Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та безпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

ДОСЛІДЖЕННЯ СИСТЕМИ СПОСТЕРЕЖЕННЯ ЗА ПАЦІЄНТАМИ З ВИКОРИСТАННЯМ МЕРЕЖІ ПУЛЬСОКСИМЕТРІВ

RESEARCH ON A PATIENT MONITORING SYSTEM USING A NETWORK OF PULSE OXIMETERS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІм-21
Трофімов Владислав Валерійович

(підпис)

Керівник:
к.т.н., доцент
Поліщук Микола Миколайович

(підпис)

Кваліфікаційну роботу
допущено до захисту
«____» грудня 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Гринюк Сергій Васильович

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Т.ТЕРЛЕЦЬКИЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Трофімову Владиславу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Дослідження системи спостереження за пацієнтами з використанням мережі пульсоксиметрів*

Керівник роботи *к.т.н., доцент Поліщук М. М.*

затверджені наказом закладу вищої освіти від «17» червня 2025 року № 290/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 09.12.2025р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Огляд і аналіз предметної області
Теоретичні основи вибору технлогій для розробки IoT пульсоксометра
Розробка об'єкту дослідження
Висновки

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Огляд і аналіз предметної області	*Поліщук М .М., доцент*		
Теоретичні основи вибору технлогій для розробки IoT пульсоксометра	*Поліщук М .М., доцент*		
Розробка об'єкту дослідження	*Поліщук М .М., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Гринюк С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І., ст.викладач*		

7. Дата видачі завдання 18.06.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми*	До 01.08.2025 р.	
2.	*Огляд і аналіз предметної області*	До 20.08.2025 р.	
3.	*Теоретичні основи вибору технологій для розробки IoT пульсоксометра*	До 25.09.2025 р.	
4.	*Розробка об'єкту дослідження*	До 20.10.2025 р.	
5.	*Висновки та пропозиції*	До 25.10.2025 р	
6.	*Формування списку використаних джерел*	До 27.10.2025 р.	
7.	*Формування додатків*	До 30.10.2025 р.	
8.	*Оформлення ілюстративного матеріалу*	До 05.11.2025 р.	
9.	*Представлення остаточного варіанту кваліфікаційної роботи керівникові*	До 11.11.2025 р.	
10.	*Нормоконтроль*	До 29.11.2025 р.	
11.	*Інструментальна перевірка на академічний плагіат*	До 02.12.2025 р.	
12.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	До 09.12.2025 р.	

Здобувач вищої освіти _____ Трофімов В. В.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Поліщук М. М.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Трофімов В. В. Дослідження системи спостереження за пацієнтами з використанням мережі пульсоксиметрів. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, додатків.

Перший розділ у ньому розглядаються основи принципів роботи пульсоксиметрії, сучасні методи вимірювання фізіологічних параметрів, архітектура IoT-систем та особливості побудови бездротових сенсорних мереж. Значну увагу приділено аналізу існуючих рішень для дистанційного медичного моніторингу, їх перевагам і недолікам, а також можливостям використання MQTT та MQTT-SN для передачі даних у реальному часі.

В другому розділі здійснено вибір та обґрунтування засобів розробки системи моніторингу. Обрано FastAPI для створення серверної частини з підтримкою WebSocket-з'єднань і обробки даних у реальному часі. Як транспорт телеметрії використано протоколи MQTT та MQTT-SN, що забезпечують ефективну роботу пульсоксиметрів на базі MAX30102 та Arduino Nano ESP32. Для клієнтського інтерфейсу застосовано HTML, CSS та JavaScript, а для візуалізації показників – бібліотеку Chart.js.

Третій розділ присвячено опису розробленої системи моніторингу стану пацієнтів та її програмних модулів. У цьому розділі детально представлено роботу серверної частини на FastAPI, механізм WebSocket-передавання даних у реальному часі, а також логіку приймання телеметрії через MQTT-SN від пульсоксиметрів на базі MAX30102 та Arduino Nano ESP32.

Ключові слова: пульсоксиметр, моніторинг пацієнтів, SpO_2, частота серцевих скорочень, MQTT-SN, IoT, FastAPI, WebSocket, Arduino Nano ESP32, MAX30102, візуалізація даних, Chart.js, реальний час

ANNOTATION

Trofimov V. Investigation of a Patient Monitoring System Using a Network of Pulse Oximeters. Manuscript.

Qualifying work of a Master's of EP «Computer Engineering» specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

Qualification work consists of an introduction, three sections, conclusions, a references, three appendices.

The first section examines the fundamental principles of pulse oximetry, modern methods of measuring physiological parameters, IoT system architecture, and the specifics of building wireless sensor networks. Considerable attention is paid to the analysis of existing remote medical monitoring solutions, their advantages and limitations, as well as the applicability of MQTT and MQTT-SN for real-time data transmission.

In the second section, focuses on the selection and justification of development tools for the monitoring system. FastAPI was chosen to implement the server side with WebSocket support and real-time data handling. MQTT and MQTT-SN protocols were selected as telemetry transport layers to ensure efficient communication with MAX30102-based pulse oximeters and Arduino Nano ESP32 devices. HTML, CSS, and JavaScript were used for the client-side interface, while Chart.js was employed for data visualization.

The third section presents the developed patient monitoring system and its software modules. This chapter provides a detailed description of the FastAPI server logic, the WebSocket-based real-time data streaming mechanism, and the MQTT-SN telemetry reception workflow from MAX30102- and ESP32-based pulse oximeters.

Keywords: pulse oximeter, patient monitoring, SpO_2, heart rate, MQTT-SN, IoT, FastAPI, WebSocket, Arduino Nano ESP32, MAX30102, data visualization, Chart.js, real time.

ЗМІСТ

ВСТУП

Сучасний розвиток медицини нерозривно пов’язаний із використанням інформаційних технологій та систем автоматизованого збору фізіологічних показників. Одним із найважливіших напрямів цифрової трансформації охорони здоров’я є створення систем моніторингу стану пацієнтів у реальному часі. Такі системи дозволяють відстежувати життєво важливі параметри організму, забезпечують раннє виявлення відхилень та можуть істотно знизити навантаження на медичний персонал.

Одним із базових показників, що використовується для оцінки функціонального стану організму, є рівень насичення крові киснем (SpO_2) та частота серцевих скорочень (ЧСС). Для їх вимірювання застосовуються пульсоксиметри – неінвазивні сенсорні пристрої, які використовують фотоплетизмографічний метод для визначення насичення гемоглобіну киснем. Застосування мережі таких пристроїв дозволяє створити інтегровану систему спостереження за групою пацієнтів, що особливо актуально у відділеннях інтенсивної терапії, кардіології, а також для дистанційного моніторингу хронічно хворих пацієнтів.

В умовах повномасштабної війни проти України особливої актуальності набуває розвиток телемедичних рішень, які дозволяють забезпечити безперервне спостереження за станом пацієнтів навіть у складних умовах. Такі системи можуть використовуватися у польових госпіталях, евакуаційних пунктах та лікарнях із перевантаженим персоналом. Автоматизований контроль життєво важливих показників, зокрема рівня насичення крові киснем і частоти серцевих скорочень, сприяє своєчасному реагуванню на критичні зміни стану поранених та хворих, що безпосередньо впливає на збереження життя людей.

В останні роки активно розвиваються технології Інтернету речей (IoT), які відкривають можливість об’єднання численних сенсорів у єдину мережу. Використання безпровідних мікроконтролерів, таких як Arduino, у поєднанні з датчиками пульсоксиметрії (наприклад, MAX30102), створює передумови для

розробки доступних та масштабованих систем моніторингу. Такі рішення здатні у реальному часі передавати показники стану пацієнтів на центральний сервер або у хмарну інфраструктуру для подальшої обробки та візуалізації даних.

Актуальність теми полягає у необхідності дослідження архітектури та принципів побудови систем спостереження за пацієнтами на основі мережі пульсоксиметрів. Подібні системи повинні забезпечувати стабільність передачі даних, своєчасне виявлення аномалій у показниках, а також масштабованість при збільшенні кількості підключених сенсорів. Розробка прототипу такої системи сприятиме розвитку технологій медичного моніторингу та створенню інтелектуальних засобів підтримки прийняття рішень у медицині.

Мета дослідження полягає у розробленні та дослідженні прототипу системи спостереження за пацієнтами з використанням мережі пульсоксиметрів, що передають дані у режимі реального часу на моніторинговий сервіс.

Завдання дослідження:

– провести аналіз існуючих технічних рішень для моніторингу фізіологічних параметрів пацієнтів;

– дослідити принцип роботи пульсоксиметрів і обґрунтувати вибір датчика для системи;

– обрати апаратну платформу та спроектувати схему сенсорного вузла;

– розробити структуру програмного забезпечення для збору та передачі даних;

– створити прототип моніторингового сервісу для відображення інформації про пацієнтів у реальному часі;

– провести тестування системи з використанням програмного генератора сигналів і проаналізувати результати.

Об’єкт дослідження – процес збору та передавання фізіологічних даних у системах спостереження за пацієнтами.

Предмет дослідження – методи побудови мережі пульсоксиметрів та засоби обробки і візуалізації даних у системах моніторингу.

Методи дослідження – системний аналіз, порівняння технічних рішень, моделювання процесів передачі даних, розроблення апаратних схем і програмних модулів, експериментальної перевірки працездатності системи.

Інформаційна база дослідження – технічна документація на сенсор MAX30102, специфікації мікроконтролерів Arduino та ESP32, стандарти безпровідних комунікацій BLE та Wi-Fi, а також наукові публікації, присвячені застосуванню технологій Інтернету речей у медичній галузі.

Апробація роботи на Міжнародній науково-практичній конференції молодих вчених та студентів (6 травня 2025 р) [1] та у науковій статті в Технічних вістях [2], де було висвітлено основні положення кваліфікаційної роботи.

РОЗДІЛ 1
ОГЛЯД І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі моніторингу стану пацієнтів

Сучасна медицина переживає період глибокої цифрової трансформації, у межах якої значна увага приділяється використанню інформаційних технологій для забезпечення дієвої діагностики, контролю та лікування пацієнтів. Одним з ключових напрямів розвитку є системи моніторингу фізіологічних параметрів у режимі реального часу.

Їхнє впровадження стало можливим завдяки стрімкому розвитку електронних сенсорів, мікроконтролерів та технологій Інтернету речей (IoT), які дозволяють збирати, передавати, обробляти та візуалізувати дані практично без участі медичного персоналу. Окрім того, значна кількість пацієнтів потребує довготривалого дистанційного спостереження через хронічні захворювання, серцево-судинні патології, дихальні розлади, порушення кровообігу чи постопераційний нагляд.

У контексті України актуальність проблеми зростає ще більше. В умовах воєнного стану і підвищеного навантаження на медичну систему виникає потреба у доступних, мобільних та автономних рішеннях, здатних забезпечити оперативний збір і передачу медичних показників. Польові госпіталі, мобільні пункти стабілізації та тимчасові медичні центри часто не мають можливості постійного спостереження за кожним пацієнтом, а кількість кваліфікованого персоналу обмежена. Портативні пульсоксиметричні мережі можуть стати вирішенням цієї проблеми, оскільки вони дають змогу одночасно контролювати десятки пацієнтів, оперативно визначати критичні зміни у їхньому стані та знижувати ризики запізнілого реагування.

Одним з головних параметрів, які підлягають контролю, є рівень насичення крові киснем (SpO_2). Порушення кислородного обміну є показником значної кількості патологічних станів, включаючи пневмонії, інфекційні захворювання дихальних шляхів, травми грудної клітки, втрати крові, шокові

стани та інші ситуації, у яких від своєчасного виявлення змін критично залежить життя пацієнта.

Потреба у доступних інструментах для спостереження стала основою розвитку неінвазивних сенсорних технологій, зокрема фотоплетизмографії, яка лежить в основі роботи сучасних пульсоксиметрів. Їхня компактність, низьке енергоспоживання та простота використання роблять можливим вбудовування таких сенсорів у портативні пристрої, мобільні датчики чи навіть носимі медичні системи.

Проте складність полягає не лише у вимірюванні показників, а й у їхній оперативній передачі, зберіганні та обробці. Саме тут важливу роль відіграють IoT-технології, що дозволяють будувати розподілені мережі сенсорів із можливістю централізованого контролю.

Значну увагу також привертає тенденція до створення відкритих, недорогих і модульних рішень, що можуть бути адаптовані до різних умов і сценаріїв використання. На відміну від дорогого комерційного обладнання, системи на базі доступних компонентів дозволяють розробляти кастомізовані рішення для широкого спектра потреб – від домашнього спостереження за хворими до розгортання польових систем у зоні бойових дій.

