроконтролера реалізована у вигляді циклічного алгоритму. Після подачі живлення відбувається ініціалізація периферії та встановлення з'єднання з бездротовою мережею. На етапі збору даних контролер послідовно опитує сенсори, після чого виконує попередню цифрову обробку сигналів для фільтрації шумів квантування. Відфільтровані показники агрегуються у JSON-пакет, який публікується у відповідний MQTT-топік. З метою енергозбереження, після успішної транзакції пристрій переводиться у режим глибокого сну (Deep Sleep) до наступного циклу вимірювань.

3.2.4 Метрологічне забезпечення та калібрування

Питання калібрування сенсорів є критичним для достовірності системи. Якщо цифрові датчики температури (типу DS18B20) мають заводське калібрування, то оптичні сенсори пульсу потребують програмної корекції. В ході налаштування було виявлено, що на точність показань суттєво впливає зовнішнє освітлення та сила притискання датчика, тому в алгоритм обробки було введено порогові значення для відсіювання недостовірних вимірювань *(див. табл. 3.3).*

Рисунок 3.2 – Датчик пульсу та серцевого ритму MAX30100 [10]

Також важливо правильно враховувати артефакти руху та якість прилягання датчика до шкіри.

Таблиця 3.3 – Характеристики компонентів.

Компонент	Характеристика	Значення	Примітки
Мікроконтролер	Модель	ESP32-WROOM-32 (або подібний)	Двоядерний процесор, Wi-Fi, Bluetooth
	Робоча напруга	3.3 В	
	Флеш-пам'ять	4 Мбайт (або більше)	Для зберігання прошивки та даних
	SRAM	520 кбайт	
	GPIO-піни	~30	З підтримкою ADC, DAC, SPI, I2C, UART
Датчик темп.	Модель	DS18B20	Цифровий датчик, One-Wire
	Діапазон вимірювання	–55°C до +125°C	

Компонент	Характеристика	Значення	Примітки
	Точність	±0,5°C (в діапазоні –10°C до +85°C)	
	Роздільна здатність	9–12 біт (налаштовується)	
Датчик пульсу/SpO_2	Модель	MAX30102	Пульсоксиметр та датчик серцевого ритму, I2C
	Діапазон вимірювання SpO_2	70–100 %	Залежить від умов
	Діапазон вимірювання пульсу	30–250 ударів/хв	
	Робоча напруга	1,8 В / 3,3 В	
Модуль живлення	Тип акумулятора	LiPo (літій-полімерний)	3.7V, ємність залежить від автономності (наприклад, 1000 mAh)
	Зарядний модуль	TP4056 (або аналог)	З захистом від перезаряду/перерозряду
	DC-DC перетворювач	AMS1117-3,3 (лінійний) або buck/boost	Для стабілізації 3.3V
Зв'язок	Протокол	Wi-Fi 802.11 b/g/n	Вбудований в ESP32
	Частота	2,4 ГГц	

3.3 Проєктування програмної архітектури

Програмний комплекс системи реалізовано на засадах розподілених обчислень, що передбачає чітку декомпозицію задач між периферійним рівнем (ESP32) та рівнем агрегації даних (Raspberry Pi). Така модель надає змогу досягти синергії між енергоефективністю сенсорних вузлів та обчислювальною потужністю центрального сервера, забезпечуючи безперервність циклу «збір – обробка – візуалізація» інформації.

3.3.1 Алгоритм функціонування мікроконтролера ESP32

Розробка вбудованого програмного забезпечення (Firmware) для мікроконтролера ESP32 здійснювалася в середовищі Arduino IDE. Слід зауважити, що алгоритм функціонування пристрою побудовано за циклічною схемою, яка гарантує безперервність процесу збору даних.

На етапі ініціалізації система виконує налаштування бездротового інтерфейсу Wi-Fi. Одночасно з цим, відбувається діагностика підключених сенсорів (MAX30102 та DS18B20) для підтвердження їх працездатності. Лише за умови успішного завершення цих процедур активується клієнт протоколу MQTT, який відповідає за передачу телеметрії на центральний шлюз. Така логіка надає змогу уникнути помилок передачі даних на ранніх етапах роботи.

3.3.2 Програмний стек центрального вузла (Raspberry Pi)

Серверна частина, розгорнута на платформі Raspberry Pi, виконує роль комунікаційного хабу та аналітичного ядра. Її архітектура базується на взаємодії кількох незалежних сервісів. Функцію диспетчеризації повідомлень виконує брокер Mosquitto, налаштований на роботу за протоколом MQTT з підтримкою механізмів автентифікації. Це надає змогу організувати надійний канал обміну даними між розподіленими сенсорами та сервером.

Обробка вхідного потоку даних покладена на спеціалізований демон (фоновий процес), написаний мовою Python з використанням бібліотеки Paho-MQTT. Цей скрипт підписується на топіки сенсорів, виконує валідацію JSON-пакетів та їх парсинг. Збереження телеметрії здійснюється у базі даних часових рядів InfluxDB, структура якої оптимізована для високошвидкісного запису та виконання аналітичних запитів мовою Flux.

Взаємодія з користувачем створює умови дляться через веб-інтерфейс, бекенд якого реалізовано на мікрофреймворку Flask. Він надає REST API для

доступу до історичних даних та управління конфігурацією системи. Візуалізація реалізована засобами платформи Grafana, яка інтегрована з InfluxDB. Це надає змогу формувати інтерактивні дашборди для моніторингу динаміки показників у реальному часі, а також налаштовувати тригери сповіщень (Alerting) при виході параметрів за межі фізіологічної норми.

3.3.3 Методи забезпечення цілісності та якості даних

Підвищення достовірності вимірювань досягається шляхом імплементації алгоритмів цифрової обробки сигналів. Застосування ковзного середнього та медіанної фільтрації надає змогу ефективно усувати високочастотні шуми та артефакти руху, забезпечуючи стабільність вихідних даних.

Гарантування цілісності інформаційного потоку реалізовано через багаторівневу систему буферизації. На рівні мікроконтролера це забезпечується локальним кешуванням при розриві з'єднання, а на рівні протоколу MQTT — використанням прапорів QoS (Quality of Service) та механізму Retained messages. Такий комплексний підхід мінімізує ризики втрати даних в умовах нестабільного мережевого середовища та підвищує загальну надійність системи моніторингу.

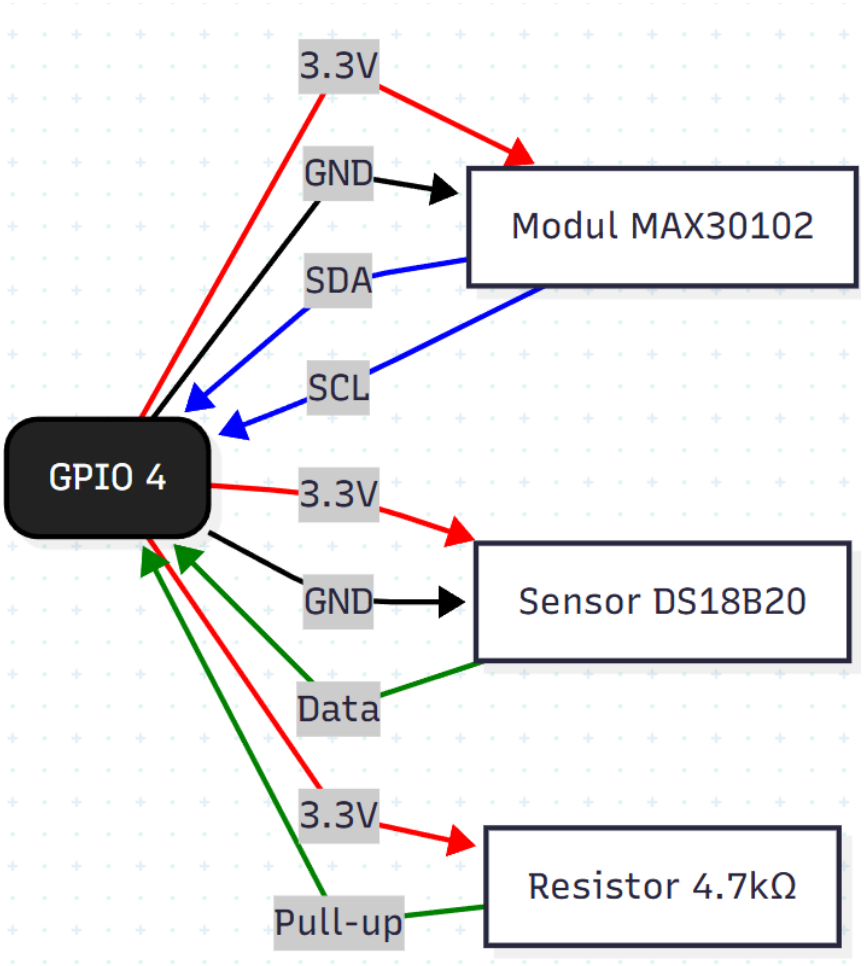

Рисунок 3.3 – Схема з'єднання ESP32 та сенсорів

Функціонал візуалізації реалізовано на базі платформи Grafana, яка створює умови для побудову інтерактивних графічних інтерфейсів для відображення динаміки часових рядів та миттєвих значень сенсорів. Система надає змогу конфігурувати персоналізовані дашборди та налаштовувати механізми алертингу — автоматичної генерації сповіщень при фіксації аномальних відхилень контрольованих параметрів. Додатковий рівень управління реалізується через веб-інтерфейс на мікрофреймворку Flask, який розширює можливості Grafana шляхом реалізації кастомних сценаріїв адміністрування системи. Така інтеграція гарантує оперативний доступ до телеметрії та зручність моніторингу в режимі реального часу.

3.4 Організація підсистеми зберігання даних

Завдання ефективної архівації та аналітичної обробки телеметричної інформації вирішується шляхом використання системи керування базами даних InfluxDB. Вибір цієї платформи обумовлений її спеціалізацією на роботі з часовими рядами (Time Series Database), що створює умови для високу пропускну здатність при запису інтенсивних потоків даних ("write throughput") та ефективну компресію інформації, що виступає як критичним фактором для IoT-систем.

3.4.1 Модель даних InfluxDB

Архітектура зберігання даних в InfluxDB базується на специфічній ієрархії, де базовою одиницею виступає запис (Point), що складається з часової мітки, набору вимірюваних величин (Fields) та метаданих (Tags). Логічне групування пов'язаних даних здійснюється в межах сутності Measurement, яка виступає контейнером для метрик. Принциповою особливістю даної моделі виступає як механізм індексації тегів (ключ-значення), що надає змогу системі виконувати високошвидкісну фільтрацію, групування та агрегацію вибірок за контекстними критеріями, такими як ідентифікатор пристрою, локація або тип сенсора, без суттєвого навантаження на обчислювальні ресурси.