Таким чином, актуальність задачі створення системи моніторингу стану пацієнтів на основі мережі пульсоксиметрів визначається низкою чинників: розвитком телемедицини, потребою у доступних автономних рішеннях, зростаючою кількістю пацієнтів, що потребують безперервного контролю, а також необхідністю оптимізації роботи медичного персоналу.

Впровадження IoT-систем, здатних збирати та аналізувати дані з декількох сенсорів у реальному часі, дозволяє суттєво підвищити якість медичного обслуговування, своєчасно виявляти ризики та забезпечувати більш ефективне використання наявних ресурсів. Це робить досліджувану тему важливою як у науковому, так і у практичному аспекті та визначає її значний потенціал для подальшого розвитку.

1.2 Фізіологічні основи пульсоксиметрії

Пульсоксиметрія є одним із ключових методів неінвазивного контролю фізіологічного стану людини, який дозволяє оперативно визначати рівень насичення гемоглобіну киснем (SpO_2) та частоту серцевих скорочень (ЧСС). Попри простоту використання та компактність сучасних пристроїв, принципи їхньої роботи ґрунтуються на складних фізичних процесах, сформованих на перетині оптики, електроніки та біомедичної інженерії.

1.2.1 Історичні аспекти та розвиток технології

Перші розробки в області фотометричного контролю стану крові з’явилися ще на початку XX століття. У 1930-х роках японський учений Таказі Аоягі експериментально довів можливість визначення рівня оксигенації крові на основі характеристики поглинання світла різних довжин хвиль. Саме він сформулював основну ідею сучасної пульсоксиметрії – використання двох світлових каналів (червоного та інфрачервоного), що по-різному поглинаються оксигемоглобіном і дезоксигемоглобіном.

Сучасний вигляд пульсоксиметри набули у 1970-1980-х роках, коли електронні схеми стали достатньо компактними, а мікропроцесори – доступними для масового використання (рис. 1.1). З того часу технологія зазнала значної еволюції: від громіздких стаціонарних блоків до мініатюрних сенсорів, які інтегруються в мобільні пристрої, розумні годинники та IoT-системи.

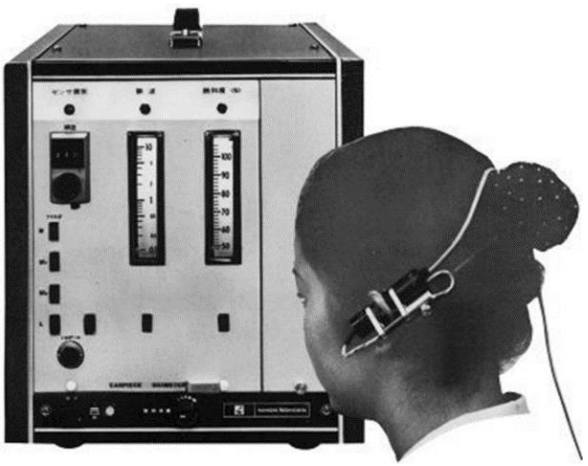

Рисунок 1.1 – Перший практичний пульсоксометр [3]

1.2.2 Фізичні основи методу фотоплетизмографії

Основою роботи пульсоксиметра є фотоплетизмографія (ФПГ) – метод, що вимірює зміну оптичних властивостей тканин під дією пульсового кровотоку [3].

Механізм побудований на кількох важливих закономірностях, а саме гемоглобіну у різних формах, які по-різному поглинають світло. Оксигемоглобін (HbO_2) більше поглинає інфрачервоне світло (~940 нм), дезоксигемоглобін (Hb) сильніше поглинає червоне світло (~660 нм) (рис. 1.2).

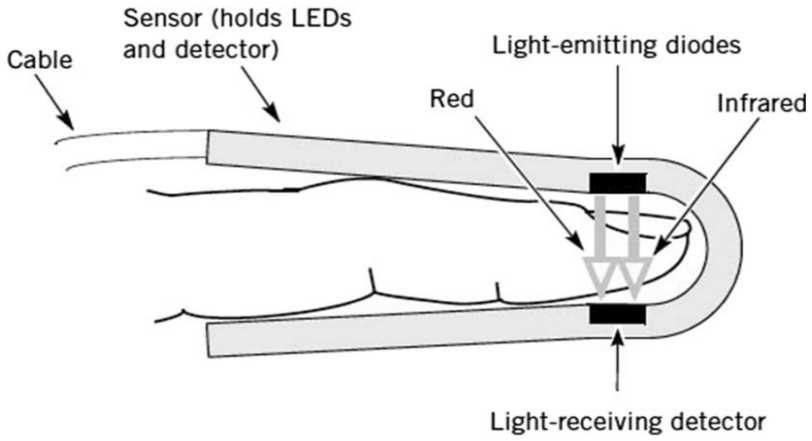

Рисунок 1.2 – Схема роботи пусоксометра [4]

Пульсова хвиля модулює інтенсивність світла, що проходить крізь тканини. У момент систоли об’єм крові в периферичних судинах збільшується, через що поглинальна здатність тканин зростає і на фотодетектор потрапляє менше світла (рис. 1.3).

Рисунок 1.3 – Систола і діастола [5]

У фазі діастоли, коли об’єм крові зменшується, інтенсивність отриманого світла відповідно підвищується. Це періодичне чергування рівнів поглинання формує фотоплетизмографічний сигнал, який дозволяє виділити пульсову компоненту та визначити частоту серцевих скорочень, а також оцінити насичення гемоглобіну киснем за співвідношенням інтенсивностей світла різних довжин хвиль

Парatканинна частина сигналу є сталою. Кістки, м’язи та жирова тканина поглинають світло приблизно однаково впродовж циклу серцебиття, тому алгоритм виділяє пульсуючу компоненту сигналу.

Таким чином, пульсоксиметр реєструє двокомпонентний фотоплетизмографічний сигнал:

– AC-компонент – змінна частина, пов’язана з пульсовим наповненням;

– DC-компонент – постійна частина, яка залежить від статичної структури тканин.

Саме аналіз співвідношення AC/DC у двох каналах дозволяє розрахувати рівень насичення крові киснем.

1.3 Принцип визначення SpO_2

Отримане значення R порівнюється з емпіричними кривими калібрування, на основі яких обчислюється сатурація згідно формули (1.1). Ключові моменти: правильне визначення AC-компоненти є критично важливим, співвідношення каналів забезпечує стійкість методу до змін освітлення, калібрувальні таблиці вбудовані у мікросхему або програмні бібліотеки.

$$R = \frac{(AC_{red}/DC_{red})}{(AC_{IR}/DC_{IR})}\ , \qquad (1.1)$$

де ACred – змінна складова сигналу червоного каналу;

DCred – постійна складова сигналу червоного каналу;

ACIR – змінна складова сигналу інфрачервоного каналу;

DCIR – постійна складова сигналу інфрачервоного каналу.

У сучасних сенсорах, таких як MAX30102, значна частина обробки (усереднення, фільтрація, компенсація шумів) виконується апаратно, що суттєво підвищує точність вимірювання.

Особливості вимірювання частоти серцевих скорочень визначається шляхом аналізу періодичності пульсуючої АС-компоненти ФПГ-сигналу. Сучасні алгоритми використовують: пошук локальних максимумів та мінімумів, фільтри рухомого середнього, цифрові фільтри низької та високої частоти, адаптивні порогові значення. Це дозволяє мінімізувати вплив: рухових артефактів, зовнішнього світла, недостатнього прилягання сенсора. У результаті ЧСС вимірюється з точністю до ±2 уд/хв, що є достатнім для більшості клінічних та польових застосувань.

1.4 Види пульсоксиметрів та особливості їхнього застосування

Пульсоксиметри, як медичні прилади, мають широкий спектр варіацій, що зумовлено різними умовами експлуатації, вимогами до точності та призначенням. Умовно їх можна класифікувати за конструкцією, принципом роботи, способом носіння та функціональною складністю. Розглянемо основні типи більш детально, з акцентом на особливостях їхнього застосування в клінічних і позаклінічних умовах.

Пальцеві пульсоксиметри є найпоширенішим типом завдяки своїй портативності та простоті використання. Вони зазвичай мають прищіпкову форму і призначені для короткочасного або періодичного вимірювання SpO_2 та ЧСС. Незважаючи на компактність, такі пристрої забезпечують достатньо високу точність для побутового використання.

Їхнім головним недоліком є залежність від правильного розміщення на пальці та чутливість до рухових артефактів. У медичних закладах вони

застосовуються для швидкого первинного контролю стану пацієнтів, проте не підходять для постійного моніторингу (рис. 1.4).

Рисунок 1.4 – Пальцевий пульсоксометр [6]

Наручні пульсоксиметри, інтегровані у фітнес-браслети та розумні годинники, стали надзвичайно популярними завдяки поширенню носимих пристроїв. Вони використовують оптичні сенсори, вбудовані в нижню частину корпусу, і забезпечують безперервне спостереження за частотою серцевих скорочень та інколи SpO_2. Однак їхня точність часто поступається клінічним пристроям через складність забезпечення щільного прилягання сенсора до шкіри, а також через значні рухові артефакти (рис. 1.5).

Рисунок 1.5 – Наручний пульсоксометр [7]

Стаціонарні медичні пульсоксиметри є високоточними приладами, які застосовуються в інтенсивній терапії, операційних, реанімаційних та діагностичних кабінетах. Вони використовують професійні сенсорні датчики з мінімальною похибкою, здатні працювати навіть за умов низької перфузії. Такі пристрої оснащені можливістю довготривалого моніторингу, збереженням історії вимірювань, сигналізацією та підключенням до лікарняних інформаційних систем. Їх головним недоліком є висока вартість, громіздкість та залежність від стаціонарної інфраструктури (рис. 1.6).

Рисунок 1.6 – Стаціонарний пульсоксометр [8]

Пульсоксиметри для новонароджених є спеціалізованими пристроями, призначеними для безперервного контролю рівня насичення крові киснем та частоти серцевих скорочень у дітей з перших хвилин життя. На відміну від дорослих моделей, вони використовують мініатюрні, надчутливі датчики, здатні працювати при дуже низькому перфузійному індексі й не травмувати тонку шкіру немовлят.

Такі пристрої широко застосовуються у відділеннях інтенсивної терапії новонароджених, під час оцінки адаптації після пологів, а також для раннього виявлення респіраторних порушень та критичних вроджених вад серця (рис. 1.7).

Рисунок 1.7 – Пульсоксометр для немовлят [9]

IoT-пульсоксиметри – це новий клас сенсорів, розроблених для інтеграції в мережеві системи. Вони складаються з оптичного сенсора пульсоксиметрії (наприклад, MAX30102, MAX30100) та мікроконтролера з бездротовим модулем (ESP32, nRF52840, RP2040 з BLE тощо). Основна перевага таких пристроїв, це можливість віддаленого й безперервного збору даних, їх миттєвої передачі до центрального сервера та масштабування системи на довільну кількість сенсорних вузлів. Таким чином, IoT-пульсоксиметри є оптимальними для мобільних госпіталів, польових медичних підрозділів, систем домашнього спостереження та наукових досліджень. У даній роботі використовується сенсор MAX30102 у поєднанні з Arduino Nano ESP32, що дозволяє отримувати

високоточні вимірювання та передавати їх за допомогою Bluetooth або Wi-Fi у режимі реального часу (рис. 1.8).

Рисунок 1.8 – IoT пульсоксометр [10]

1.6 Особливості застосування систем моніторингу в умовах обмежених ресурсів

Попри високу ефективність та простоту застосування, пульсоксиметрія має низку технічних обмежень, які можуть впливати на точність вимірювання та стабільність сигналу. Розглянемо основні проблеми детальніше. Одним із найбільш значущих факторів є рухові артефакти. Будь-яке переміщення пальця, кисті чи самого сенсора може призвести до спотворення фотоплетизмографічного сигналу, оскільки зміна кута падіння та проникності світла впливає на форму сигналу.

У мобільних або військових умовах це є особливо критичним, адже пацієнти можуть знаходитися у стані стресу, гіпотермії або шоку. Ще одним поширеним обмеженням є низька перфузія кінцівок, тобто слабкий кровообіг у пальцях. Це спостерігається при холоді, шокових станах, вазоспазмах, діабетичній ангіопатії тощо. У таких випадках пульсуюча складова сигналу стає

надто малою, через що сенсор може не визначити коректне значення SpO_2 або взагалі втратити вимірювання.

Зовнішнє освітлення, особливо інтенсивні LED-джерела, може проникати в сенсор і створювати додатковий шум. Хоча більшість сенсорів мають захисні екрани та власні фільтри, ситуації з надмірно яскравим або мерехтливим світлом (наприклад, у операційних) можуть ускладнювати точне вимірювання. Також значну роль відіграють індивідуальні фізіологічні характеристики пацієнта: товщина шкіри, пігментація, наявність татуювань, вологість, наявність лаку на нігтях або косметичних засобів. Усе це впливає на здатність світла проникати в тканини та повертатися до фотодіода.

Серцеві аритмії також ускладнюють інтерпретацію сигналу. Сильна нерегулярність серцебиття може спотворити AC-компоненту фотоплетизмограми, що ускладнить коректний розрахунок ЧСС. Для компенсації цих ефектів у сучасних сенсорах застосовуються такі технології: цифрові фільтри низької та високої частоти, які відсікають шум, адаптивні алгоритми підсилення, що регулюють струм світлодіодів залежно від умов вимірювання, корекція фону (ambient light cancellation), включення декількох вибірок для усереднення сигналу, автоматичний вибір режиму sampling rate і pulse width для підвищення чутливості, вбудовані інтелектуальні алгоритми визначення пульсу, що відокремлюють корисний сигнал від шуму.

Сенсор MAX30102 має апаратні засоби фільтрації, регулювання підсилення та компенсації світла, що робить його придатним для роботи у складних польових умовах, де стабільність і надійність сигналу є критично важливими.

1.7 Значення пульсоксиметрії для IoT-систем медичного моніторингу

Пульсоксиметрія відіграє ключову роль у створенні сучасних систем дистанційного медичного моніторингу, особливо в рамках концепції Інтернету речей (IoT). Однією з основних переваг пульсоксиметрії в IoT є невеликий обсяг

даних, що передаються від сенсора: значення SpO_2, ЧСС, часовий штамп та ідентифікатор пристрою. Це робить метод ідеально адаптованим до MQTT та MQTT-SN – протоколів, оптимізованих для малопотужних мереж і обмежених каналів зв'язку [11]. Така структура телеметрії дозволяє передавати тисячі вимірювань з мінімальним навантаженням на мережу.

Іншою важливою перевагою є можливість безперервного моніторингу. Пульсоксиметрія дозволяє отримувати від одного сенсора десятки показників за хвилину, що дає змогу лікарям отримувати повну картину стану пацієнта у реальному часі. У військових умовах це особливо важливо при моніторингу поранених, пацієнтів з дихальною недостатністю або після операційних втручань.

IoT-пульсоксиметрія також забезпечує масштабованість – до системи можна підключити будь-яку кількість сенсорів, від кількох одиниць до сотень, без зміни архітектури та суттєвого навантаження на сервер. Кожен сенсор формується як окремий вузол у мережі, який передає стандартизовані повідомлення на брокер або моніторинговий сервер. Крім того, важливою є можливість інтеграції з іншими IoT-пристроями, такими як температурні сенсори, датчики ЕКГ, системи локації пацієнтів, дихальні монітори.

Додатковим фактором є автономність та енергоефективність. Використання ESP32 у поєднанні з низькопотужними сенсорами дозволяє створювати пристрої, що можуть працювати багато годин або навіть діб від акумулятора, що важливо під час евакуації, транспортування пацієнтів або роботи у зонах без доступу до електроживлення. Усе це робить пульсоксиметрію важливим і технологічно зручним інструментом для систем телемедичного спостереження, зокрема у рамках даної дипломної роботи, де реалізовано мережевий моніторинг на основі MAX30102 та ESP32 з передачею даних у реальному часі на сервер FastAPI.

РОЗДІЛ 2
ТЕОРЕТИЧНІ ОСНОВИ ВИБОРУ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ IOT ПУЛЬСОКСОМЕТРА

2.1 Аналіз сучасних підходів до побудови сенсорних IoT-систем медичного моніторингу

Сенсорні мережі для медичного моніторингу стали одним із ключових напрямів розвитку сучасних інформаційних технологій у сфері охорони здоров’я. Вони дозволяють здійснювати збір, обробку та передавання фізіологічних параметрів пацієнтів у режимі реального часу, забезпечуючи постійний нагляд за їхнім станом незалежно від місця перебування. На відміну від традиційного медичного обладнання, яке потребує безпосередньої присутності лікаря або медичного персоналу, IoT-системи забезпечують високий рівень автономності та доступності даних.

У межах даного дослідження особливу увагу приділено системам моніторингу SpO_2 і частоти серцевих скорочень (ЧСС), оскільки ці параметри є критично важливими для раннього виявлення гіпоксії, порушень кровообігу та інших патологічних станів. Створення мережі пульсоксиметрів дозволяє проводити спостереження одночасно за кількома пацієнтами, що робить такі рішення особливо актуальними для мобільних госпіталів, реанімаційних відділень, систем екстреної медичної допомоги та польових умов.

Для того щоб побудувати ефективну сенсорну мережу, необхідно враховувати цілу низку факторів: архітектуру взаємодії пристроїв, методи організації бездротового зв’язку, можливості обробки даних на рівні мікроконтролера, пропускну здатність каналів зв’язку, вимоги до енергоспоживання, а також умови експлуатації системи. У цьому підрозділі наведено основні підходи до створення сенсорних IoT-систем, їх переваги, недоліки та можливості застосування в умовах медичних закладів.

2.1.1 Архітектури моделі побудови сенсорних мереж

На сучасному етапі розвитку IoT-технологій існує кілька основних архітектурних моделей, які застосовуються для організації мереж сенсорних пристроїв. До найпоширеніших відносять централізовану, децентралізовану (mesh) та гібридну архітектури. Кожна з них має свої переваги й обмеження, що визначають доцільність їх використання у конкретних сценаріях медичного моніторингу.

Централізована архітектура Це найпростіша й найпоширеніша модель, у якій усі сенсорні вузли передають дані на один центральний сервер або шлюз. Саме сервер відповідає за обробку інформації, її подальшу маршрутизацію та зберігання. Такий підхід забезпечує: контроль над усіма потоками даних, спрощену логіку мікроконтролерів, які виконують лише збір та передачу інформації, можливість підключення нових сенсорів без зміни конфігурації існуючої мережі, централізовані алгоритми виявлення аномалій та аналізу стану пацієнтів.

Недоліком цієї архітектури є залежність від працездатності центрального вузла, а також високе навантаження на нього при великій кількості сенсорів. Однак для систем із 5-50 пристроями, що відповідає типовим сценаріям польових медичних пунктів, централізована модель є найбільш оптимальною (рис. 2.1).

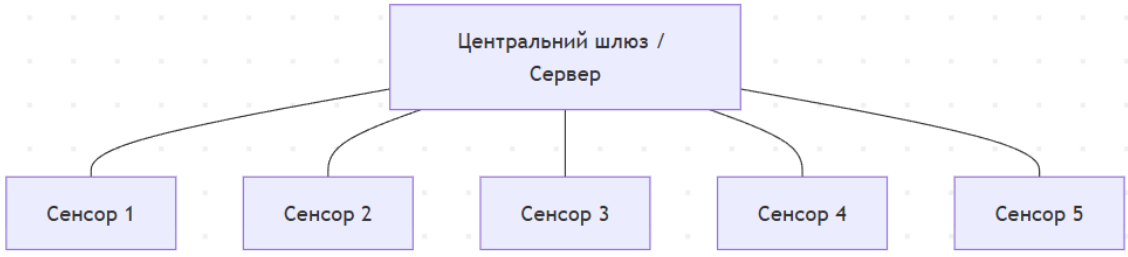

Рисунок 2.1 – Централізована архітектура

У мережах типу mesh сенсорні вузли можуть передавати дані один одному, створюючи самовідновлювану мережу з множинними маршрутами. Такий підхід

забезпечує: високу живучість мережі у разі виходу з ладу окремих вузлів, можливість охоплення великих площ без потужних передавачів, балансування навантаження між вузлами. Проте mesh-системи складні у розробці та конфігурації, мають збільшені затримки передачі даних, а також потребують потужніших мікроконтролерів через складність маршрутів. У задачах медичного моніторингу, де важлива мінімальна затримка та висока передбачуваність роботи, mesh мережі використовуються рідше (рис. 2.2).

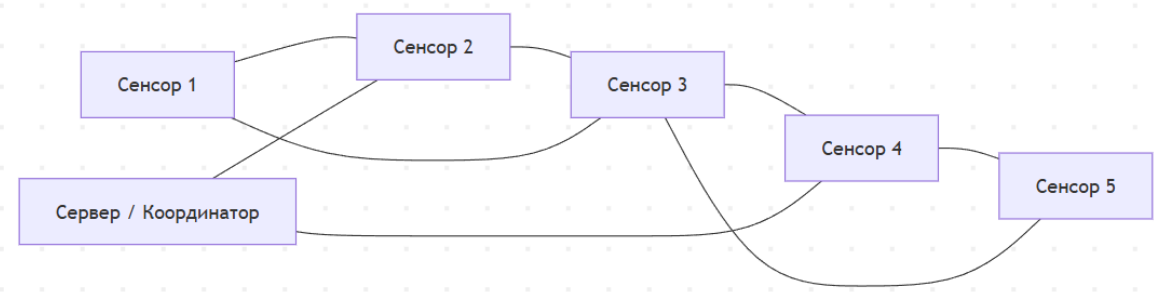

Рисунок 2.2 – Mesh архітектура

Гібридні архітектури поєднують елементи централізованої та децентралізованої моделей. Їх застосування виправдане в масштабних медичних системах із сотнями пацієнтів, де необхідно поєднувати живучість та мінімальні затримки (рис. 2.3).

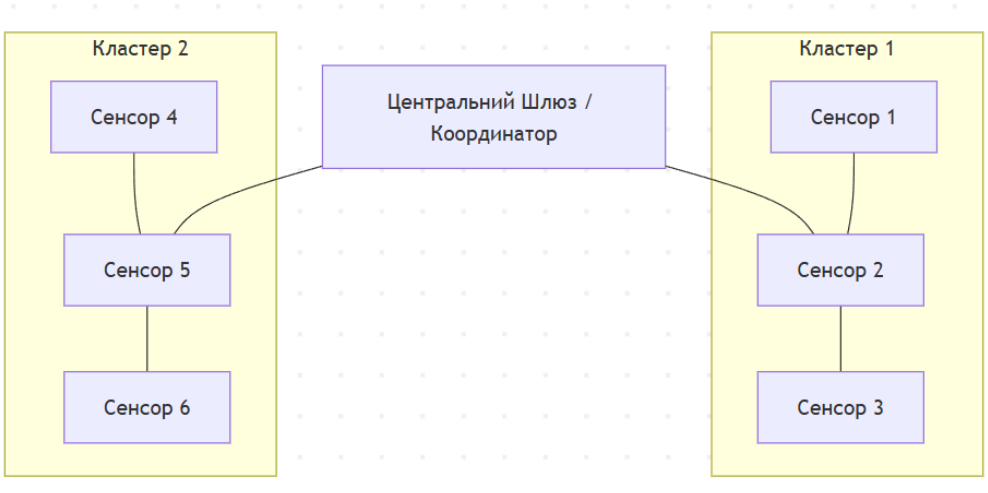

Рисунок 2.3 – Гібридна архітектура

У рамках цього дослідження використано централізовану архітектуру, яка забезпечує прогнозованість роботи, низькі затримки й простоту розширення системи.

2.1.2 Підходи до інтеграції сенсорів у медичні IoT-системи

Існує два основні підходи до інтеграції сенсорних модулів у системи моніторингу: локальна обробка даних на мікроконтролері та передавання сирих даних на сервер.

У нашому проєкті застосовується перший підхід – обчислення на мікроконтролері, що оптимально для IoT-пристроїв і не потребує високошвидкісних каналів (табл. 2.1).

Таблиця 2.1 – порівняння підходів інтеграції сенсорів в IoT-системи

Критерій	Локальна обробка даних на мікроконтролері	Передавання сирих даних на сервер
Суть підходу	Мікроконтролер зчитує сигнал PPG, фільтрує його, обчислює SpO_2 і ЧСС, передає готові показники.	Мікроконтролер лише формує сирий сигнал і надсилає його на сервер для подальшої обробки.
Переваги	Зниження навантаження на сервер, менша кількість передаваних даних, менше енергоспоживання, швидша реакція системи.	Максимальна точність обробки, можливість застосування складних алгоритмів аналізу (ШІ, адаптивні фільтри).
Недоліки	Ускладнена прошивка пристрою, точність залежить від ресурсів мікроконтролера.	Великий обсяг трафіку, вищі вимоги до стабільності та пропускної здатності каналу зв’язку.
Оптимальні випадки використання	Автономні IoT-сенсори, портативні системи, мережі з багатьох пристроїв, системи з обмеженим каналом зв’язку.	Лабораторні системи, медичні аналізатори, серверні комплекси, дослідження сигналів високої точності.
Приклад застосування в проекті	Підхід, застосований у даній роботі: ESP32 розраховує SpO_2 і ЧСС та передає готові JSON-дані.	Може бути використано у майбутньому для більш поглибленої аналітики (збір сирих PPG-сигналів).