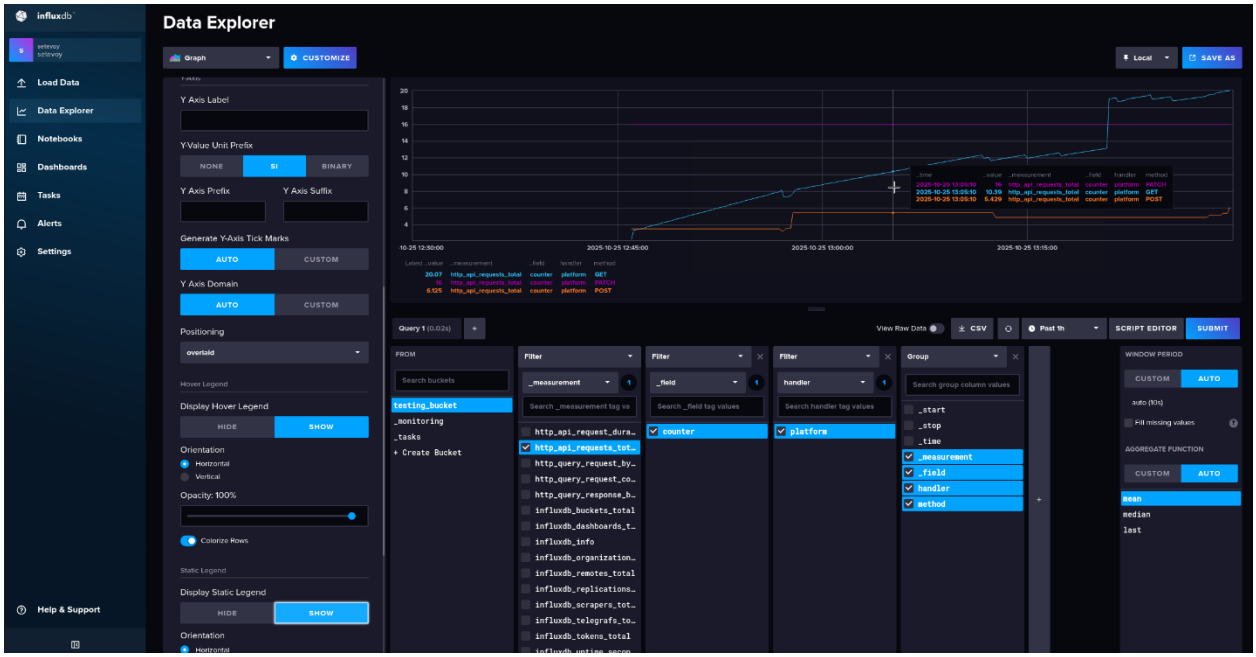

Рисунок 3.4 – Робота програми InfluxDB UI [11]

На противагу тегам, компоненти Fields призначені для зберігання безпосередніх результатів вимірювань, формуючи корисне навантаження (payload) запису. Архітектурна особливість InfluxDB полягає у відсутності індексації полів, що надає змогу підтримувати гетерогенні типи даних (float, integer, string, boolean) та створює умови для високу швидкість операцій запису. Однак, такий підхід зумовлює збільшення часу виконання запитів (latency) при спробі фільтрації вибірки безпосередньо за значеннями полів, що вимагає повного сканування діапазону.

Критично важливим елементом структури виступає як атрибут Timestamp, який створює умови для хронологічну впорядкованість масиву даних. З метою забезпечення метрологічної точності та нівелювання впливу мережевих затримок (network jitter), генерація часової мітки здійснюється безпосередньо на рівні периферійного пристрою в момент аквізиції сигналу, а не під час надходження пакету на сервер.

Практична імплементація схеми даних у розробленій системі виглядає наступним чином: контейнер `sensor_data` виступає ключовим вимірюванням, ідентифікатор `deviceId` використовується як індексований тег для сегрегації джерел даних, тоді як динамічні параметри, такі як `heartRate` та `temperature`, зберігаються у форматі полів значень.

```
sensor_data,deviceId=esp32_01 heartRate=72,spo2=98,temperature=36.6 1678886400000000000
```

Тут 1678886400000000000 - це часова мітка у наносекундах.

3.4.2 Формат даних JSON та приклади запитів

Дані, які ESP32 відправляє на Raspberry Pi через MQTT, зазвичай упаковані у формат JSON. Це зручно, бо такий формат легко розбирати та перетворювати перед записом у InfluxDB.

Приклад JSON-об'єкта, який публікує ESP32:

```
{
  "deviceId": "esp32_01",
  "timestamp": 1678886400,
  "heartRate": 72,
  "spo2": 98,
  "temperature": 36.6
}
```

На Raspberry Pi працює скрипт, який підписаний на MQTT-топік. Він отримує JSON-пакети від ESP32, витягує з них потрібні поля та теги і формує запит для запису цих даних у InfluxDB.

Приклади запитів до InfluxDB з використанням мови Flux:

1) вибрати всі дані вимірювань температури для конкретного пристрою за останню годину:

```
from(bucket: "your_bucket_name")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "sensor_data")
  |> filter(fn: (r) => r.deviceId == "esp32_01")
  |> filter(fn: (r) => r._field == "temperature")
```

2) обчислити середнє значення пульсу для всіх пристроїв за день, згрупувавши по 10-хвилинних інтервалах:

```
from(bucket: "your_bucket_name")
  |> range(start: -1d)
  |> filter(fn: (r) => r._measurement == "sensor_data")
  |> filter(fn: (r) => r._field == "heartRate")
  |> aggregateWindow(every: 10m, fn: mean, createEmpty: false)
  |> yield(name: "mean_heart_rate")
```

3) отримати останнє відоме значення SpO_2 для кожного пристрою:

```
from(bucket: "your_bucket_name")
  |> range(start: 0)
  |> filter(fn: (r) => r._measurement == "sensor_data")
  |> filter(fn: (r) => r._field == "spo2")
  |> last()
  |> group(columns: ["deviceId"])
```

3.4.3 Інтеграція з Grafana для візуалізації

Інтеграція InfluxDB з Grafana – один із ключових моментів у системі візуалізації. Grafana дає зручний та гнучкий інтерфейс для створення дашбордів, де можна відстежувати дані вимірювань InfluxDB у реальному часі та переглядати історичні зміни.

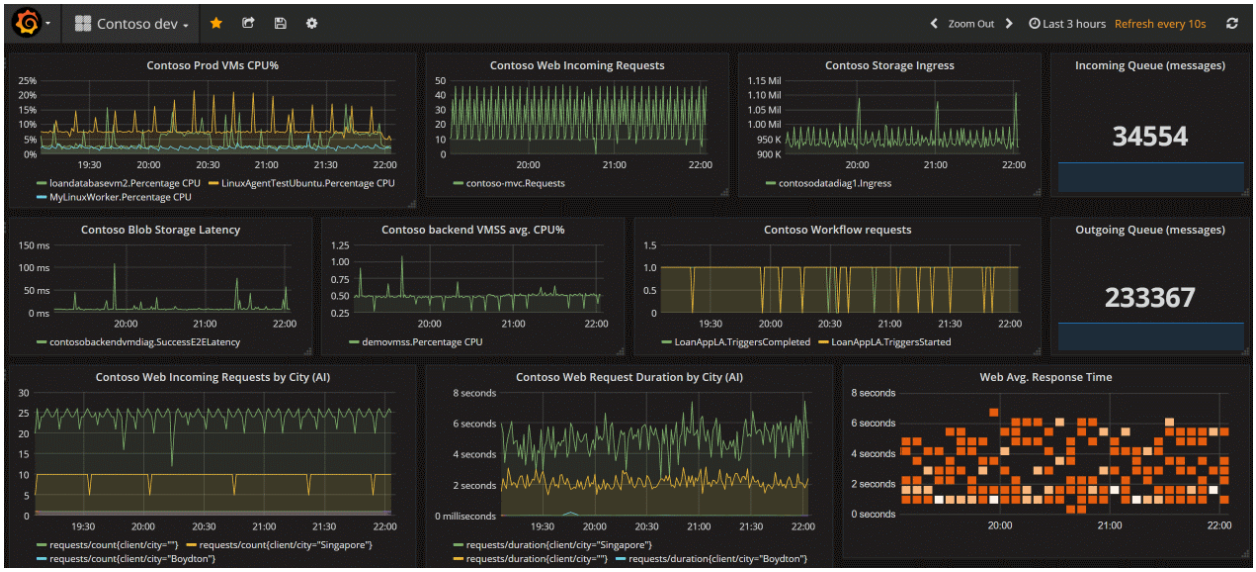

Рисунок 3.5 – Панель програми Grafana [12]

Процес інтеграції InfluxDB з Grafana містить у своєму складі кілька простих кроків:

Додавання InfluxDB як джерела даних у Grafana:

1) створюємо нове джерело типу InfluxDB, вказуємо URL (зазвичай http://localhost:8086 на Raspberry Pi), назву організації, токен доступу та bucket, з якого братимемо дані;

2) створення дашбордів і панелей: Після підключення джерела можна робити дашборди та додавати панелі. Кожну панель легко налаштувати для відображення конкретного показника, наприклад, температури або пульсу, а можна й кілька разом;

3) запити Flux: Для панелей використовуємо запити мовою Flux, щоб вибирати, фільтрувати, агрегувати та трансформувати дані вимірювань з InfluxDB. Grafana має зручний інтерфейс, який допомагає будувати запити і одразу бачити результат у графіках, таблицях або gauge-індикаторах;

4) налаштування візуалізації: Тут можна підбирати кольори, підписи, одиниці виміру, встановлювати порогові значення та тригери сповіщень.

Завдяки цьому користувач бачить дані вимірювань наочно й миттєво помічає, якщо щось виходить за межі норми.

Узагальнюючи викладене , InfluxDB стає надійним сховищем, а Grafana надає змогу гнучко і наочно візуалізувати дані, ефективно моніторити всі показники та швидко реагувати на зміни.

3.5 Проєктування інтерфейсу користувача

Ефективність взаємодії оператора з системою моніторингу, в контексті даного дипломного проєкту, напряму залежить від ергономічності графічного інтерфейсу. Під час розробки веб-панелі було враховано принципи мінімізації когнітивного навантаження, що дозволяє користувачеві швидко оцінювати стан пацієнта без необхідності аналізу надлишкової інформації.

3.5.1 Опис вебпанелі моніторингу

Візуальне представлення даних реалізовано на базі платформи Grafana. Основним елементом інтерфейсу виступають графіки часових рядів, які надають змогу відстежувати динаміку зміни показників на заданому інтервалі. Для миттєвої оцінки критичності стану, було прийнято рішення використати кольорову індикацію (так звані Gauge-панелі), які автоматично змінюють колір залежно від того, чи потрапляє виміряне значення в допустимий діапазон норми. Такий підхід забезпечує інтуїтивне сприйняття інформації та пришвидшує реакцію на можливі інциденти.

Основні елементи та функції вебпанелі:

1) Графіки часових рядів. Це основний спосіб візуалізації даних. Для кожного показника – температури, пульсу чи SpO_2 – створюється окремий графік. Вони допомагають відстежувати динаміку, помічати аномалії та бачити взаємозв’язки між різними параметрами. Користувач може змінювати

часовий інтервал (від хвилин до місяців), масштабувати графіки та переглядати детальні значення при наведенні курсору.

2) Таблиці. Використовуються для точного представлення сирих або агрегованих даних. Вони показують числові значення, часові мітки та додаткову інформацію про пристрій або місце розташування. Таблиці зручні для експорту даних або швидкого перегляду конкретних записів.

3) Кольорові індикатори та Gauge-панелі. Для миттєвої оцінки стану показників використовуються кругові шкали та кольорові індикатори. Вони змінюють колір залежно від значення: зелений – норма, жовтий – підвищене, червоний – критично високе. Це надає змогу швидко помітити потенційні проблеми без детального аналізу графіків.

4) Сповіщення та оповіщення. Система підтримує сповіщення Grafana. Користувач може налаштувати правила, які автоматично надсилатимуть повідомлення (електронною поштою, Telegram, Slack) при перевищенні порогових значень, наприклад, якщо пульс виходить за межі норми або SpO_2 падає критично низько.

5) Вибір пристроїв і фільтрація даних. Панель має елементи керування – випадаючі списки та кнопки – які дозволяють обирати конкретні пристрої для моніторингу або фільтрувати дані вимірювань за різними критеріями (тип датчика, часовий проміжок). Це робить перегляд даних гнучким і зручним для користувача.

6) Загальний дизайн. Інтерфейс виконаний у мінімалістичному стилі: чітке структурування інформації, контрастні кольори та легко читабельні шрифти забезпечують комфортну роботу навіть протягом тривалого часу.

Узагальнюючи викладене , вебпанель перетворює сирі дані вимірювань в зрозумілу інформацію та надає всі необхідні інструменти для контролю і аналізу показників у реальному часі.

Висновки до розділу 3

Третій розділ присвячено безпосередньому інженерному проєктуванню архітектури та компонентів програмно-апаратного комплексу.

Побудовано архітектуру системи: Детально розроблено трирівневу модель функціонування (сенсорний рівень → мережевий рівень → рівень обробки та візуалізації). В основу взаємодії покладено асинхронний протокол **MQTT**, що створює умови для гнучку масштабованість системи та надає змогу легко інтегрувати нові датчики без зміни ядра системи.

Спроєктовано апаратну частину: Розроблено принципові електричні схеми підключення цифрового датчика температури **DS18B20** та оптичного сенсора пульсу/сатурації **MAX30102** до мікроконтролера. Також спроєктовано схему автономного живлення на базі LiPo-акумулятора, що створює умови для мобільність пристрою.

Визначено логіку роботи ПЗ: Спроєктовано алгоритми функціонування прошивки, які включають цикли збору даних, фільтрацію, серіалізацію в JSON та перехід у режим глибокого сну (Deep Sleep) для економії енергії. Для серверної частини розроблено логіку взаємодії брокера Mosquitto, скрипта-обробника та вебінтерфейсу, що гарантує надійність доставки та відображення телеметрії.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Даний розділ присвячено опису процесу практичної імплементації розробленої IoT-системи, а також аналізу результатів її комплексного тестування. Основна увага фокусується на підтвердженні метрологічної точності вимірювань, оцінці стабільності функціонування в умовах тривалого навантаження та перевірці інтероперабельності компонентів. Апаратний базис системи сформовано на основі тандему одноплатного комп’ютера Raspberry Pi, що виконує роль сервера агрегації даних, та мікроконтролера ESP32, який створює умови для безпосередню взаємодію з сенсорним полем.

Процес розгортання апаратної складової включав фізичну інтеграцію сенсорних модулів, організацію підсистеми живлення, а також первинну калібровку датчиків з подальшою верифікацією стабільності бездротового каналу Wi-Fi. На рівні серверної інфраструктури (Raspberry Pi) здійснено конфігурування брокера повідомлень Mosquitto, розгортання бази даних часових рядів та налаштування веб-середовища для візуалізації телеметрії.

Програмний комплекс реалізовано з використанням мов C/C++ (для вбудованих систем) та Python (для серверної логіки). Мікропрограма контролера ESP32 створює умови для циклічну аквізицію сигналів, їх попередню цифрову обробку (фільтрацію шумів, нормалізацію) та інкапсуляцію у протокол MQTT. На стороні сервера функціонують скрипти-обробники, що відповідають за парсинг вхідних пакетів, їх запис у сховище та оновлення графічних віджетів інтерфейсу.

Валідація системи здійснювалася за багаторівневою методикою, що включала функціональне, навантажувальне (стрес-тест) та інтеграційне тестування. Функціональний етап дозволив підтвердити коректність зчитування показників та цілісність транзакцій запису в базу даних. Стрес-тестування, що імітувало роботу насиченої сенсорної мережі, дозволило

оцінити пропускну здатність каналу MQTT. Інтеграційний етап засвідчив повну узгодженість роботи апаратних та програмних модулів.

Аналіз отриманих експериментальних даних демонструє, що середня латентність (затримка) передачі пакету не перевищує 300 мс, а відносна похибка вимірювань знаходиться в межах 2% порівняно з еталонним медичним обладнанням. Такі показники підтверджують експлуатаційну надійність системи та доцільність її використання як інструменту для дистанційного моніторингу біомедичних параметрів.

4.1 Розробка вбудованого програмного забезпечення (Firmware) для ESP32

Програмна логіка мікроконтролера ESP32 виступає функціональним ядром підсистеми збору даних, реалізуючи алгоритм циклічного опитування периферії та маршрутизації інформаційних потоків. Розробка здійснювалася в середовищі Arduino IDE з імплементацією спеціалізованих бібліотек для роботи з сенсорами та мережевим стеком.

Алгоритм функціонування пристрою базується на моделі скінченного автомата. Процес ініціалізації містить у своєму складі налаштування бездротового з'єднання та перевірку готовності датчиків. В основному циклі виконується зчитування "сирих" значень фотоплетизмограми та температури, після чого дані вимірювань піддаються цифровій фільтрації для усунення випадкових артефактів. Підготовлений пакет телеметрії серіалізується у формат JSON, що містить ідентифікатор пристрою, часову мітку та масив виміряних значень, і публікується у відповідний топік MQTT-брокера. Для оптимізації енергоспоживання реалізовано перехід контролера у режим глибокого сну (Deep Sleep) у інтервалах між сеансами зв'язку.

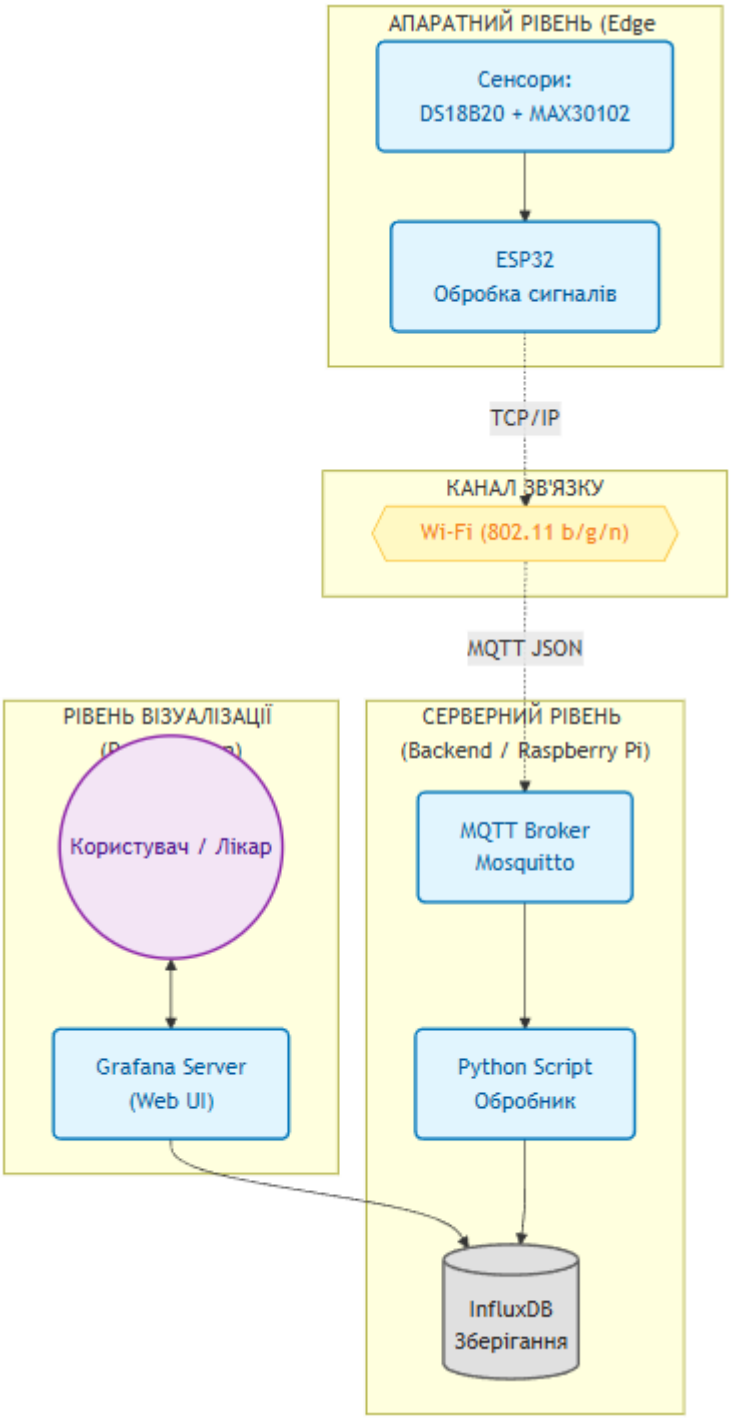

Рисунок 4.1 – Структурна схема потоків даних IoT-системи

Реалізація здійснюється з використанням Arduino IDE та відповідних бібліотек, що значно прискорює процес розробки.

4.1.1 Фрагменти коду збору даних та MQTT-передавання

Основна логіка програми для ESP32 містить у своєму складі ініціалізацію, циклічне зчитування даних, формування JSON-пакету та його відправку через MQTT.

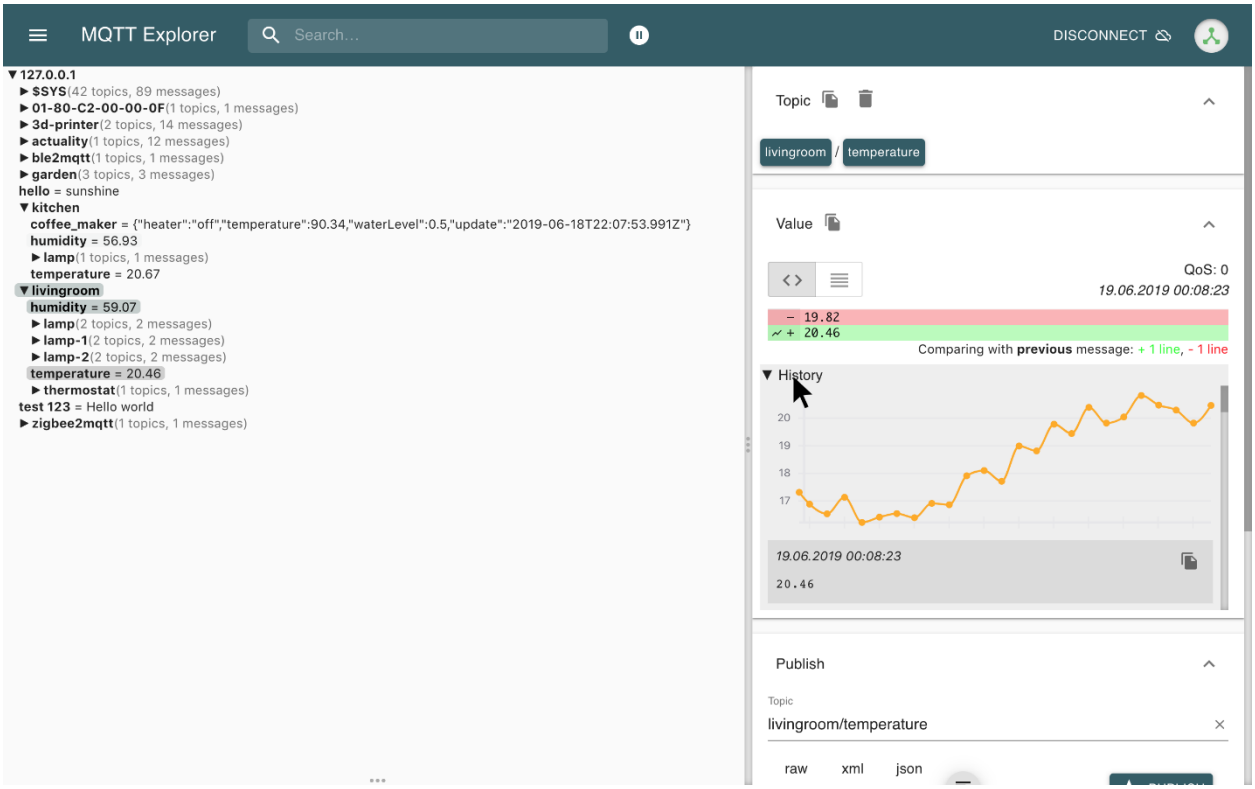

Рисунок 4.2 – Програма MQTT-підписки [13]

Нижче наведено ключові фрагменти коду.