Такий підхід дає змогу зменшити навантаження на мережу, оскільки передаються вже оброблені, компактні пакети даних.

2.1.3 Пульсоксиметрія як ключовий метод вимірювання в IoT-моніторингу

З огляду на популярність пульсоксиметрів у клінічному середовищі, їх інтеграція у IoT-системи стала закономірним розвитком технологій. Сучасні сенсорні модулі забезпечують:

– цифровий інтерфейс передачі даних;

– можливість фільтрації сигналу на апаратному рівні;

– оптимізацію енергоспоживання;

– готові драйвери та бібліотеки.

MAX30102, використаний у даному дослідженні, є одним із найпоширеніших модулів, що поєднує компактність, точність та програмну гнучкість.

2.1.4 Тенденції розвитку медичних IoT-систем

Серед ключових тенденцій слід виокремити: мініатюризацію сенсорів – можливість розміщення датчиків у браслетах, пластирах або медичних пов’язках, зростання ролі BLE та Wi-Fi IoT, що забезпечують низьку затримку та низькі витрати енергії, використання WebSocket для передачі телеметрії у реальному часі, перехід до безшлюзових архітектур, де мікроконтролер напряму спілкується із сервером, активну стандартизацію (IEEE 11073, Bluetooth HDP), збільшення ролі алгоритмів на стороні клієнта, що допомагає зменшити навантаження на сервер [12].

2.2 Вибір апаратних засобів для побудови сенсорної IoT-системи моніторингу

Проєктування сенсорної мережі для медичного моніторингу потребує ретельного вибору апаратних компонентів, які здатні забезпечити точність вимірювань, стабільність роботи та надійність передавання даних у різних умовах експлуатації. У цьому підрозділі розглянуто характеристики ключових апаратних елементів системи – мікроконтролерів, сенсорних модулів, інтерфейсів підключення та допоміжних електронних компонентів, що

формують основу IoT-платформи. Аналіз проведено з урахуванням вимог проєкту: мобільність, низьке енергоспоживання, підтримка бездротового зв'язку, можливість роботи в автономних польових умовах та стабільність обслуговування кількох сенсорних вузлів.

2.2.1 Мікроконтролери як основа сенсорного вузла

Мікроконтролер відіграє центральну роль у системі – він виконує функції збору даних, керування сенсором, обробки сигналів і передавання результатів на сервер. Вибір конкретної платформи визначає можливості всієї системи: продуктивність, рівень енергоспоживання, підтримку бездротових протоколів та масштабованість.

У межах дослідження проведено порівняння трьох найпоширеніших варіантів: Arduino Uno R3 [13], Arduino Pro Mini [14] та Arduino Nano ESP32 [15]. Такий вибір зумовлений широкою популярністю цих платформ, наявністю великої кількості бібліотек та спрощеним процесом розробки.

З урахуванням характеристик та вимог проєкту саме Arduino Nano ESP32 є оптимальною платформою для побудови сенсорного вузла (табл. 2.2).

Таблиця 2.2 – Порівняння вищенаведених мікроконтроллерів

Параметр	Arduino Uno R3	Arduino Pro Mini	Arduino Nano ESP32
Мікроконтролер	ATmega328P	ATmega328P	ESP32-S3
Пам'ять (Flash / SRAM)	32 КБ / 2 КБ	32 КБ / 2 КБ	16 МБ / 512 КБ
Інтерфейси	UART, SPI, I²C	UART, SPI, I²C	UART, SPI, I²C, Wi-Fi, BLE
Робоча напруга	5 В	3,3 / 5 В	3,3 В
Частота	16 МГц	8-16 МГц	240 МГц
Бездротовий зв'язок	Відсутній	Відсутній	Wi-Fi + Bluetooth LE
Порт для програмування	USB Type-B	Через зовнішній адаптер	USB-C або OTA
Енергоспоживання	Середнє	Низьке	Дуже низьке
Вартість	~$10-12	~$6-8	~$9-11
Придатність для проєкту	Базовий варіант	Обмежений	Найкращий вибір

2.2.2 Обґрунтування вибору сенсорного модуля

Пульсоксиметричні сенсори, представлені на ринку, відрізняються точністю, типом оптичних елементів, алгоритмами обробки та умовами застосування. Для IoT-систем особливо важливою є можливість цифрового інтерфейсу, низьке енергоспоживання та наявність готових драйверів.

У рамках дослідження обрано модуль MAX30102, який поєднує два світлодіоди (червоний та інфрачервоний) і фотодетектор у компактному корпусі [16]. Його ключові переваги: цифровий інтерфейс I^2C, низьке енергоспоживання (від 600 мкА у стандартному режимі), високочутливий фотодетектор, наявність вбудованого АЦП та фільтрів для зменшення шумів, програмовані режими роботи, сертифікована бібліотека для обчислення SpO_2 та ЧСС у реальному часі. Ці характеристики роблять MAX30102 оптимальним для компактних пристроїв медичного призначення (рис. 2.4).

Рисунок 2.4 – Зовнішній вигляд сенсору MAX10302

Для порівняння також розглядаються:

– MAX30100 – попередня версія, менш стабільна у шумних умовах;

– AFE4400 – професійний фотоплетизмографічний модуль, але значно дорожчий;

– MAX86150 – забезпечує ЕКГ + PPG, але вимагає складнішу обробку.

З огляду на співвідношення ціни та можливостей саме MAX30102 є найбільш раціональним вибором.

2.2.3 Особливості інтеграції сенсора з мікроконтролером

Під’єднання MAX30102 передбачає використання інтерфейсу I²C, що дає можливість реалізувати просту й надійну комунікацію. Для коректної роботи сенсора необхідно забезпечити: живлення 3,3 В, наявність ліній SDA та SCL із підтягувальними резисторами, правильне налаштування параметрів sampling rate, оптимізацію яскравості світлодіодів для економії енергії, стабільне прилягання сенсора до поверхні шкіри. У фірмових бібліотеках передбачено фільтри для компенсації шумів, що суттєво спрощує інтеграцію.

2.2.4 Організація живлення та автономності системи

Для мобільних медичних систем важливо забезпечити можливість автономної роботи. Arduino Nano ESP32 підтримує кілька режимів енергозбереження: Modem-sleep: вимкнення бездротових модулів при низькому навантаженні, Light-sleep: часткове вимкнення периферії, Deep-sleep: мінімальне споживання для роботи від батарей. Як джерела живлення використовуватимуться Li-Po пакети з живленням 3,7 В, модулі зарядки типу TP4056, стабілізатори AMS1117-3.3 або аналогічні. Додатковим фактором є стабільність струму для сенсора MAX30102, оскільки некоректне живлення призводить до шумів у ППГ сигналі.

2.2.5 Вимоги до фізичного розміщення сенсора та ергономіка

Для коректного вимірювання фізіологічних параметрів важливо забезпечити: щільне прилягання сенсора до поверхні шкіри, мінімальний вплив зовнішнього освітлення, фіксацію пальця або тканини, ізоляцію від рухових артефактів. У мобільних системах часто застосовуються: м’які силіконові кріплення, текстильні манжети, жорсткі корпуси з прорізами для пальця, магнітні фіксатори. Ергономіка також впливає на точність показників, особливо при тривалому моніторингу.

2.2.6 Підсумкове обґрунтування вибору апаратної частини

Проведений аналіз дозволив сформувати оптимальну конфігурацію апаратного забезпечення:

– мікроконтролер – Arduino Nano ESP32 (через підтримку BLE/Wi-Fi та високу продуктивність);

– сенсор – MAX30102 (з високою точністю вимірювань та оптичною стабільністю);

– інтерфейс даних – I²C;

– живлення – Li-Po акумулятор та стабілізатор 3,3 В;

– конструкція вузла – компактний модуль із ізоляцією від зовнішнього світла.

Ця конфігурація забезпечує стабільну роботу системи, гарантує високу якість вимірювань та дозволяє масштабувати мережу до десятків пристроїв.

2.3 Вибір програмних засобів реалізації системи моніторингу

Проєктування програмної частини системи моніторингу пацієнтів є не менш важливим етапом, ніж формування апаратної складової. Програмні засоби забезпечують взаємодію між сенсорними модулями, серверною інфраструктурою та інтерфейсом користувача. Саме вони визначають, наскільки ефективно відбуватиметься збір, передавання, обробка та візуалізація медичних даних у реальному часі. Вибір відповідних мов програмування, протоколів, бібліотек і фреймворків обумовлюється вимогами до продуктивності, швидкості реакції системи та стабільності. У межах даного проєкту програмна частина складається з трьох великих компонентів: програмного забезпечення сенсорного вузла (Arduino Nano ESP32 + бібліотеки для MAX30102 та BLE/MQTT-SN), серверного застосунку, який збирає дані від пристроїв, забезпечує маршрутизацію потоків та передає інформацію у WebSocket-стрім, клієнтського інтерфейсу, який у браузері відображає значення SpO_2 та частоти серцевих скорочень у вигляді інтерактивних графіків. Кожен із цих компонентів потребує окремого підходу та набору технологій.

2.3.1 Програмні засоби сенсорного вузла

Програмний модуль на основі Arduino Nano ESP32 виконує функції зчитування сирих фотоплетизмографічних (PPG) сигналів із MAX30102 через інтерфейс I²C, попередню цифрова обробку даних (усереднення, згладжування, класифікація піків), обчислення рівня SpO_2 та ЧСС або передавання сирих даних для серверної обробки, формування структурованих JSON-повідомлень, передавання даних на створений сервер через бездротову мережу.

Для програмування ESP32 обрано Arduino IDE, яка підтримує бібліотеку Wire.h для роботи з I²C, бібліотеки Adafruit MAX30102 / SparkFun MAX3010x, що забезпечують алгоритми обробки, стандартні інструменти для роботи з BLE та ESP-NOW, розширення для MQTT-SN, спрощеного протоколу телеметрії для систем із низьким споживанням. Причини вибору Arduino IDE та екосистеми Arduino це простота інтеграції з ESP32-платою, велика кількість тестованих бібліотек для роботи з MAX30102, швидка компіляція та завантаження прошивки через USB; кросплатформеність (Windows/Linux/MacOS), можливість швидкого експериментування з параметрами сенсорів. Завдяки цьому підхід значно скорочує час розробки та мінімізує технічні ризики. Алгоритми обробки сигналів на мікроконтролері Сенсор MAX30102 формує фотоплетизмографічні сигнали, які потребують фільтрації.

Програмне забезпечення ESP32 виконує цифрове згладжування (moving average), видалення шумів (DC removal, low-pass фільтр), виявлення піків для вимірювання ЧСС, розрахунок SpO_2 на основі співвідношення поглинання червоного та інфрачервоного світла. Алгоритми адаптовано під обмежені обчислювальні ресурси ESP32, що важливо для реального часу.

2.3.2 Серверні технології для приймання, обробки та маршрутизації даних

Центральна частина системи – серверний застосунок, що приймає дані від мережі сенсорів та передає їх клієнтам. Для створення серверної частини обрано Python та фреймворк FastAPI, що поєднує простоту декларування HTTP-ендпоінтів і підтримку асинхронного виконання завдань. Причинами вибору стали вбудлвана підтримка асинхронності, що дозволяє обробляти сотні

підключень, мінімальна затримка при роботі WebSocket-стрімів, висока швидкодія завдяки використанню Uvicorn/Starlette, здручне документування API завдяки OpenAPI, модульність та легкість інтеграції з IoT-протоколами.

Сервер виконує такі ключові функції як:

– прийом даних з MQTT-SN шлюзу – перетворення ESP32 пакетів у стандартні JSON-повідомлення;

– формування внутрішнього потоку даних – розселання показників одразу усім підключеним WebSocket-клієнтам;

– зберігає останні значення у кільцевому буфері – зберігання даних для користувачів, які щойно підключились;

– обслуговує діагностичні HTTP-ендпоінти – як приклад /probe_mqttsn для перевідки відправлення і обробки пакетів.

Протоколом для сенсорної мережі послугує MQTT-SN – полегшена версія MQTT, оптимізована для пристроїв із низьким енергоспоживанням, нестабільних мереж і Bluetooth-з’єднань. Основними його перевагами є підтримка коротких топіків («S1», «HR», «OX»), зменшений обсяг службових повідомлень, готовність до широкої мережі датчиків, швидка доставка телеметрії. У межах дослідження MQTT-SN використано як транспорт даних, який надходить на шлюз, а далі на сервер моніторингу.

2.3.3 Вебтехнології для інтерфейсу моніторингу

Клієнтську частину системи реалізовано з використанням стандартного стеку веброзробки: HTML5 [17] для структури сторінки, CSS3 [18] для стилізації та адаптивний інтерфейсу, JavaScript для логіки отримання даних через WebSocket, Chart.js для побудова графіків SpO_2 та ЧСС у реальному часі.

Роль WebSocket у візуалізації даних важлива в проекті через те, що він забезпечує сталий двоспрямований канал між браузером та сервером, майже нульову затримку при передачі даних, можливість обробляти сотні оновлень за секунду. На відміну від HTTP-запитів, які працюють «запит-відповідь», WebSocket дозволяє серверу активно надсилати нові дані у вигляді стріму. Це критично для медичних систем, де важлива точність та швидкість індикації.

Візуалізацію на основі Chart.js обрано через підтримку графіків часових рядів, плавність анімації, модульність, можливість додавання порогових ліній, високій продуктивності.

2.3.4 Формати даних, а також засоби тестування та діагностики

Уся система використовує єдиний формат обміну – JSON. Це забезпечує зручність інтеграції між Python, JavaScript та мікроконтролером ESP32, простоту логування та відлагодження, можливість розширення структури даних (рис. 2.5).

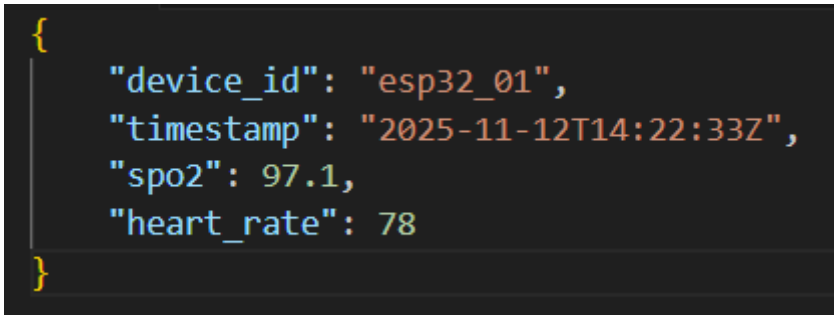

Рисунок 2.5 – Приклад структури пакета даних

Для відлагодження системи застосовано автоемулятор даних (на Python), що генерує псевдовипадкові значення SpO_2 та ЧСС, вбудований ендпоінт /probe_mqttsn, який дозволяє вручну відправити тестове повідомлення, логування MQTT-SN-трафіку, засоби браузера DevTools [19] для аналізу WebSocket-стріму, графічний моніторинг мережевих затримок. Ці інструменти дали змогу відтестувати систему навіть без фізичних сенсорів.

2.4 Порівняльний аналіз Wi-Fi та Bluetooth Low Energy як технологій передавання даних у медичних IoT-системах

У системах дистанційного моніторингу пацієнтів вибір технології зв’язку є одним із ключових рішень, оскільки саме канал передавання визначає стабільність отримання телеметрії, автономність сенсорних вузлів та можливість розгортання мережі у реальних умовах. Найчастіше для IoT-пристроїв застосовують дві технології: Wi-Fi та Bluetooth Low Energy (BLE). Обидві є

бездротовими та працюють у діапазоні 2,4 ГГц, але мають принципові відмінності, що впливають на архітектуру системи.

2.4.1 Характеристики Wi-Fi

Wi-Fi – це високошвидкісна мережна технологія, призначена для передавання великих обсягів даних. Її переваги це – велика пропускна здатність (десятки-сотні Мбіт/с), стабільність зв’язку на середніх відстанях 30-70 м, підтримка прямого підключення до серверів через локальну мережу або інтернет, широка сумісність з існуючими мережами медичних закладів.

Проте її недоліками слугують високе енергоспоживання, що робить Wi-Fi менш придатним для автономних сенсорів, підвищена зашумленість діапазону, що може впливати на стабільність у перевантажених середовищах (хвороби, реанімація), потрібна точка доступу або маршрутизатор, що ускладнює розгортання у польових умовах [20].

2.4.2 Характеристики Bluetooth Low Energy (BLE)

BLE розроблений спеціально для малопотужних сенсорних пристроїв, таких як медичні датчики, фітнес-браслети та мобільні монітори. Його використання обумовлюється надзвичайно низьким енергоспоживанням, підтримкою близької взаємодії між сенсором та приймачем, Mesh-мережам BLE 5.0, що дозволяють об’єднувати велику кількість сенсорів, спеціалізацією під медичні стандарти (HDP, IEEE 11073) [21].

Мінусами ж використання цієї технології є невеликий радіус дії (10-30 м), меншою швидкістю передедавання, що не підходить для великих потоків даних, потребує центрального пристрою для збору телеметрії.

2.4.3 Порівняння технологій та висновок

BLE можна використовувати в компактних фітнес-пристроях, але для медичного моніторингу мережевого типу, особливо у багатовузлових конфігураціях, він створює зайві ризики та суттєво ускладнює архітектуру.

Таким чином, у межах проєкту для передавання телеметрії SpO_2 та ЧСС було обрано Wi-Fi та MQTT-SN. Wi-Fi використано для серверної частини та Web-візуалізації (табл. 2.3).

Таблиця 2.3 – Порівняльна таблиця BLE та Wi-Fi

Параметр	Wi-Fi	BLE
Дальність дії	30-100 м	10-30 м (до 100 м у BLE 5.0 зі зниженням швидкості)
Стійкість каналу при багатьох вузлах (3-10)	Висока, без втрат	Низька через колізії та затримки
Швидкість передачі	До 600 Мбіт	До 2 Мбіт/с
Підтримка MQTT	Повна, нативна	Потрібен окремий шлюз
Сумісність із WebSocket та браузером	Пряма, без компромісів	Неможлива без gateway
Пропускна здатність при 1 Гц, 10 пристроїв	Високий запас	Можу бути нестабільною
Енергоспоживання	Високе	Дуже низьке
Простота розробки	Висока (MQTT, TCP, HTTP)	Низька (GATT, реклама, характеристики)
Масштабованість	Хороша	Обмежена
Придатність для медичної телеметрії	Чудова	Передбачає компроміси та ускладнення
Висновок	Стаціонарні сервери, системи з великим обсягом даних	Сенсорні IoT-пристрої, медичні датчики, портативні системи

2.5 Підсумок вибору

Проведений у розділі аналіз апаратних платформ, сенсорних модулів, методів обробки фізіологічних даних та технологій передавання дозволив сформувати оптимальний набір засобів для побудови системи моніторингу стану пацієнтів на основі мережі пульсоксиметрів. Порівняння мікроконтролерів показало, що саме Arduino Nano ESP32 є найбільш придатним для реалізації сенсорного вузла завдяки поєднанню обчислювальної потужності, енергоефективності та наявності вбудованого Wi-Fi -модуля.

Аналіз оптичних сенсорів підтвердив доцільність використання MAX30102, який забезпечує достатню точність та стабільність показників при компактних розмірах і низькому енергоспоживанні. Вивчення методів цифрової обробки сигналу дало підстави обрати варіант локального обчислення SpO_2 та ЧСС на мікроконтролері, що зменшує навантаження на мережу та підвищує швидкість реакції системи.

Серед технологій зв’язку порівняння Wi-Fi та Bluetooth Low Energy показало перевагу Wi-Fi у контексті автономних медичних IoT-пристроїв, де критичними є сумісність зі стандартами та можливість роботи у персональних або локальних мережах, висока пропускна спроміжність.

Для серверної частини найкращим рішенням виявився стек на основі Python та FastAPI, який дозволяє організувати швидкий прийом і обробку даних, а також підтримує WebSocket-з’єднання для передачі інформації у режимі реального часу. Протокол MQTT-SN у свою чергу забезпечує ефективний та легковаговий канал взаємодії між сенсорними вузлами та сервером.

Таким чином, комбінація ESP32, MAX30102, Wi-Fi, MQTT-SN та WebSocket формує збалансовану, масштабовану та технологічно оптимальну архітектуру, яка повністю відповідає вимогам побудови системи мережевого моніторингу пацієнтів у реальному часі та є придатною до подальшого розширення й практичного застосування.

РОЗДІЛ 3
РОЗРОБКА ОБ'ЄКТА ДОСЛІДЖЕННЯ

3.1 Архітектура та загальний принцип роботи системи моніторингу

Практична реалізація системи спостереження за пацієнтами базується на модульній архітектурі, яка об'єднує сенсорні вузли, комунікаційний прошарок та серверний компонент із веб-інтерфейсом для візуалізації даних у реальному часі.

Така структура дозволяє досягти гнучкості, масштабованості та адаптивності системи, що є критично важливим у телемедичних застосуваннях та системах медичного моніторингу.

Загальна архітектура системи представлена на рисунку 3.1. Вона передбачає циркуляцію інформації від фізичного сенсора до кінцевого користувача (лікаря чи оператора пункту моніторингу) у мінімально можливий час. Це забезпечує оперативне реагування на критичні зміни рівня насичення крові киснем (SpO_2) або частоти серцевих скорочень (ЧСС), що є одними з найважливіших життєвих параметрів.

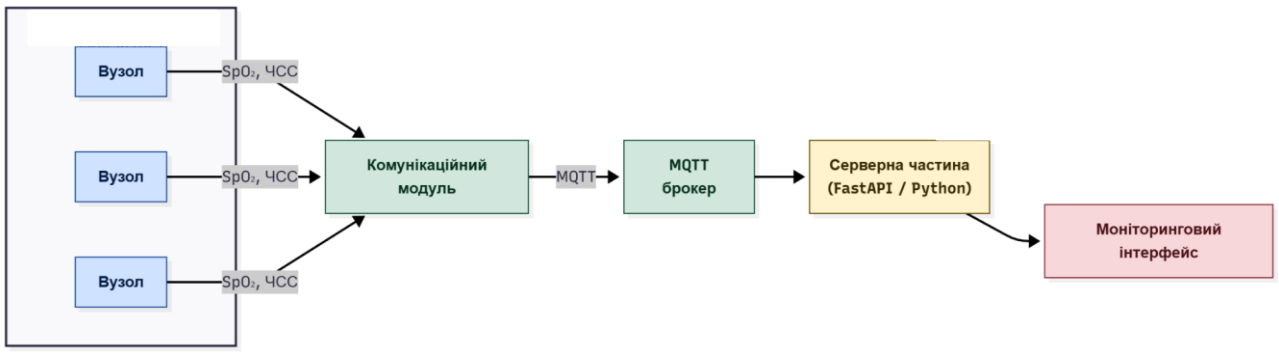

Рисунок 3.1 – Загальна архітектура системи спостереження за пацієнтами

Архітектура системи складається сенсорних вузлів, до складу яких входять мікроконтролери Arduino Nano ESP32 із під'єднаними сенсорами MAX30102. Кожен вузол реалізує функції зчитування червоного та інфрачервоного

PPG-сигналу, локальної фільтрації даних, розрахунку SpO_2 та ЧСС, формування структурованих JSON-пакетів, передавання вимірювань до сервера через протокол MQTT-SN. Система може включати один або десятки таких вузлів, що працюють паралельно, утворюючи мережу.

3.2 Реалізація веб-інтерфейсу моніторингу

Веб-інтерфейс у розробленій системі виконує роль клієнтської частини, яка підключається до MQTT-брокера через WebSocket, отримує телеметрію від пульсоксиметрів та візуалізує її у вигляді динамічних графіків. Реалізація інтерфейсу побудована на зв'язці JavaScript бібліотека Chart.js [22] браузерний MQTT-клієнт (mqtt.js). Весь код працює безпосередньо у браузері та не потребує встановлення додаткового програмного забезпечення на стороні користувача.

3.2.1 Ініціалізація параметрів і посилань на елементи інтерфейсу

На початку сценарію оголошуються глобальні константи та змінні, що визначають поведінку панелі моніторингу (рис. 3.2).

app.js > makeCard

```
// === налаштування ===
const MAX_POINTS = 600;          // ~10 хв при 1 Гц
const SPO2_THRESHOLD = 90;
const chartsRoot = document.getElementById('charts');
const $wsurl = document.getElementById('wsurl');
const $topic = document.getElementById('topic');
const $status = document.getElementById('status');
const $connect = document.getElementById('connectBtn');

let client = null;
const deviceCharts = new Map();  // device_id -> { chart, stateEl, statsEl, data, stats }

function setStatus(text, cls="") {
  $status.textContent = text;
  $status.className = `muted ${cls}`;
}

function fmtTime(d) {
  if (!d) return "–";
  try {
    const dt = (d instanceof Date) ? d : new Date(d);
    return dt.toLocaleTimeString([], { hour: '2-digit', minute:'2-digit', second:'2-digit' });
  } catch { return "–"; }
}
```

Рисунок 3.2 – Перша частина серверного коду

MAX_POINTS = 600 – максимальна кількість точок на графіку для одного пристрою. При частоті 1 Гц це приблизно 10 хвилин історії вимірювань. Після

досягнення цього ліміту найстаріші значення видаляються, що не дає графікам безкінечно розростатися.