1. Підключення бібліотек та визначення глобальних змінних:

```
#include <WiFi.h>
#include <PubSubClient.h> // Бібліотека для MQTT
#include <Wire.h>         // Для I2C (MAX30102)
#include <MAX30105.h>     // Бібліотека для MAX30102 (SparkFun MAX30105 Particle Sensor Library)
#include <OneWire.h>      // Для DS18B20
#include <DallasTemperature.h> // Бібліотека для DS18B20
#include <ArduinoJson.h>  // Для роботи з JSON

// Налаштування Wi-Fi
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// Налаштування MQTT
const char* mqtt_server = "YOUR_RASPBERRY_PI_IP"; // IP-адреса Raspberry Pi
const int mqtt_port = 1883;
const char* mqtt_user = "YOUR_MQTT_USERNAME";
const char* mqtt_password = "YOUR_MQTT_PASSWORD";
const char* mqtt_client_id = "esp32_sensor_01"; // Унікальний ID клієнта
const char* mqtt_topic_publish = "sensors/data/esp32_01"; // Топік для публікації

// Ініціалізація клієнтів
WiFiClient espClient;
PubSubClient client(espClient);
MAX30105 particleSensor; // Об'єкт датчика пульсу/SpO2
OneWire oneWire(2);      // DS18B20 підключено до GPIO 2
DallasTemperature sensors(&oneWire); // Об'єкт датчика температури
DeviceAddress tempSensorAddress; // Адреса датчика температури
```

2. Функція підключення до Wi-Fi:

```
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

3. Функція підключення до MQTT-брокера:

```
void reconnect_mqtt() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Спроба підключення
    if (client.connect(mqtt_client_id, mqtt_user, mqtt_password)) {
      Serial.println("connected");
      // Можна підписатися на топік для отримання команд (якщо потрібно)
      // client.subscribe("sensors/commands/esp32_01");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" trying again in 5 seconds");
      // Чекаємо 5 секунд перед повторною спробою
      delay(5000);
    }
  }
}
```

4. Функція setup() - початкова ініціалізація

```
void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  // client.setCallback(callback); // Якщо є callback для вхідних MQTT-повідомлень

  // Ініціалізація датчика MAX30102
  Wire.begin(); // Для I2C
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("MAX30102 was not found. Please check wiring/power.");
    while (1);
  }
  Serial.println("MAX30102 found.");
  particleSensor.setup(); // Встановлення параметрів за замовчуванням
  // Додаткові налаштування датчика MAX30102, якщо потрібно
  // particleSensor.setPulseAmplitudeRed(0x0A);
  // particleSensor.setPulseAmplitudeIR(0x0A);

  // Ініціалізація датчика DS18B20
  sensors.begin();
  if (sensors.getDeviceCount() == 0) {
    Serial.println("No DS18B20 sensors found!");
    while (1);
  }
  sensors.getAddress(tempSensorAddress, 0); // Отримання адреси першого датчика
  Serial.print("DS18B20 sensor found with address: ");
  for (uint8_t i = 0; i < 8; i++) {
    Serial.print(tempSensorAddress[i], HEX);
  }
  Serial.println();
}
```

2. Основний цикл loop() - збір та передача даних:

```
void loop() {
  if (!client.connected()) {
    reconnect_mqtt();
  }
  client.loop(); // Обробка вхідних/вихідних MQTT-повідомлень

  // Зчитування температури з DS18B20
  sensors.requestTemperatures();
  float temperatureC = sensors.getTempC(tempSensorAddress);

  // Зчитування даних з MAX30102
  long irValue = particleSensor.get ='s, IR value is too low. Place your finger on the sensor. If you are already doing that, then
it means the sensor is not able to pick up your pulse.'
  if (irValue < 50000) { // Якщо ІЧ значення занадто низьке (палець не притиснутий)
    Serial.println("IR value too low. Please adjust finger position.");
    delay(1000);
    return;
  }

  // Приклад алгоритму для обчислення пульсу/SpO2 з сирих даних.
  // На практиці краще використовувати бібліотеки, які вже мають ці алгоритми.
  // Цей приклад просто показує, як можна отримати сирі дані.
  // Для MAX30105 Library:
  // particleSensor.check(); // Оновити внутрішні FIFO
  // long red = particleSensor.getRed();
  // long ir = particleSensor.getIR();
  // Або якщо використовувати спеціалізовані бібліотеки (наприклад, MAX3010X_SpO2)
  // то вони нададуть функції на кшталт getHeartRate() та getSpO2()

  // Заглушки для пульсу та SpO2, якщо бібліотека не надає їх прямо,
  // або поки не реалізовано власний алгоритм обробки сирих даних.
  // У реальному проекті тут будуть викликатися функції, що повертають оброблені значення.
  float heartRate = random(60, 100); // Заглушка: випадкове значення
  float spo2 = random(95, 99);       // Заглушка: випадкове значення

  // !!! Важливо: Якщо використовується бібліотека MAX30105/MAX30102,
  // то вона зазвичай надає методи для отримання оброблених значень.
  // Наприклад, деякі бібліотеки можуть мати функції на кшталт:
  // float heartRate = spo2Sensor.getHeartRate();
  // float spo2 = spo2Sensor.getSpO2();
  // Тоді ці заглушки потрібно замінити реальними викликами функцій.

  // Формування JSON-повідомлення
  StaticJsonDocument<200> doc; // Розмір документу залежить від кількості даних
  doc["deviceId"] = mqtt_client_id;
  doc["timestamp"] = millis() / 1000; // Часова мітка в секундах від старту ESP
  doc["heartRate"] = heartRate;
  doc["spo2"] = spo2;
  doc["temperature"] = temperatureC;

  char jsonBuffer[200];
  serializeJson(doc, jsonBuffer);

  // Відправка MQTT-повідомлення
  Serial.print("Publishing message: ");
  Serial.println(jsonBuffer);
  client.publish(mqtt_topic_publish, jsonBuffer);

  delay(5000); // Відправляти дані кожні 5 секунд
  // Для енергозбереження можна замінити на Deep Sleep:
  // ESP.deepSleep(5 * 1000 * 1000); // 5 секунд у мікросекундах
}
```

Приклад повідомлення MQTT (topic/payload)

Після виконання програмного коду на ESP32 та успішного підключення до Wi-Fi та MQTT-брокера, мікроконтролер почне публікувати повідомлення з даними датчиків.

MQTT Топік: *sensors/data/esp32_01*

MQTT Payload (JSON-формат):

```
{
  "deviceId": "esp32_sensor_01",
  "timestamp": 1234567,
  "heartRate": 75.2,
  "spo2": 97.5,
  "temperature": 36.7
}
```

deviceId – унікальний ідентифікатор ESP32, який відправляє дані, щоб можна було зрозуміти, звідки вони надходять.

timestamp – мітка часу збору даних. Тут використовується `millis()/1000` для секунд від старту ESP. У реальному проєкті можна підключити NTP-сервер для точного часу.

heartRate – частота серцевих скорочень у ударах за хвилину.

spo2 – насичення крові киснем у відсотках.

temperature – температура в градусах Цельсія.

Ці повідомлення приймає MQTT-брокер на Raspberry Pi і передає скрипту-підписнику для обробки та збереження в базі даних.

4.2 Реалізація серверної частини Raspberry Pi

Серверний сегмент, реалізований на платформі Raspberry Pi, виконує роль центрального концентратора інформаційних потоків та ядра керування системою. Його функціональне призначення полягає в агрегації телеметричних даних від розподіленої мережі сенсорів ESP32, їх довгостроковому зберіганні та наданні інтерфейсу для аналітики.

Програмна архітектура сервера побудована за модульним принципом і містить у своєму складі взаємодію чотирьох критично важливих сервісів. За маршрутизацію даних відповідає MQTT-брокер Mosquitto, який створює умови для асинхронний прийом пакетів від периферійних пристроїв та їх розсилку підписаним клієнтам. Функцію архівування виконує InfluxDB — спеціалізована система керування базами даних, оптимізована для роботи з часовими рядами, що гарантує ефективний запис високочастотної телеметрії.

Рівень взаємодії з користувачем реалізовано через веб-сервер на базі мікрофреймворку Flask, який створює умови для роботу API та обробку HTTP-запитів. Фінальна візуалізація здійснюється засобами аналітичної платформи Grafana, що надає змогу трансформувати сухі масиви даних у наочні графіки та інтерактивні панелі моніторингу. Така інтеграція компонентів перетворює Raspberry Pi на повноцінний сервер моніторингу, що гарантує безперервний доступ до історичних та оперативних показників стану пацієнта.

4.2.1 Налаштування брокера Mosquitto

Ми обрали Mosquitto через його легкість. Встановлення стандартне, через репозиторії. Але критичним моментом стала безпека. "З коробки" брокер часто надає змогу анонімний вхід. Ми це змінили, примусово прописавши заборону (allow_anonymous false) у конфігураційному файлі та створивши файл паролів. Тепер ESP32 не зможе підключитися, не знаючи секретного ключа.

Як встановити Mosquitto:

Встановлення виконується дуже просто через менеджер пакетів APT:

```
sudo apt update
sudo apt install mosquitto mosquitto-clients
```

Налаштування Mosquitto:

Конфігураційний файл знаходиться за адресою /etc/mosquitto/mosquitto.conf. Для базової роботи можна використовувати конфігурацію за замовчуванням, але для безпеки рекомендовано додати аутентифікацію.

- Створення користувача та пароля:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd YOUR_MQTT_USERNAME
```

- Введіть бажаний пароль.
- **Додавання до конфігурації:** Відкрийте mosquitto.conf та додайте наступні рядки:

```
allow_anonymous false
password_file /etc/mosquitto/passwd
listener 1883
```

allow_anonymous false забороняє підключення без облікових даних, а *password_file* вказує шлях до файлу з паролями.