SPO2_THRESHOLD = 90 – порогове значення насичення крові киснем. Якщо SpO_2 падає нижче цього рівня, інтерфейс відображає попередження.

Через document.getElementById зчитуються посилання на основні елементи DOM, а саме на контейнер для карток графіків chartsRoot, поля введення URL WebSocket ($wsurl) та теми MQTT ($topic), текстовий елемент статусу ($status), кнопка підключення ($connect). Окремо оголошується змінна client (об’єкт MQTT-клієнта), структура deviceCharts = new Map() (асоціативне сховище, де ключем є device_id), а значенням, відповідно, об’єкт з усією інформацією про відповідний графік, статистику та елементи інтерфейсу.

Функція setStatus(text, cls) відповідає за оновлення тексту та CSS-класу елемента статусу. Таким чином, одним викликом можна змінити і повідомлення, і колір (наприклад, зелений для «ok», червоний для «bad»). Допоміжна функція fmtTime(d) перетворює об’єкт Date (або рядок з датою) у зручний для користувача формат hh:mm:ss. Вона використовується для відображення часу, коли було зафіксоване мінімальне або максимальне значення SpO_2 або ЧСС.

3.2.2 Формування картки пристрою та підготовка графіка

Функція makeCard(deviceId) створює DOM-вузол , всередині якого розміщується заголовок з ідентифікатором пристрою та уточненням, що відобрає показники SpO_2 / ЧСС (рис. 3.3).

```
function makeCard(deviceId) {
  const card = document.createElement('div');
  card.className = 'card';
  card.innerHTML = `
    <div style="display:flex;justify-content:space-between;gap:8px;align-items:center;margin-bottom:8px;flex-wrap:wrap">
      <div><strong>${deviceId}</strong> <span class="muted">(SpO₂ / ЧСС)</span></div>
      <div class="muted" id="state-${deviceId}">...</div>
      <button id="reset-${deviceId}" class="reset-btn">Reset</button>
    </div>

    <div id="stats-${deviceId}" class="muted" style="display:grid;grid-template-columns:1fr 1fr;gap:6px;margin-bottom:8px">
      <div>SpO₂ min: <b id="spmin-${deviceId}">–</b> <span class="muted">@</span> <span id="sptmin-${deviceId}">–</span></div>
      <div>SpO₂ max: <b id="spmax-${deviceId}">–</b> <span class="muted">@</span> <span id="sptmax-${deviceId}">–</span></div>
      <div>HR min:   <b id="hrmin-${deviceId}">–</b> <span class="muted">@</span> <span id="hrtmin-${deviceId}">–</span></div>
      <div>HR max:   <b id="hrmax-${deviceId}">–</b> <span class="muted">@</span> <span id="hrtmax-${deviceId}">–</span></div>
    </div>

    <canvas id="cv-${deviceId}" height="260"></canvas>
  `;
```

Рисунок 3.3 – Друга частина серверного коду

Текстовий блок стану state-${deviceId}, де показується актуальний статус (норма, попередження, локальні міні/макс), кнопка Reset, яка дозволяє очистити історію даних для конкретного пристрою, блок «статистики», де вказано: SpO_2 min/max час, коли це значення було зафіксовано, HR (ЧСС) min/max відповідний час. <canvas id="cv-${deviceId}"> це елемент, у якому Chart.js буде малювати графік. Картка додається у головний контейнер chartsRoot і повертається як DOM-елемент.

Ця функція викликає makeCard(...), отримує контекст малювання із створеного <canvas> та ініціалізує дві часові серії для Chart.js, де перша серія – SpO_2 (%), прив'язана до осі y1, друга серія – ЧСС (уд/хв), прив'язана до осі y2 (рис. 3.4).

```
function createChartForDevice(deviceId) {
  const card = makeCard(deviceId);
  const ctx = card.querySelector(`#cv-${deviceId}`).getContext('2d');

  const data = {
    datasets: [
      {label:'SpO₂, %', yAxisID:'y1', data:[], pointRadius:0, borderWidth:2, tension:0.2},
      {label:'ЧСС, уд/хв', yAxisID:'y2', data:[], pointRadius:0, borderWidth:2, borderDash:[6,4], tension:0.2},
    ]
  };
```

Рисунок 3.4 – Функція createChartForDevice()

Параметри графіка налаштовані таким чином: по осі X використовується шкала часу (type: 'time', unit: 'second'), осі Y мають фіксовані діапазони SpO_2 80-100 %, ЧСС 40-180 уд/хв, де для SpO_2 використовується суцільна лінія, для ЧСС штрихова (borderDash), що полегшує візуальне розрізнення серій (рис. 3.5).

```
const chart = new Chart(ctx, {
  type: 'line',
  data,
  options: {
    responsive: true,
    animation: false,
    scales: {
      x: { type: 'time', time: { unit: 'second' }, grid: { color:'#1f2937' }, ticks:{ color:'#9ca3af' } },
      y1: { position:'left', min:80, max:100, grid:{ color:'#1f2937' }, ticks:{ color:'#9ca3af' }, title:{display:true, text:'SpO₂, %', color:'#9ca3af'} },
      y2: { position:'right', min:40, max:180, grid:{ drawOnChartArea:false }, ticks:{ color:'#9ca3af' }, title:{display:true, text:'ЧСС, уд/хв', color:'#9ca3af'} }
    },
    plugins: {
      legend: { labels:{ color:'#e6edf3' } }
    }
  }
});
```

Рисунок 3.5 – Налаштування параментрів змінної chart

Відключена анімація (animation: false) для забезпечення максимально швидкого оновлення графіків у режимі реального часу, палітра кольорів легенди та сітки підібрана під темну тему інтерфейсу.

Паралельно формується початкова структура статистики stats, де зберігатимуться глобальні мінімальні та максимальні значення SpO_2 і ЧСС, а також час їхнього виникнення (рис. 3.6).

```
const stats = {
  spMin: null, spMinT: null,
  spMax: null, spMaxT: null,
  hrMin: null, hrMinT: null,
  hrMax: null, hrMaxT: null,
};

const entry = {
  chart,
  stateEl: card.querySelector(`#state-${deviceId}`),
  statsEl:  card.querySelector(`#stats-${deviceId}`),
  data,
  stats,
  els: {
    spmin:  card.querySelector(`#spmin-${deviceId}`),
    sptmin: card.querySelector(`#sptmin-${deviceId}`),
    spmax:  card.querySelector(`#spmax-${deviceId}`),
    sptmax: card.querySelector(`#sptmax-${deviceId}`),
    hrmin:  card.querySelector(`#hrmin-${deviceId}`),
    hrtmin: card.querySelector(`#hrtmin-${deviceId}`),
    hrmax:  card.querySelector(`#hrmax-${deviceId}`),
    hrtmax: card.querySelector(`#hrtmax-${deviceId}`),
  }
};
```

Рисунок 3.6 – Налаштування параметрів констант stats та entry

Далі збирається об’єкт entry, який включає об’єкт chart (екземпляр Chart.js), елементи інтерфейсу для виведення статистики (spmin-…, sptmin-…, тощо), поточні статистичні дані, текстовий елемент стану (stateEl). Цей об’єкт

зберігається у deviceCharts з ключем deviceId, що дозволяє зручно звертатися до нього при надходженні кожного нового повідомлення.

3.2.3 Скидання даних для окремого пристрою

Кнопка Reset на кожній картці дозволяє очистити історію та статистику саме для цього пристрою, не впливаючи на інші (рис. 3.7).

```
// кнопка Reset — тільки для цього девайса
card.querySelector(`#reset-${deviceId}`).addEventListener('click', () => {
  entry.data.datasets[0].data.length = 0;
  entry.data.datasets[1].data.length = 0;
  entry.stats = {
    spMin: null, spMinT: null,
    spMax: null, spMaxT: null,
    hrMin: null, hrMinT: null,
    hrMax: null, hrMaxT: null,
  };
  // очистити UI
  entry.els.spmin.textContent  = '—';
  entry.els.sptmin.textContent = '—';
  entry.els.spmax.textContent  = '—';
  entry.els.sptmax.textContent = '—';
  entry.els.hrmin.textContent  = '—';
  entry.els.hrtmin.textContent = '—';
  entry.els.hrmax.textContent  = '—';
  entry.els.hrtmax.textContent = '—';
  entry.stateEl.textContent = 'reset done';
  entry.stateEl.className = 'muted';
  entry.chart.update('none');
});
```

Рисунок 3.7 – Опис функції «Reset»

Обробник події click для reset-${deviceId}:

– очищає масиви даних у двох серіях (datasets[0].data та datasets[1].data), скидає об’єкт stats до початкового стану (усі мін/макс = null);

– встановлює у всіх елементах відображення статистики символ « – »;

– змінює текст стану на reset done та скидає стилі;

– викликає chart.update('none') для перерисовки порожнього графіка.

Це особливо корисно при тестуванні, а також у випадках, коли потрібно розпочати нову сесію моніторингу для того ж пацієнта.

3.2.4 Оновлення статистики та виявлення локальних екстремумів

Окремий блок коду відповідає за аналіз вхідних даних і розрахунок статистичних характеристик (рис. 3.8).

```
function updateStats(entry, ts, spo2, hr) {
  const s = entry.stats;

  if (s.spMin === null || spo2 < s.spMin) { s.spMin = spo2; s.spMinT = ts; entry.els.spmin.textContent = spo2.toFixed(1); entry.els.sptmin.textContent = fmtTime(ts); }
  if (s.spMax === null || spo2 > s.spMax) { s.spMax = spo2; s.spMaxT = ts; entry.els.spmax.textContent = spo2.toFixed(1); entry.els.sptmax.textContent = fmtTime(ts); }

  if (s.hrMin === null || hr < s.hrMin)   { s.hrMin = hr;   s.hrMinT = ts; entry.els.hrmin.textContent = String(hr); entry.els.hrtmin.textContent = fmtTime(ts); }
  if (s.hrMax === null || hr > s.hrMax)   { s.hrMax = hr;   s.hrMaxT = ts; entry.els.hrmax.textContent = String(hr); entry.els.hrtmax.textContent = fmtTime(ts); }
}

// простий детектор локальних min/max (для короткого повідомлення у статусі)
function localExtrema(series) {
  if (series.length < 3) return null;
  const n = series.length;
  const a = series[n-3].y, b = series[n-2].y, c = series[n-1].y;
  if (b > a && b > c) return { type:'max', value:b };
  if (b < a && b < c) return { type:'min', value:b };
  return null;
}
```

Рисунок 3.8 – Опис функцій localExtrema та updateStats

Функція updateStats(entry, ts, spo2, hr) порівнює нові значення SpO_2 та ЧСС із поточними глобальними мінімумами/максимумами.

Якщо нове значення є першим або покращує попередній рекорд, оновлюються числове значення, час фіксації ts, текстові поля у інтерфейсі (spmin-…, sptmin-…, hrmax-… тощо). Таким чином, у кожного пристрою у будь-який момент видно крайні значення, які спостерігались за час роботи панелі.

Функція localExtrema(series) реалізує простий детектор локальних екстремумів. Вона бере три останні точки серії і аналізує середню:

– якщо b > a і b > c, вважається локальним максимумом;

– якщо b < a і b < c, вважається локальним мінімумом.

Результат використовується не для зберігання у статистиці, а для короткого текстового наголосу у статусі (наприклад, «SpO_2 MAX»), що дозволяє оператору швидше помітити локальні піки або падіння.

3.2.5 Обробка вхідних повідомлень з брокера

Ключовою функцією фронтенду є handleMessage(msg). Вона парсить JSON-рядок. Якщо формат некоректний, то функція просто завершується.

Також очікує, що в повідомленні є такі параметри як device_id (ідентифікатор сенсорного вузла), timestamp (час вимірювання), spo2 (значення насичення киснем), heart_rate (значення частоти серцевих скорочень). Якщо device_id ще не зустрічався, викликається createChartForDevice(dev) (рис. 3.9).

```
function handleMessage(msg) {
  // очікуємо JSON з device_id, spo2, heart_rate, timestamp
  let data;
  try { data = JSON.parse(msg); } catch { return; }
  const dev = data.device_id || 'unknown';
  const ts  = data.timestamp ? new Date(data.timestamp) : new Date();
  const spo2 = Number(data.spo2);
  const hr   = Number(data.heart_rate);
  if (!isFinite(spo2) || !isFinite(hr)) return;

  const entry = deviceCharts.get(dev) || createChartForDevice(dev);
  const { chart, stateEl } = entry;

  chart.data.datasets[0].data.push({ x: ts, y: spo2 });
  chart.data.datasets[1].data.push({ x: ts, y: hr });
  trim(chart.data.datasets[0].data);
  trim(chart.data.datasets[1].data);

  // оновити глобальні для девайса мін/макс з часу
  updateStats(entry, ts, spo2, hr);

  // простий alert: SpO2 нижче порогу
  if (spo2 < SPO2_THRESHOLD) {
    stateEl.textContent = `ALERT: SpO₂ ${spo2}% < ${SPO2_THRESHOLD}% • HR ${hr}`;
    stateEl.className = 'bad';
  } else {
    stateEl.textContent = `SpO₂ ${spo2}% • HR ${hr}`;
    stateEl.className = 'ok';
  }

  // локальна вершина/падина — короткий пінг у статусі
  const ex1 = localExtrema(chart.data.datasets[0].data);
  const ex2 = localExtrema(chart.data.datasets[1].data);
  if (ex1) stateEl.textContent += ` • SpO₂ ${ex1.type.toUpperCase()}: ${ex1.value.toFixed(1)}`;
  if (ex2) stateEl.textContent += ` • HR ${ex2.type.toUpperCase()}: ${ex2.value}`;

  chart.update('none');
}
```

Рисунок 3.9 – Опис функції HandleMessage()

Для нового вузла автоматично створюється картка, графік і структура статистики. До обох масивів даних додаються нові точки { x: ts, y: spo2 } та { x: ts, y: hr }.

Після цього викликається trim(...), яка видаляє найстарші точки, якщо їх кількість перевищує MAX_POINTS. Викликається updateStats(...) для оновлення глобальних мінімумів/максимумів (рис. 3.10).

```
function trim(arr) {
  while (arr.length > MAX_POINTS) arr.shift();
}
```

Рисунок 3.10 – Опис функції trim()

Виконується логіка попереджень, себто, якщо SpO_2 < SPO2_THRESHOLD, у полі стану виводиться червоним текстом повідомлення ALERT, де SpO_2 ... < поріг, інакше показується поточне значення SpO_2 та HR у «зеленому» стані.

За допомогою localExtrema(...) для кожної серії визначаються локальні піки або провали, про що додатково зазначається у статусному тексті. Наприкінці викликається chart.update('none') для миттєвого оновлення графіка без анімації. Таким чином, кожен новий пакет із сенсорів відразу трансформується у візуально зрозумілу інформацію з мінімально можливою затримкою.

3.2.6 Підключення до MQTT-брокера та керування станом з'єднання.

Функція connect() відповідає за встановлення й підтримку з'єднання з MQTT-брокером через WebSocket (рисунок 3.11).

```
function connect() {
  let url = ($wsurl.value || '').trim();
  let topic = ($topic.value || '').trim();
  if (!/^wss?:\/\//i.test(url)) url = 'ws://127.0.0.1:8083/mqtt';
  // автозаміна помилкового :1884 на :8083
  if (/^ws(s)?:\/\/[^/]+:1884\/mqtt$/i.test(url)) {
    url = url.replace(':1884', ':8083');
    $wsurl.value = url;
    console.warn('WS URL виправлено на', url);
  }
  if (!topic) topic = 'sensors/pulseox';

  try { client && client.end(true); } catch {}
```

Рисунок 3.11 – Перша частина функції connect()

Зчитуються значення з полів wsurl та topic. Якщо URL порожній або не починається з ws:// чи wss://, використовується значення за замовчуванням – ws://127.0.0.1:8083/mqtt. Реалізовано невеликий захист від помилки, коли користувач випадково вказує порт 1884 (UDP-порт шлюзу MQTT-SN) замість 8083 (WebSocket-порт). У такому випадку URL автоматично виправляється. Якщо тема не задана, за замовчуванням використовується

sensors/pulseox. Якщо попередній клієнт існує, він коректно закривається (client.end(true)).

У статусі відображається connecting…, після чого створюється MQTT-клієнт (рис. 3.12).

```
  setStatus('connecting…');
  client = mqtt.connect(url, {
    clientId: 'web_' + Math.random().toString(16).slice(2),
    reconnectPeriod: 1500,
    connectTimeout: 5000
  });

  client.on('connect', () => {
    setStatus('connected', 'ok');
    client.subscribe(topic, { qos: 0 }, (err) => {
      if (err) setStatus('sub error', 'bad');
      else setStatus(`subscribed: ${topic}`, 'ok');
    });
  });

  client.on('message', (_t, payload) => {
    handleMessage(payload.toString());
  });

  client.on('reconnect', () => setStatus('reconnecting…','warn'));
  client.on('close',     () => setStatus('closed','bad'));
  client.on('error',     (e) => setStatus(`error: ${e.message}`,'bad'));
}

document.getElementById('connectBtn').addEventListener('click', connect);
window.addEventListener('load', connect);
```

Рисунок 3.12 – Друга частина функції connect()

Випадково генерується clientId, що дозволяє відкривати декілька панелей моніторингу одночасно. reconnectPeriod задає інтервал автоматичних спроб перепідключення у разі обриву зв’язку. connectTimeout обмежує час очікування підключення.

Налаштовуються обробники подій:

– client.on('connect', ...) – при успішному підключенні статус змінюється на connected, виконується підписка на обрану тему topic з QoS = 0, у разі помилки підписки виводиться sub error;

– client.on('message', ...) – на кожне отримане MQTT-повідомлення викликається handleMessage(...), яка відповідає за розбір JSON та оновлення графіків;

– client.on('reconnect', ...) – статус змінюється на reconnecting…, що інформує користувача про тимчасові проблеми з мережею;

– client.on('close', ...) – при закритті з’єднання статус переходить у closed;

– client.on('error', ...) – у разі помилки в статус виводиться текст error: <опис>.

На завершення, до кнопки connectBtn прив’язано обробник, який дозволяє вручну перезапустити з’єднання, а подія window.load автоматично викликає connect() при завантаженні сторінки. Завдяки цьому оператор одразу бачить потік даних без додаткових дій.

3.3 Клієнтська частина

Цей підрозділ присвячено аналізу HTML-коду, який реалізує графічний інтерфейс моніторингу пульсоксиметрів у режимі реального часу. HTML-документ визначає структуру сторінки, підключає необхідні стилі та зовнішні бібліотеки, а також забезпечує взаємодію з JavaScript-модулем app.js, який уже було розглянуто в попередньому підпункті.

3.3.1 Опис блоку <head>

На рисунку 3.13 зображено частину блоку <head> в якому задається кодування UTF-8, назва сторінки, адаптивність під мобільні пристрої, пустий favicon, щоб уникнути помилок GET /favicon.ico 404. Це рішення дозволяє працювати повністю локально, без потреби в окремих ресурсах.

```
<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8" />
  <title>Pulse-ox Live Monitor</title>
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <link rel="icon" href="data:,">
```

Рисунок 3.13 – Перша частина блоку <head>

На рисунку 3.14 задаються параметри для візуального оформлення сторінки, а саме:

– темна кольорова тема, зручна для тривалої роботи;

– стилі для карток пристроїв;

– адаптивна сітка графіків (display: grid);

– стилізація кнопок та полів вводу, кольорові стани (ok, warn, bad).

```
<style>
  body{font-family:system-ui,Arial;margin:0;background:#0b1220;color:#e6edf3}
  header{padding:16px;border-bottom:1px solid #1f2937}
  main{max-width:1100px;margin:0 auto;padding:16px}
  .row{display:flex;gap:8px;flex-wrap:wrap}
  input,button{background:#0f172a;color:#e6edf3;border:1px solid #263247;border-radius:10px;padd
  input{min-width:280px}
  button{cursor:pointer}
  .grid{display:grid;grid-template-columns:repeat(auto-fill,minmax(360px,1fr));gap:16px;margin-top:16px
  .card{background:#111827;border:1px solid #1f2937;border-radius:14px;padding:12px}
  .muted{color:#9ca3af;font-size:12px}
  .ok{color:#22c55e}.warn{color:#f59e0b}.bad{color:#ef4444}
</style>
<script src="https://unpkg.com/mqtt/dist/mqtt.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js@4"></script>
<script src="https://cdn.jsdelivr.net/npm/chartjs-adapter-date-fns@3"></script>
</head>
```

Рисунок 3.14 – Друга частина блоку <head>

3.3.2 Опис блоку <body>

На рисунку 3.15 ми бачимо <header>, в якому реалізовані блоки для вибрання адреси WebSocket-брокера, задавання MQTT-топіка, кнопка ініціалізації підключення, відслідковування статусу з’єднання. Стани (offline, connecting…, connected) змінюються автоматично через JS.

```
<header>
  <div class="row" style="align-items:center">
    <strong>WS URL:</strong>
    <input id="wsurl" value="ws://127.0.0.1:8083/mqtt" />
    <strong>Topic:</strong>
    <input id="topic" value="sensors/pulseox" />
    <button id="connectBtn">Підключитись</button>
    <span id="status" class="muted">offline</span>
  </div>
</header>
```

Рисунок 3.15 – Опис <header>

Контейнер #charts – порожній при завантаженні сторінки (рис. 3.16).

```
<main>
  <p class="muted">Новий графік з'явиться автоматично, коли прийде повідомлення з новим <code>
  <div id="charts" class="grid"></div>
</main>

<script type="module" src="./app.js"></script>
```

Рисунок 3.16 – Опис блоку <main> та <script>

При появі нового device_id JavaScript автоматично створює картку пристрою. Кожен пристрій отримує власний графік та статистичний блок. Таким чином HTML підтримує необмежену кількість сенсорів одночасно. Використання type="module" дозволяє застосовувати сучасний синтаксис ES6, автоматично розбивати код на модулі, уникати глобальних змінних, легко масштабувати логіку інтерфейсу. JS-код повністю відповідає за обробку даних, побудову графіків та динамічну генерацію HTML-елементів.

3.4 Емулятор для відладки системи

Цей підрозділ присвячено аналізу програмного коду емулятора сенсорних вузлів, який використовується для генерації тестових даних пульсоксиметрів у режимі реального часу. Скрипт реалізований мовою Python і моделює роботу одного або множини пристроїв, що передають показники SpO_2 та частоти серцевих скорочень через протокол MQTT-SN.

3.4.1 Використані бібліотеки

На рисунку 3.17 зображується підлючення бібліотек.

```
import argparse, json, random, time, sys, threading
from datetime import datetime, timezone
from mqttsn.client import Client
```

Рисунок 3.17 – Бібліотеки використані в емуляторі

Були використані наступні бібліотеки:

– argparse для обробки аргументів командного рядка [23];

– json для серіалізації повідомлень перед відправкою [24];

– random використовується для генерації псевдовипадкових медичних даних [25];

– time для контролю інтервалів між публікаціями [26];

– sys необхідний для логування помилок у поток stderr [27];

– threading дозволяє запускати багато пристроїв паралельно [28];

– datetime створює точний UTC-штамп часу [29];

– mqttsn.client.Client для емуляції MQTT-SN клієнта, який виконує реальне підключення та публікацію [30].

Це створює основу для багатопоточного IoT-емулятора.

3.4.2 Створення CLI інтерфесу

На рисунку 3.18 блок коду створює повноцінний інтерфейс конфігурації емулятора. Користувач може задати IP шлюзу, порт, тему, ID одного пристрою або автоматичне створення кількох, частоту публікацій, режим відлагодження. Функція повертає готовий парсер, який використовується у main().

```
def make_parser():
    p = argparse.ArgumentParser(description="MQTT-SN multi-device emulator")
    p.add_argument("--host", default="127.0.0.1", help="EMQX MQTT-SN gateway IP")
    p.add_argument("--port", type=int, default=1884, help="EMQX MQTT-SN UDP port (default 1884)")
    p.add_argument("--topic", default="sensors/pulseox", help="довга тема або 2-симв. short (наприклад 'S
    p.add_argument("--device-id", default=None, help="один конкретний device_id; якщо задано – ігнорує --
    p.add_argument("--count", type=int, default=0, help="скільки пристроїв запустити (esp32_01..esp32_N)"
    p.add_argument("--prefix", default="esp32_", help="префікс для --count")
    p.add_argument("--rate", type=float, default=1.0, help="повідомлень/с")
    p.add_argument("--debug", type=int, default=0, help="1 = більше логів")
    return p
```

Рисунок 3.18 – CLI інтерфес

3.4.3 Функція run_one()

run_one() реалізує весь функціонал емулятора, а саме логування процесу з’єднання та емуляції, підключення, генерація фізіологічних параметрів.