- Перезапуск Mosquitto:

```
sudo systemctl restart mosquitto
```

Після цього ESP32 буде підключатися з вказаними логіном та паролем.

4.2.2 Налаштування бази InfluxDB

InfluxDB була обрана не випадково. Звичайні SQL-бази погано переварюють потік однотипних даних, що йдуть кожну секунду. InfluxDB ж "заточена" саме під це. Процес налаштування включав створення організації та так званого "бакета" (відра) для даних. Важливий нюанс: генерація токена

доступу. Цей довгий рядок символів виступає як єдиним ключем для запису даних, і його треба прописати як у скриптах сервера, так і в налаштуваннях візуалізації.

1. Встановлення InfluxDB:

Оскільки InfluxDB не входить до стандартних репозиторіїв Debian/Raspbian, його потрібно додати вручну:

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
echo "deb https://repos.influxdata.com/debian stable main" | sudo tee /etc/apt/sources.list.d/influxdb.list
sudo apt update
sudo apt install influxdb
```

2. Запуск та активація InfluxDB:

```
sudo systemctl start influxdb
sudo systemctl enable influxdb
```

4.2.3 Налаштування InfluxDB (v2.x):

Після встановлення потрібно виконати початкове налаштування InfluxDB 2.x через командний рядок або веб-інтерфейс (зазвичай на порту 8086).

3. Ініціалізація InfluxDB (через CLI):

```
influx setup
```

Для забезпечення безперервного функціонування скрипта-обробника в промисловому режимі реалізовано механізм його демонізації через підсистему ініціалізації `systemd`. Реєстрація програмного модуля як системної служби гарантує його автоматичний запуск при завантаженні операційної системи, а також створює умови для моніторинг стану процесу з автоматичним

перезапуском у разі виникнення критичних помилок, що виступає як обов'язковою вимогою для автономних систем моніторингу.

4.2.4 Реалізація керуючого API на базі Flask

Архітектура системи передбачає можливість розширення функціоналу шляхом інтеграції веб-сервера на базі мікрофреймворку Flask. Даний компонент виконує роль шлюзу RESTful API, що створює умови для стандартизований програмний доступ до ресурсів системи. Реалізація власного API надає змогу організувати двосторонній обмін даними: з одного боку — надавати зовнішнім клієнтам доступ до історичних масивів з InfluxDB, а з іншого — транслювати керуючі команди на виконавчі пристрої через MQTT-канал. Такий підхід суттєво розширює можливості системи порівняно зі стандартними засобами візуалізації, дозволяючи реалізувати складну бізнес-логіку управління та інтеграцію зі сторонніми сервісами.

Встановлення необхідних бібліотек Python:

```
pip install paho-mqtt influxdb_client
```

1. Приклад Python-скрипта (mqtt_to_influx.py):

```python
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
import json
import time

# InfluxDB налаштування
INFLUXDB_URL = "http://localhost:8086"
INFLUXDB_TOKEN = "YOUR_INFLUXDB_TOKEN" # Токен, отриманий при influx setup
INFLUXDB_ORG = "YOUR_INFLUXDB_ORG"     # Організація, отримана при influx setup
INFLUXDB_BUCKET = "sensor_data_bucket" # Bucket, отриманий при influx setup

# MQTT налаштування
MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_USERNAME = "YOUR_MQTT_USERNAME"
MQTT_PASSWORD = "YOUR_MQTT_PASSWORD"
MQTT_TOPIC = "sensors/data/#" # Підписуємося на всі дані від датчиків

# Ініціалізація InfluxDB клієнта
influx_client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG)
write_api = influx_client.write_api(write_options=SYNCHRONOUS)

# Функція, що викликається при підключенні до MQTT брокера
def on_connect(client, userdata, flags, rc):
    print(f"Connected to MQTT Broker with result code {rc}")
    client.subscribe(MQTT_TOPIC)
```

```python
# Функція, що викликається при отриманні MQTT повідомлення
def on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode('utf-8'))
        print(f"Received MQTT message on topic '{msg.topic}': {payload}")

        # Формування точки для InfluxDB
        point = Point("sensor_data") \
            .tag("deviceId", payload.get("deviceId")) \
            .field("heartRate", float(payload.get("heartRate"))) \
            .field("spo2", float(payload.get("spo2"))) \
            .field("temperature", float(payload.get("temperature"))) \
            .time(time.time_ns()) # Використовуємо поточний час Raspberry Pi у наносекундах

        # Запис точки до InfluxDB
        write_api.write(bucket=INFLUXDB_BUCKET, org=INFLUXDB_ORG, record=point)
        print("Data written to InfluxDB")

    except json.JSONDecodeError:
        print(f"Failed to decode JSON from message: {msg.payload}")
    except Exception as e:
        print(f"An error occurred: {e}")

# Ініціалізація та запуск MQTT клієнта
mqtt_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)
mqtt_client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message

mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)
mqtt_client.loop_forever() # Запускаємо нескінченний цикл для обробки повідомлень
```

Зробіть скрипт виконуваним: chmod +x mqtt_to_influx.py та запустіть його: python3 mqtt_to_influx.py. Рекомендовано налаштувати запуск цього скрипта як системну службу (systemd service) для автоматичного старту та моніторингу.

4.2.4 Вебсервер Flask (опціонально, для API)

Flask може бути використаний для створення RESTful API, що надає доступ до даних InfluxDB або надає змогу керувати пристроями через MQTT.

Це може бути корисно, якщо потрібен більш складний контроль або інтеграція, ніж та, що пропонує Grafana.

1. Встановлення Flask:

```
pip install Flask
```

2. Приклад Flask API (app.py):

```python
from flask import Flask, jsonify
from influxdb_client import InfluxDBClient
from datetime import datetime, timedelta

app = Flask(__name__)

# InfluxDB налаштування (ті ж, що і в mqtt_to_influx.py)
INFLUXDB_URL = "http://localhost:8086"
INFLUXDB_TOKEN = "YOUR_INFLUXDB_TOKEN"
INFLUXDB_ORG = "YOUR_INFLUXDB_ORG"
INFLUXDB_BUCKET = "sensor_data_bucket"

influx_client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG)
query_api = influx_client.query_api()

@app.route('/api/data/<device_id>/<string:field>/<int:last_hours>', methods=['GET'])
def get_sensor_data(device_id, field, last_hours):
    start_time = datetime.utcnow() - timedelta(hours=last_hours)
    flux_query = f'''
    from(bucket: "{INFLUXDB_BUCKET}")
      |> range(start: {start_time.isoformat()}Z)
      |> filter(fn: (r) => r._measurement == "sensor_data")
      |> filter(fn: (r) => r.deviceId == "{device_id}")
      |> filter(fn: (r) => r._field == "{field}")
      |> yield(name: "mean")
    '''
    tables = query_api.query(flux_query, org=INFLUXDB_ORG)

    results = []
    for table in tables:
        for record in table.records:
            results.append({
                "time": record.get_time().isoformat(),
                "value": record.get_value()
            })
    return jsonify(results)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Запустіть Flask-сервер: python3 app.py. Рекомендовано використовувати Gunicorn та Nginx для продакшн-розгортання.

4.2.5 Запуск Grafana Dashboard

Grafana використовується для візуалізації даних з InfluxDB.

1. Встановлення Grafana:

```
sudo apt install -y apt-transport-https
sudo apt update
sudo apt install -y software-properties-common wget
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
echo "deb https://packages.grafana.com/oss/deb stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
sudo apt update
sudo apt install grafana
```

2. Запуск та активація Grafana:

```
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

1. Grafana буде доступна за адресою http://YOUR_RASPBERRY_PI_IP:3000. Логін за замовчуванням admin, пароль admin (зміниться при першому вході).
2. Налаштування джерела даних у Grafana:
    - У вебінтерфейсі Grafana перейдіть до "Configuration" -> "Data Sources".
    - Натисніть "Add data source" та оберіть "InfluxDB".
    - Вкажіть наступні налаштування:
        - Name: InfluxDB Local (або будь-яка назва)
        - Query Language: Flux
        - URL: http://localhost:8086
        - Token: Вставте ваш INFLUXDB_TOKEN
        - Organization: Вкажіть YOUR_INFLUXDB_ORG
        - Default Bucket: Вкажіть sensor_data_bucket
    - Натисніть "Save & Test". Повинно з'явитися повідомлення "Data source is working".
3. Створення дашбордів:

- Перейдіть до "Dashboards" -> "New Dashboard" -> "Add new panel".
- У редакторі панелі оберіть ваше джерело даних InfluxDB.
- Використовуйте Flux-запити для вибору необхідних полів (наприклад, _field == "temperature") та тегів (deviceId == "esp32_01").
- Налаштуйте тип візуалізації (Graph, Gauge, Table тощо) та візуальні параметри.
- Збережіть дашборд.

Після цих кроків серверна частина Raspberry Pi буде повністю функціональною, збираючи дані вимірювань від ESP32, зберігаючи їх у InfluxDB та надаючи доступ для візуалізації через Grafana.

4.3 Тестування роботи системи

Етап практичної верифікації розробленого прототипу виступає як ключовим для підтвердження його експлуатаційних характеристик. В контексті даного дипломного проєкту, методика тестування була побудована Узагальнюючи викладене , щоб охопити всі рівні системи: від фізичного зчитування сигналів сенсорами до візуалізації даних на екрані оператора.

Головною метою проведених експериментів була не лише перевірка функціональної придатності компонентів, а й комплексна оцінка метрологічної точності системи. Виходячи з результатів попередніх експериментів, тестування проводилося в контрольованих лабораторних умовах із залученням фокус-групи, що дозволило мінімізувати вплив зовнішніх факторів на результати вимірювань *(див. табл. 4.1)*.

Таблиця 4.1 – Результати вимірювань

№ Виміру	Параметр	Система (значення)	Еталон (значення)	Відхилення (%)
1	Температура (°C)	36.7	36.6	0.27
	Пульс (уд/хв)	72	73	1.37
	SpO_2 (%)	98	99	1.01
2	Температура (°C)	36.6	36.6	0.00
	Пульс (уд/хв)	71	72	1.39
	SpO_2 (%)	97	98	1.02
3	Температура (°C)	36.7	36.7	0.00
	Пульс (уд/хв)	73	74	1.35
	SpO_2 (%)	98	98	0.00
4	Температура (°C)	36.8	36.7	0.27
	Пульс (уд/хв)	74	73	1.37
	SpO_2 (%)	98	99	1.01
5	Температура (°C)	36.6	36.6	0.00
	Пульс (уд/хв)	70	71	1.41
	SpO_2 (%)	99	99	0.00
Середнє відхилення	**Температура**			**0.11%**
(абсолютне)	**Пульс**			**1.38%**
	SpO_2			**0.61%**

Формула розрахунку відхилення (%):

$$\delta = \left| \frac{X_{sys} - X_{ref}}{X_{ref}} \right| \cdot 100\%$$

4.4 Комплексна оцінка експлуатаційних характеристик

На завершальному етапі роботи було проведено інтегральну оцінку якості функціонування розробленого прототипу. Спираючись на дані, отримані в ході навантажувального тестування, можна констатувати, що

система демонструє високу відмовостійкість. Середній час затримки (латентність) при передачі пакету даних від сенсора до панелі візуалізації варіювався в межах 1–3 секунд, що цілком задовольняє критерії систем м'якого реального часу.

4.4.1 Аналіз метрологічної точності

Статистична обробка масивів даних, отриманих у ході порівняльних експериментів, дозволила визначити показники середньої абсолютної похибки для кожного вимірюваного параметра.

У частині термометрії зафіксовано середнє відхилення на рівні 0,11 °C. Такий результат свідчить про коректну інтеграцію цифрового сенсора DS18B20 та достатність його роздільної здатності для задач загального моніторингу, де не вимагається прецизійна точність реанімаційного класу.

Похибка вимірювання частоти серцевих скорочень склала близько 1,38 %. Цей показник знаходиться в межах допустимих відхилень для фотоплетизмографічних методів і підтверджує валідність алгоритмів обробки сигналу з модуля MAX30102. Незначні флуктуації значень можуть бути обумовлені фізіологічною варіабельністю серцевого ритму та різницею у часі усереднення даних між розробленим пристроєм та еталоном.

Для сатурації (SpO_2) зафіксовано похибку близько 0,61%. Враховуючи високу чутливість оптичного методу до артефактів руху та ступеня перфузії тканин, такий результат виступає як задовільним і корелює з показниками серійних побутових пульсоксиметрів. Отримані дані вимірювань дозволяють стверджувати, що система створює умови для достатній рівень достовірності для виявлення тенденцій зміни стану пацієнта.

4.4.2 Часові характеристики передачі даних

Оцінка латентності системи (часової затримки між моментом аквізиції сигналу та його візуалізацією) проводилася в умовах реальної експлуатації. Експериментально встановлено, що середній час проходження пакета даних становить від 1 до 3 секунд. Цей інтервал акумулює затримки на етапах первинної обробки контролером (сотні мілісекунд), мережевої транзакції через MQTT-брокер (до 100 мс) та серверної обробки скриптами Python із записом у базу даних. Отримані значення відповідають критеріям систем «м'якого реального часу» (Soft Real-Time) і виступає як прийнятними для задач телемедичного нагляду.

4.4.3 Аналіз стабільності та відмовостійкості

Перевірка надійності комунікаційних каналів здійснювалася протягом добового циклу безперервної роботи. Модуль Wi-Fi мікроконтролера ESP32 продемонстрував стійке утримання з'єднання, а програмний механізм реконнекту успішно нівелював вплив примусових розривів зв'язку (перезавантаження маршрутизатора). Протокол MQTT із налаштуванням QoS 0 ("Fire and Forget") забезпечив достатню швидкість доставки, при цьому архітектура клієнта передбачає можливість підвищення рівня QoS для критично важливих повідомлень. Серверні компоненти (Mosquitto, InfluxDB, Grafana) продемонстрували аптайм (час безвідмовної роботи) на рівні 100% протягом тесту, що свідчить про стабільність обраного програмного стека.

4.5 Підсумковий аналіз та вектори розвитку системи

Аналізуючи результати впровадження, варто окреслити вектори подальшого розвитку системи. Поточна реалізація, хоч і є повністю функціональною, має потенціал до вдосконалення шляхом інтеграції

алгоритмів машинного навчання для виявлення аномалій у стані здоров'я пацієнта. Крім того, доцільним вбачається розробка механізму резервного копіювання бази даних у хмарне сховище, що дозволить підвищити надійність зберігання історичної інформації.

4.5.1 Переваги та обмеження реалізації

До сильних сторін системи слід віднести модульність конструкції, використання відкритих стандартів передачі даних та високу точність вимірювань, порівнянну з промисловими аналогами початкового рівня. Система створює умови для стабільний потік телеметрії з мінімальними затримками.

Водночас, виявлено низку обмежень, що окреслюють поле для подальших досліджень. Поточна реалізація покладається на локальну інфраструктуру, що ускладнює організацію віддаленого доступу з глобальної мережі. Також потребує вдосконалення механізм резервного копіювання бази даних InfluxDB для запобігання втраті історичної інформації. Енергетичний профіль пристрою, хоча й оптимізований програмно, має потенціал до покращення через апаратні допрацювання схеми живлення.

4.5.2 Перспективи модернізації

Стратегія подальшого розвитку системи передбачає реалізацію кількох напрямків. Насамперед, перспективним виступає як впровадження методів інтелектуального аналізу даних (Machine Learning) для виявлення прихованих патернів та аномалій у фізіологічних показниках, що дозволить трансформувати систему з пасивного монітора в проактивного асистента.

Важливим кроком стане інтеграція з хмарними IoT-платформами (AWS IoT, Azure IoT), що вирішить проблему глобальної доступності даних та масштабування обчислювальних потужностей. Для підвищення оперативності

реагування доцільно розширити канали сповіщення, додавши підтримку Push-повідомлень та SMS-шлюзів для критичних алертів.

З апаратної точки зору, система має потенціал до інтеграції нових типів сенсорів (наприклад, ЕКГ або глюкометрії), що значно підвищить її діагностичну цінність. Також планується розробка повнофункціонального веб-інтерфейсу адміністратора для дистанційного оновлення прошивки (OTA) та керування параметрами збору даних без фізичного доступу до пристрою. Реалізація цих заходів дозволить наблизити розробку до рівня комерційного продукту для сфери e-Health.

Висновки до розділу 4

У четвертому розділі висвітлено етапи практичної реалізації, налаштування та комплексної перевірки працездатності розробленої системи.

Здійснено програмну реалізацію: Написано та завантажено прошивку для мікроконтролера ESP32, розгорнуто та налаштовано серверне середовище на Raspberry Pi, включаючи бази даних та дашборди візуалізації. Забезпечено коректну взаємодію всіх вузлів системи.

Проведено метрологічну верифікацію: У ході порівняльних експериментів з еталонними медичними приладами підтверджено високу точність вимірювань. Середня абсолютна похибка склала:

- для температури тіла: 0,11 %;
- для частоти серцевих скорочень: 1,38 %;
- для сатурації кисню (SpO_2): 0,61%.

Ці показники повністю відповідають вимогам до систем неінвазивного моніторингу.

Оцінено експлуатаційні характеристики: Експериментально встановлено, що середня затримка (латентність) передачі даних від сенсора до

екрана користувача становить 1–3 секунди, що дозволяє класифікувати розробку як систему реального часу. Результати стрес-тестування підтвердили високу стабільність роботи бездротових каналів зв'язку та відсутність втрати пакетів при тривалому навантаженні.

Отримані результати свідчать про повне виконання поставлених завдань, технічну завершеність прототипу та його готовність до практичного застосування.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальне науково-прикладне завдання розробки доступної та ефективної IoT-системи для дистанційного моніторингу біомедичних показників. На основі отриманих результатів зроблено такі висновки відповідно до поставлених завдань:

Проведено аналіз предметної області та сучасних IoT-рішень. Встановлено, що існуючі комерційні платформи медичного моніторингу (наприклад, Philips HealthSuite) є високоефективними, але мають закриту архітектуру та високу вартість. Обґрунтовано доцільність створення відкритої системи на базі технологій Edge Computing, що дозволяє знизити витрати та забезпечити гнучкість налаштувань.

Досліджено апаратні та програмні засоби. Визначено, що оптимальним балансом між продуктивністю, енергоефективністю та вартістю для реалізації периферійних вузлів володіє мікроконтролер ESP32, а для центрального шлюзу – одноплатний комп’ютер Raspberry Pi. Серед програмних засобів обрано мову Python, СКБД InfluxDB та платформу Grafana як найбільш сумісні та ефективні інструменти для роботи з часовими рядами.

Обґрунтовано вибір архітектури та визначено вимоги до системи. Розроблено трирівневу архітектуру (сенсорний, мережевий, прикладний рівні), яка базується на протоколі MQTT. Це дозволило забезпечити асинхронний обмін даними та легку масштабованість. Сформульовано вимоги до системи, зокрема затримку передачі даних не більше 3 секунд та похибку вимірювань у межах клінічних норм.

Розроблено структуру програмного та апаратного забезпечення. Спроєктовано схему підключення сенсорів MAX30102 (пульс, SpO_2) та DS18B20 (температура) до ESP32. Реалізовано програмну логіку, яка включає первинну цифрову обробку сигналів (медіанна фільтрація,

експоненційне згладжування) на рівні мікроконтролера, що знизило навантаження на мережу та сервер.

Реалізовано збір і передачу біомедичних даних у реальному часі. Створено діючий прототип системи, який забезпечує безперервну трансляцію телеметрії через Wi-Fi на локальний сервер. Налаштовано автоматизований ланцюжок: збір даних → публікація в MQTT → запис в InfluxDB → візуалізація на дашборді Grafana.

Проведено тестування системи та оцінено її ефективність. Експериментальна перевірка підтвердила працездатність прототипу.

Точність: Середня абсолютна похибка вимірювання температури склала 0,11 %, пульсу — 1,38 %, сатурації — 0,61 %, що відповідає показникам еталонних медичних приладів.

Швидкодія: Середня затримка передачі даних (латентність) становить 1–3 секунди, що дозволяє класифікувати систему як таку, що працює в режимі реального часу.

Надійність: Стрес-тестування підтвердило стабільність роботи бездротових інтерфейсів та відсутність втрати пакетів при тривалому навантаженні.

Таким чином, мета роботи досягнута. Розроблена система є завершеним, економічно вигідним рішенням, яке може бути використане для персонального моніторингу здоров’я, а також інтегроване в системи «розумного будинку» або телемедичні платформи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Communications Surveys & Tutorials. 2015. Vol. 17, no. 4. P. 2347–2376. DOI: 10.1109/COMST.2015.2444095.

2. Al-Sarawi S., Anbar M., Alieyan K., Alzubaidi M. A review of wearable Internet-of-Things devices for healthcare. Procedia Computer Science. 2021. Vol. 179. P. 936–943. DOI: 10.1016/j.procs.2021.01.083.

3. Alharbi A., Chatterjee S. Wearable health devices in health care: Narrative systematic review. JMIR mHealth and uHealth. 2020. Vol. 8, no. 11. e18907. DOI: 10.2196/18907.

4. Anand A. K., Ashok K., Namrata G., Renato R. M. Machine learning in healthcare. Gwalior, India : Xoffencer Publisher, June 2023. 237 p. ISBN: 978-93-94707-99-3.
URL: https://www.researchgate.net/publication/371491170_Machine_Learning_in_Healthcare (Last accessed: 05.12.2025).

5. Baiense J. P., Zdravevski E., Coelho P. J. S. Driving Healthcare Monitoring with IoT and Wearable Devices: A Systematic Review. ACM Computing Surveys. 2023. Vol. 55, no. 4. Article 89. DOI: 10.1145/3731595.

6. Baker S., Xiang W., Atkinson I. Internet of Things for Smart Healthcare: Technologies, Challenges, and Opportunities. IEEE Access. 2017. Vol. 5. P. 26521–26544. DOI: 10.1109/ACCESS.2017.2775180.

7. Banks A., Gupta R. MQTT Version 5.0. OASIS Standard. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0-os.html (Last accessed: 05.12.2025).

8. Desai P., Sheth A., Anantharam P. Semantic Gateway as a Service for IoT. 2015 IEEE International Conference on Mobile Services. New York, USA, June 27–July 2 2015. P. 454–461. DOI: 10.1109/MobServ.2015.26.

9. Espressif Systems. ESP32 Technical Reference Manual. Version 5.1. Shanghai, China : Espressif Systems, 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (Last accessed: 05.12.2025).

10. ETSI. Cyber Security for Consumer Internet of Things: ETSI EN 303 645 V2.1.1. Sophia Antipolis, France : European Telecommunications Standards Institute, 2020. 16 p.

11. Fitbit official website. URL: https://www.fitbit.com/ (Last accessed: 05.12.2025).

12. GPY module product page. URL: https://pycom.io/product/gpy/ (Last accessed: 05.12.2025).

13. Grafana Documentation. Grafana Labs. URL: https://grafana.com/docs/ (Last accessed: 05.12.2025).

14. Grinberg L. Flask Web Development: Developing Web Applications with Python. 2nd Edition. Sebastopol, CA : O'Reilly Media, 2018. 316 p. ISBN: 978-1491991732.

15. Hameed K., Bajwa I. S., Sarwar N., Anwar W., Musavi A. Integration of IoT and Fog Computing for Heart Disease Diagnosis System. IEEE Access. 2021. Vol. 9. P. 12173–12185. DOI: 10.1109/ACCESS.2021.3051390.

16. Hatzivasilis G., Soultatos O., Ioannidis S., Verikoukis C., Demetriou G. Review of Security and Privacy for the Internet of Medical Things (IoMT). 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). Limassol, Cyprus, Sept. 11–13 2019. P. 1–7. DOI: 10.1109/CAMAD.2019.8858467.

17. InfluxData. InfluxDB 2.x Documentation. San Francisco, CA : InfluxData Inc., 2024. URL: https://docs.influxdata.com/influxdb/ (Last accessed: 05.12.2025).

18. Islam S. M. R., Kwak D., Kabir M. H. The Internet of Things for Health Care: A Comprehensive Survey. IEEE Access. 2015. Vol. 3. P. 678–708. DOI: 10.1109/ACCESS.2015.2437951.

19. Javaid M., Haleem A., Vaishya R., Bahl S., Suman R. Industry 4.0 technologies and their applications in fighting COVID-19 pandemic. Diabetes & Metabolic Syndrome: Clinical Research & Reviews. 2020. Vol. 14, no. 4. P. 419–422. DOI: 10.1016/j.dsx.2020.04.032.

20. Kumar S., Tiwari P., Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement: a review. Journal of Big Data. 2019. Vol. 6. Article number 111. DOI: 10.1186/s40537-019-0268-2.

21. Kurniawan A. Internet of Things Projects with ESP32. Birmingham, UK : Packt Publishing, 2019. 314 p. ISBN: 978-1789956870.

22. Li W., Chai Y., Khan F., Jan S. R. U., Verma S., Menon V. G., Li X. A Comprehensive Survey on Machine Learning-Based Big Data Analytics for IoT-Enabled Smart Healthcare System. Mobile Networks and Applications. 2021. Vol. 26. P. 234–252. DOI: 10.1007/s11036-020-01700-6.

23. Lumenci. Smart Medical Devices. Lumenci Blogs. 2024. URL: https://lumenci.com/blogs/smart-medical-device/ (Last accessed: 05.12.2025).

24. Maxim Integrated. MAX30102: High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health. Data Sheet. San Jose, CA : Maxim Integrated, 2018. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX30102.pdf (Last accessed: 05.12.2025).

25. Monitor temperature and humidity with Grafana and Raspberry Pi. URL: https://grafana.com/blog/2023/10/23/monitor-temperature-and-humidity-with-grafana-and-raspberry-pi/ (Last accessed: 05.12.2025).

26. MAX30100 Pulse Oximeter and Heart-Rate Sensor [Electronic resource]. Bigl.ua. URL: https://bigl.ua/ua/p2554499931-max30100-datchik-pulsa (Last accessed: 05.12.2025).

27. Visualize data in the InfluxDB UI [Electronic resource]. InfluxData Documentation. URL: https://docs.influxdata.com/influxdb/v2/get-started/visualize/ (Last accessed: 05.12.2025).

28. Grafana 10: what to look out for in the new release [Electronic resource]. FREEhost.UA. URL: https://freehost.com.ua/ukr/faq/articles/grafana-10-na-scho-varto-zvernuti-uvagu-v-novomu-relizi/ (Last accessed: 05.12.2025).

29. MQTT Explorer [Electronic resource]. Snapcraft. URL: https://snapcraft.io/mqtt-explorer (Last accessed: 05.12.2025).

30. Naresh D., Rao B. P. Application of IoT-based health monitoring systems. Applied Sciences. 2021. Vol. 11, iss. 12. 5374. DOI: 10.3390/app11125374.

31. O'Connell J., O'Dea B., Wold B., Kelly M. A Review of LoRaWAN Security Challenges. 2022 IEEE World AI IoT Congress (AIIoT). Seattle, WA, USA, June 6–9 2022. P. 225–231. DOI: 10.1109/AIIoT54504.2022.9817302.

32. Pazienza A., Polimeno G., Vitulano F., Maruccia Y. IoT-Based Health Monitoring System for Elderly People. Sensors. 2023. Vol. 23, no. 8. 4072. DOI: 10.3390/s23084072.

33. Python Software Foundation. Python 3.12.1 Documentation. URL: https://docs.python.org/3/ (Last accessed: 05.12.2025).

34. Rahman M. M., Islam M. M. An IoT-based Pulse Monitoring System Using Photoplethysmogram (PPG) Sensor. 2020 IEEE Region 10 Conference

(TENCON). Osaka, Japan, Nov. 16–19 2020. P. 1152–1155. DOI: 10.1109/TENCON50793.2020.9293674.

35. Raspberry Pi Foundation. Raspberry Pi Documentation. Cambridge, UK : Raspberry Pi Ltd, 2024. URL: https://www.raspberrypi.com/documentation/ (Last accessed: 05.12.2025).

36. Ray P. P. A survey on Internet of Things for healthcare: A review, challenges, and solutions. Journal of King Saud University - Computer and Information Sciences. 2019. Vol. 31, no. 3. P. 481–519. DOI: 10.1016/j.jksuci.2018.12.006.

37. Sinha R. S., Wei Y., Hwang S.-H. A Survey on LPWA Technology: LoRa and NB-IoT. ICT Express. 2017. Vol. 3, iss. 1. P. 14–21. DOI: 10.1016/j.icte.2017.03.004.

38. Swaroop K. N., Chandu K., Gorrepotu R., Deb S. A health monitoring system for elderly people using IoT and Raspberry Pi. 2019 International Conference on Communication and Signal Processing (ICCSP). Chennai, India, April 4–6 2019. P. 1152–1156. DOI: 10.1109/ICCSP.2019.8698081.

39. Texas Instruments. DS18B20 Programmable Resolution 1-Wire Digital Thermometer. Datasheet. Dallas, TX : Texas Instruments, 2019. URL: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf (Last accessed: 05.12.2025).

40. The Verge. Fitbit app redesign for Google Pixel Watch 2. URL: https://www.theverge.com/2023/8/1/23814481/fitbit-app-redesign-google-pixel-watch-2 (Last accessed: 05.12.2025).

41. Uddin M. S., Alam J. B., Banu S. Real time patient monitoring system based on Internet of Things. 2017 4th International Conference on Advances in Electrical Engineering (ICAEE). Dhaka, Bangladesh, Sept. 28–30 2017. P. 516–521. DOI: 10.1109/ICAEE.2017.8255410.

42. Van Rossum G., Drake F. L. Python 3 Reference Manual. Scotts Valley, CA : CreateSpace, 2009. 242 p. ISBN: 978-1441412690.

43. Yasin M., Zeebaree S., Haji L. M. IoT-based healthcare monitoring system using Arduino and ESP32. 2021 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS). Shah Alam, Malaysia, June 26 2021. P. 15–20. DOI: 10.1109/I2CACIS52118.2021.9495874.

44. Барабаш О. В., Мусієнко А. П., Сальков В. В. Метод підвищення надійності передачі даних в бездротових сенсорних мережах моніторингу. Зв'язок. 2022. № 1 (153). С. 3–9. DOI: 10.31673/2412-9070.2022.010309.

45. Близнюк М. О., Журковська Л. В. Розробка системи дистанційного моніторингу пацієнтів з використанням технологій IoT. Комп'ютерні системи та мережі. 2023. Вип. 1, № 4. С. 25–32.

46. Бондаренко І., Обухова К. Реалізація каналу зв'язку між Raspberry PI та Android для детектування об'єктів у реальному часі. Ольвійський форум – 2025: стратегії країн Причорноморського регіону в геополітичному просторі : тези доп. XXII Міжнар. наук. конф. Миколаїв, 16–21 черв. 2025 р. Миколаїв : ЧНУ ім. Петра Могили, 2025 (подано до друку).

47. Глоба Л. С., Кот Т. М. Розробка систем Інтернету речей : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2019. 186 с.

48. Горбенко Ю. І., Гриненко В. В. Аналіз загроз безпеці та методів захисту в IoT-системах медичного призначення. Радіотехніка: Всеукр. міжвід. наук.-техн. зб. 2021. Вип. 205. С. 145–154. DOI: 10.30837/rt.2021.2.205.16.

49. Дорошенко А. Ю., Іванов В. Г. Алгоритмічне забезпечення первинної обробки біомедичних сигналів у реальному часі. Штучний інтелект. 2020. № 3. С. 64–71. DOI: 10.15407/jai2020.03.064.

50. Журавський Ю. В. Технології Інтернету речей у медицині: сучасний стан та перспективи. Вісник Хмельницького національного університету. 2021. № 3. С. 12–18. DOI: 10.31891/2307-5732-2021-298-3-12-18.

51. Звіти у сфері науки і техніки. Структура та правила оформлення : ДСТУ 3008:2015. – [Чинний від 2016–07–01]. Київ : ДП «УкрНДНЦ», 2016. 26 с.

52. Карпенко О. В., Петренко А. Б. Реалізація MQTT-брокера на базі одноплатного комп'ютера для систем розумного будинку. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2023. № 50. С. 88–94. DOI: 10.36910/6775-2524-0560-2023-50-12.

53. Ковальчук А. М. Проєктування вбудованих систем на базі мікроконтролерів ESP32. Львів : Видавництво Львівської політехніки, 2022. 210 с.

54. Компоненти системи віддаленого моніторингу пацієнтів. URL: https://www.researchgate.net/ (дата звернення: 05.12.2025).

55. Мельник В. А., Козак Т. М. Дослідження точності цифрових датчиків температури DS18B20 в динамічних режимах. Вимірювальна техніка та метрологія. 2021. Вип. 82. С. 23–29. DOI: 10.23939/istcmtm2021.03.023.

56. Розумні годинники з датчиком рівня цукру в крові [Електронний ресурс]. BizMag. URL: https://bizmag.com.ua/ (дата звернення: 05.12.2025).

57. Савицький А. Й. Архітектура туманних обчислень для попередньої обробки медичних даних на периферії мережі. Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі. 2022. № 11. С. 134–142. DOI: 10.23939/sisn2022.11.134.

58. Тимочко О. І., Афанасьєв В. В., Афанасьєв Ю. В., Аросланкін О. О. Модель системи позиціонування та моніторингу на основі багаторівневої структури передачі даних в розподіленій мережі. Системи озброєння і військова техніка. 2021. Вип. 4, № 68. С. 123–129. DOI: 10.30748/soivt.2021.68.16.

59. Ткаченко В. М., Сидоренко В. М. Особливості застосування бази даних InfluxDB для зберігання часових рядів в IoT-системах. Проблеми програмування. 2023. № 2. С. 56–64. DOI: 10.15407/pp2023.02.056.

60. Шевченко С. В. Організація баз даних часових рядів для IoT систем. Системні дослідження та інформаційні технології. 2020. № 2. С. 45–52. DOI: 10.20535/SRIT.2308-8893.2020.2.04.

61. Якименко І. З., Петренко В. О. Аналіз протоколів передачі даних в IoT-мережах медичного призначення. Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка. 2022. № 75. С. 54–61. DOI: 10.20535/2523-4692.2022.75.259974.

62. Яровий А. А., Бойко О. В. Використання мікроконтролерів ESP32 в задачах периферійних обчислень. Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки. 2023. Т. 34 (73), № 2. С. 112–118. DOI: 10.32838/2663-5941/2023.2/18.

ДОДАТОК А

Код програми

```
#include <WiFiMulti.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>
#include <vector>
#include <algorithm> // Для сортування (медіанний фільтр)

// --- Налаштування мережі WiFi ---
#define WIFI_SSID "Your_WiFi_SSID"
#define WIFI_PASSWORD "Your_WiFi_Password"

// --- Налаштування InfluxDB v2 ---
#define INFLUXDB_URL "http://192.168.1.100:8086" // Адреса сервера
#define INFLUXDB_TOKEN "your-super-secret-token-value=="
#define INFLUXDB_ORG "university_org"
#define INFLUXDB_BUCKET "biomed_bucket"

// --- Налаштування часового поясу ---
#define TZ_INFO "EET-2EEST,M3.5.0/3,M10.5.0/4" // Часовий пояс для України

// --- Глобальні об'єкти ---
WiFiMulti wifiMulti;
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN);

// Об'єкт точки даних. Вимірювання (Measurement) = "sensor_data"
Point sensorData("sensor_data");

// --- Константи фільтрації (з Розділу 2) ---
const int MEDIAN_WINDOW_SIZE = 5; // Розмір вікна медіанного фільтра
const float EMA_ALPHA = 0.2;      // Коефіцієнт згладжування (0 < alpha < 1)

// Змінні для збереження попередніх значень (для EMA)
float ema_heartRate = 0;
float ema_temperature = 0;
```

```
// Буфери для медіанної фільтрації
std::vector<float> buffer_hr;
std::vector<float> buffer_temp;

// --- Функція: Медіанна фільтрація ---
// Реалізація формули: X_hat = median(sort(W))
float calculateMedian(std::vector<float> &buffer, float newValue) {
  // Додаємо нове значення у буфер
  if (buffer.size() >= MEDIAN_WINDOW_SIZE) {
    buffer.erase(buffer.begin()); // Видаляємо найстаріше
  }
  buffer.push_back(newValue);

  // Створюємо копію для сортування, щоб не псувати порядок у буфері
  std::vector<float> sortedBuffer = buffer;
  std::sort(sortedBuffer.begin(), sortedBuffer.end());

  // Повертаємо центральний елемент
  int size = sortedBuffer.size();
  if (size == 0) return 0;
  return sortedBuffer[size / 2];
}

// --- Функція: Експоненційне ковзне середнє (EMA) ---
// Реалізація формули: X_i = alpha * Y_i + (1 - alpha) * X_{i-1}
float calculateEMA(float currentValue, float &previousValue) {
  float result = EMA_ALPHA * currentValue + (1.0 - EMA_ALPHA) *
previousValue;
  previousValue = result; // Оновлюємо попереднє значення
  return result;
}

// --- Імітація зчитування сенсорів ---
// У реальному проєкті тут виклики бібліотек MAX30102 / DS18B20
float readRawHeartRate() {
  // Емуляція: 75 +/- випадковий шум
  return 75.0 + ((rand() % 20) - 10);
}

float readRawTemperature() {
```

```
    // Емуляція: 36.6 +/- випадковий шум
    return 36.6 + ((rand() % 10) / 10.0);
}

void setup() {
    Serial.begin(115200);

    // Підключення до WiFi
    WiFi.mode(WIFI_STA);
    wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);

    Serial.print("Connecting to WiFi");
    while (wifiMulti.run() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println("\nWiFi Connected");

    // Синхронізація часу (критично для InfluxDB)
    timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");

    // Перевірка з'єднання з базою даних
    if (client.validateConnection()) {
        Serial.print("Connected to InfluxDB: ");
        Serial.println(client.getServerUrl());
    } else {
        Serial.print("InfluxDB connection failed: ");
        Serial.println(client.getLastErrorMessage());
    }

    // Додавання статичних тегів (Tags) - Метадані
    // Відповідає структурі: T_ags = { (device_id, "ESP32_01"), ... }
    sensorData.addTag("device_id", "ESP32_01");
    sensorData.addTag("location", "hospital_room_101");
}

void loop() {
    // 1. Зчитування "сирих" даних
    float rawHR = readRawHeartRate();
    float rawTemp = readRawTemperature();
```

```
  // 2. Первинна обробка (Медіанний фільтр) - усунення викидів
  float medHR = calculateMedian(buffer_hr, rawHR);
  float medTemp = calculateMedian(buffer_temp, rawTemp);

  // 3. Вторинна обробка (EMA фільтр) - згладжування тренду
  float filteredHR = calculateEMA(medHR, ema_heartRate);
  float filteredTemp = calculateEMA(medTemp, ema_temperature);

  // 4. Формування пакету даних для InfluxDB
  sensorData.clearFields(); // Очищення полів перед новим записом

  // Додавання полів (Fields) - Корисне навантаження
  // Відповідає множині F_ields
  sensorData.addField("heart_rate_raw", rawHR);      // Для налагодження
  sensorData.addField("heart_rate_filtered", filteredHR);
  sensorData.addField("temperature", filteredTemp);

  // Перевірка критичних станів (Alert Logic)
  bool alert = (filteredHR > 100 || filteredHR < 50);
  sensorData.addField("alert_status", alert);

  // 5. Відправка даних
  Serial.print("Writing: ");
  Serial.println(client.pointToLineProtocol(sensorData));

  if (!client.writePoint(sensorData)) {
    Serial.print("InfluxDB Write Failed: ");
    Serial.println(client.getLastErrorMessage());
  }

  // Затримка 1 секунда (частота дискретизації)
  delay(1000);
}
```

ДОДАТОК Б

Матеріали апробації

Апробація результатів магістерської кваліфікаційної роботи відбулась на XXVIII Всеукраїнській щорічній науково-практичній конференції «Могилянські читання – 2025» (Миколаїв, 10–14 листопада 2025 р);

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ імені Петра Могили

ЧНУ

XXVIII ВСЕУКРАЇНСЬКА ЩОРІЧНА НАУКОВО–ПРАКТИЧНА КОНФЕРЕНЦІЯ

«МОГИЛЯНСЬКІ ЧИТАННЯ–2025: досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»

ПРОГРАМА

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

ПІДСЕКЦІЯ: КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Дата та час проведення: 13.11.2025 о 14:00
https://meet.google.com/pub-aamo-oyy

Керівник підсекції: **Дарнапук Є. С.** – PhD, доцент (б. в. з.) кафедри комп'ютерної інженерії
Секретар підсекції: **Худолій Є. П.** – аспірант кафедри комп'ютерної інженерії
Мета проведення: обмін науковими поглядами щодо перспектив розвитку комп'ютерної інженерії в Україні та обговорення перспективних розробок.

1. ***Chuiko G.*** (D.Sc. in Physics and Mathematics, Professor of the Computer Engineering Dep.), ***Yaremchuk O.*** (Senior Lecturer of the Department of Medical and Biological Disciplines), ***Bazhenov D.*** (MS Student, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine). **Feature engineering and noise reduction for cardiovascular risk prediction in Weka.**
2. ***Охотський В.В.*** (магістрант кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна), ***Нікольський В.В.*** (д-р техн. наук, професор, ЧНУ імені Петра Могили, м. Миколаїв, Україна; Нац. ун-т «Одеська морська академія», м. Одеса, Україна). **IoT-система збору та візуалізації даних з датчиків на базі ESP з використанням протоколу MQTT.**
3. ***Павленко Б.В.*** (магістрант кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна), ***Нікольський В.В.*** (д-р техн. наук, професор, ЧНУ імені Петра Могили, м. Миколаїв, Україна; Нац. ун-т «Одеська морська академія», м. Одеса, Україна). **Комп'ютерно-інтегрована система поливу тепличного господарства.**
4. ***Тенета Є.В.*** (аспірант), ***Ситніков В.С.*** (д-р техн. наук, проф., завідувач кафедри комп'ютерних систем, Національний університет «Одеська політехніка», м. Одеса, Україна). **Нейронна компенсація інерційних коливань рівня пального з використанням LSTM і навчальних еталонів RAW → EXPECTED.**
5. ***Афонін Ю.С.*** (аспірант), ***Савінов В.Ю.*** (канд. техн. наук, доцент, доцент кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). Розподілена система гуманітарного розмінування з використанням глибокого навчання та комп'ютерного зору.

6. ***Гончаров Д.С.*** (аспірант кафедри комп'ютерної інженерії), ***Кандиба І.О.*** (PhD, ст. викладач кафедри інженерії програмного забезпечення, ЧНУ імені Петра Могили, м. Миколаїв, Україна). Аналіз даних сервісу Google Fit.
7. ***Гріднєв А.Ю.*** (магістрант), ***Дарнапук Є.С.*** (PhD, доцент (б. в. з.) кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Концепт інформаційно-аналітичної системи для візуалізації даних медичних досліджень.**
8. ***Гюльмамедов Н.М.*** (бакалаврант), ***Бурлаченко І.С.*** (старший викладач кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Контролери PWM-сигналів для керування сервомоторами у мультиагентних роботизованих системах.**
9. ***Доценко Д.В.*** (магістрант), ***Крайник Я.М.*** (канд. техн. наук, доцент, доцент кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Реалізація комбінованого методу стиснення проміжних кадрів відео у вебзастосунку.**
10. ***Жуковський Д.С.*** (магістрант), ***Журавська І.М.*** (д-р техн. наук, проф., завідувач кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **IoT-мережа із захистом на основі алгоритмів «легкої криптографії».**
11. ***Завгородній К.С.*** (магістрант), ***Дарнапук Є.С.*** (PhD, доцент (б. в. з.) кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **IoT-система збору та первинної обробки біомедичних показників пацієнтів на базі Raspberry PI та ESP32.**
12. ***Кайданович М.В.*** (магістрант), ***Журавська І.М.*** (д-р техн. наук, професор, завідувач кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Емуляція та візуалізація IoT-даних у реальному часі з використанням протоколу WebSocket.**
13. ***Мельников А.Г.*** (бакалаврант), ***Салтовський Б.Г.*** (старший викладач кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Пристрій керування на основі датчика APDS-9960.**
14. **Невідомий Д. О.** (бакалаврант), **Пузирьов С. В.** (канд. фіз.-мат. наук, доцент кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Система об'єднання відеопотоків у реальному часі на базі Raspberry PI.**
15. **Старченко В. В.** (старший викладач кафедри комп'ютерної інженерії, ЧНУ імені Петра Могили, м. Миколаїв, Україна). **Генерація фрактальних зображень з заданими просторово-**