На рисунку 3.19 бачимо початок лістинг функції run_one(). use_short – якщо довжина теми 2 символи, це short topic MQTT-SN, відповідно, треба буде публікувати з topicType=2. period – інтервал між публікаціями. Наприклад: rate_hz = 1,0, це period = 1 с, rate_hz = 2,0, це period = 0,5 с. max(rate_hz, 0,01) захищає від ділення на нуль.

```python
def run_one(host, port, topic, device_id, rate_hz, debug):
    use_short = len(topic) == 2
    period = 1.0 / max(rate_hz, 0.01)

    def log(*a):
        if debug: print(f"[{device_id}]", *a, file=sys.stderr, flush=True)

    def connect_with_retry():
        attempts = 0
        while True:
            attempts += 1
            try:
                log(f"[DBG] connecting to {host}:{port} (attempt {attempts})")
                cli = Client(client_id=device_id.encode("ascii", "ignore"), host=host, port=port)
                cli.connect()
                log("[DBG] connected")
                return cli
            except Exception as e:
                print(f"[ERROR] connect failed ({device_id}): {type(e).__name__}: {e}", flush=True)
                time.sleep(1.5)
```

Рисунок 3.19 – Логування та підключення

log це функція логування. Якщо --debug 1, усі внутрішні діагностичні повідомлення друкуються у stderr з префіксом [device_id]. Якщо debug=0, логер мовчить і консоль не засмічується.

connect_with_retry() відповідає за підключення до MQTT-SN з автоматичними ретраями. Цикл while True – намагаємось підключитися доти, доки не вийде. Client(...) – створюємо MQTT-SN клієнт: client_id береться з device_id, кодується в ASCII. cli.connect() – встановлює UDP-сесію до шлюзу. Якщо все добре то лог виглядатиме наступним чином – «[DBG] connected» у debug-режимі. повертаємо cli. Якщо видає помилку, то логувальник пише «[ERROR] connect failed (device_id)» і емулятор намагається підключитись ще раз. Це робить його стійким. Якщо брокер впав або перезапускається, скрипт не вмирає, а терпляче чекає відновлення.

На рисунку 3.20 можемо побачити реєстрацію топіків для підключення. MQTT-SN для довгих тем зазвичай вимагає REGISTER перед PUBLISH. Якщо використовується коротка тема – нічого не робимо. Якщо тема довга: пробуємо cli.register(topic). У разі помилки просто логимо й ігноруємо (багато реалізацій працюють і без явного REGISTER).

```
def ensure_topic(cli):
    if use_short:
        return
    try:
        log(f"[DBG] registering topic '{topic}'")
        cli.register(topic)
    except Exception as e:
        log(f"[DBG] register failed (ignored): {e}")

rng = random.Random(device_id)
spo2 = rng.uniform(96.0, 98.5)
hr   = rng.uniform(65.0, 85.0)

print(f"[INIT] {device_id}: MQTT-SN → {host}:{port}  topic={topic!r} (short={use_short})", flush=True
cli = connect_with_retry()
ensure_topic(cli)
```

Рисунок 3.20 – Реєстрація теми для довгих топіків, початкові значення та генератор параметрів

rng = random.Random(device_id) – ініціалізація генератора з seed, залежним від device_id. Кожен віртуальний «пацієнт» має свою унікальну, але повторювану траєкторію значень. Початкові spo2 та hr виставлені у фізіологічно адекватні діапазони, а саме SpO_2 близько 96-98,5 %, а ЧСС близько 65-85 уд/хв. Виводимо ініціальний лог з інформацією про те який це device_id, куди буде відбуватись підключення, яка тема, short=True/False. Далі cli = connect_with_retry() – отримує живий MQTT-SN клієнт, а ensure_topic(cli) реєструє тему, якщо треба.

Цей блок коду визначає, скільки віртуальних пристроїв запускати, як їх ідентифікувати, у скількох потоках їх запускати та як керувати життєвим циклом цих потоків (рис. 3.21). Першочегово викликається вищеописаний парсер, який розбирає всі аргументи передані користувачем, та розміщає їх в змінну args. Далі створюється масив потоків. Це потрібно для того щоб запускати кожен пристрій в окремому потоці, тому зберігаємо об’єкти потоків у цьому списку, щоб пізніше стежити за їхнім станом.

Надалі, на рисунку 3.21 описано три варіанти запуску емулятора. Перший режим, це запуск одного конкретного пристрою. Відповідно якщо користувач задав конкретний device_id, тоді запускатиметься один віртуальний сенсор. Другий режим, це запуск декількох пристроїв. В такому випадку користувач

вказує під час відладки кількість девайсів і їхні айді, або тільки айді, або тільки кількість. У випадку вказування кількості, автоматично генеруються назви по шаблону – «esp32_[номер]». Як і описувалось раніше, запускаються вони в окремому потоку. Варіант запуску без параметрів створює один емульований пристрій зі згенерованою назвою.

```
def main():
    args = make_parser().parse_args()

    threads = []
    if args.device_id:
        t = threading.Thread(target=run_one, args=(args.host, args.port, args.topic, args.device_id, args.rate, args.debug), daemon=True)
        t.start(); threads.append(t)
    elif args.count and args.count > 0:
        for i in range(1, args.count + 1):
            dev_id = f"{args.prefix}{i:02d}"
            t = threading.Thread(target=run_one, args=(args.host, args.port, args.topic, dev_id, args.rate, args.debug), daemon=True)
            t.start(); threads.append(t)
    else:
        dev_id = "esp32_emul_01"
        t = threading.Thread(target=run_one, args=(args.host, args.port, args.topic, dev_id, args.rate, args.debug), daemon=True)
        t.start(); threads.append(t)

    try:
        while any(t.is_alive() for t in threads):
            time.sleep(0.5)
    except KeyboardInterrupt:
        pass

if __name__ == "__main__":
    main()
```

Рисунок 3.21 – Функція main()

І остання частина лістингу, це логіка перевірки справності підключення та контролювання статусу роботи всієї системи. По суті, реалізовано нескладний механізм «пульсації» контролю: кожні 0,5 секунди виконується перевірка активності хоча б одного з потоків, що відповідають за отримання чи обробку даних. Якщо хоча б один із них продовжує працювати, головний процес вважає систему активною та переходить до наступної ітерації перевірки.

3.5 Створення макету пульсоксиметра

Створення макетного сенсорного вузла пульсоксиметра стало основним практичним етапом розробки системи моніторингу, оскільки саме цей елемент забезпечує отримання та передачу фізіологічних даних у реальному часі. У межах роботи було зібрано повністю функціональний макет на базі мікроконтролера ESP32 та сенсорного модуля MAX30102. Його було

реалізовано на макетній платі (breadboard), що дозволило швидко перевіряти різні конфігурації підключень, живлення та розміщення елементів.

3.5.1 Створення апаратного макету сенсорного вузла

Основою апаратної частини слугував мікроконтролер ESP32, який забезпечує обчислення, керування периферією та бездротовий зв'язок. Для зчитування показників рівня насичення крові киснем і частоти серцевих скорочень використовувався модуль MAX30102, що реалізує фотоплетизмографічний метод. Підключення сенсора до контролера здійснювалося через інтерфейс I²C: лінія SDA була під'єднана до GPIO21, а SCL до GPIO22. Важливою умовою було дотримання напруги живлення є те, що MAX30102 працює виключно від 3,3 В, тому його лінія VCC була під'єднана саме до стабілізованого виходу ESP32, що запобігло можливому пошкодженню модулю.

Оскільки сенсорний вузол планувався як прототип портативного пристрою, значну увагу приділено організації живлення. Макет працював не тільки від USB-порту, але й від літій-полімерного акумулятора (Li-Po) [31] на 3,7 В. Для заряджання та керування акумулятором застосовувався контролер TP4056 з інтегрованим захистом, що забезпечував стабільність напруги та захист від перезаряду. Для підвищення безпеки в живлення було включено поліфуз (self-resettable fuse), який автоматично відновлює роботу після надструму. Така конфігурація дозволила забезпечити захист від короткого замикання під час експериментів на макетній платі, що є критично важливим під час побудови прототипів.

Після апаратного збирання макет був підготовлений до тестування. Першим етапом стала перевірка працездатності MAX30102 і правильності його підключення. Стандартним способом діагностики є аналіз значень інфрачервоного (IR) каналу: якщо модуль фіксує присутність пальця, інтенсивність сигналу суттєво зростає. Це підтверджує як коректність підключення сенсора, так і його здатність фіксувати PPG-дані. У процесі

тестування було встановлено, що навіть на макетній платі з довгими дротами сенсор працює стабільно, а з'єднання по I²C не дає збоїв.

Конфігурація роботи MAX30102 включала налаштування сили струму світлодіодів, частоти дискретизації та параметрів внутрішніх фільтрів. Ці параметри впливають на форму PPG-хвилі й точність розрахунків SpO_2 та частоти серцевих скорочень. У ході налаштування було підібрано такі значення, що забезпечували чіткі та стабільні показники навіть при легких рухах пальця. Через те, що макет зібраний на breadboard, певні електромагнітні шуми все ж спостерігались, однак вони не впливали критично на результат і були частково компенсовані внутрішніми фільтрами MAX30102.

Далі сенсорний вузол переходив до формування телеметрії. Зібрані значення SpO_2 і ЧСС разом з часовою міткою та ідентифікатором пристрою формувалися в JSON-повідомлення. У нашому прототипі кожний вузол має власний device_id, що дає змогу відслідковувати кілька пристроїв одночасно і відображати їхні дані в окремих графіках у вебінтерфейсі системи моніторингу. Передавання даних здійснювалося через бездротовий модуль ESP32 відповідно до протоколу MQTT-SN, який оптимізований для мереж з обмеженими ресурсами.

Після цього розпочалося комплексне тестування макету. Було перевірено роботу пристрою при різних рівнях заряду акумулятора, а також під час переходу з USB-живлення на Li-Po та навпаки. Завдяки контролеру TP4056 та поліфузу макет працював стабільно та безпечо навіть при навмисному моделюванні коротких замикань на окремих ділянках схеми. Також були протестовані різні положення пальця, різна сила натиснення і вплив зовнішнього освітлення. Усі ці експерименти підтвердили, що сенсор достатньо чутливий та стабільний, щоб використовувати його у багатосенсорній IoT-системі.

Таким чином, зібраний на макетній платі сенсорний вузол пульсоксиметра довів свою працездатність у реальних умовах. Обрана апаратна конфігурація забезпечує безпечну роботу, стабільне живлення, надійне зчитування сигналів та можливість передачі даних у реальному часі. Отримані результати

підтверджують, що така архітектура є ефективною та може бути масштабована у повноцінну систему моніторингу пацієнтів.

3.5.2 Програмна реалізація сенсорного вузла

На рисунку 3.22 фрагмент коду підключає всі необхідні бібліотеки для роботи сенсорного модуля.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include <time.h>
```

Рисунок 3.22 – Підключення бібліотек в Arduino IDE

WiFi.h відповідає за роботу ESP32 у режимі клієнта Wi-Fi-мережі [32]. PubSubClient.h забезпечує публікацію даних через протокол MQTT, що використовується як транспорт для передачі телеметрії [33]. Бібліотека ArduinoJson.h потрібна для формування компактних JSON-повідомлень перед відправкою на сервер [34]. Заголовок Wire.h вмикає інтерфейс I²C, через який ESP32 взаємодіє з оптичним сенсором MAX30102 [35]. У свою чергу, MAX30105.h містить драйвери для читання IR та RED сигналів, а heartRate.h – алгоритми оцінки частоти серцевих скорочень на основі коливань PPG-сигналу [36]. Файл time.h використовується для синхронізації часу за допомогою NTP, що дозволяє додавати до показників коректні часові мітки [37].

У цьому блоці оголошуються основні параметри конфігурації всієї системи. Задаються мережеві реквізити Wi-Fi: назва точки доступу та пароль, за допомогою яких ESP32 встановлює бездротове з’єднання. Далі визначається IP-адреса брокера MQTT та порт, що використовується для передачі телеметричних даних. Строкова константа MQTT_TOPIC задає тему, в яку будуть публікуватися виміряні значення. Змінна DEVICE_ID використовується як унікальний ідентифікатор сенсорного вузла в мережі. Параметр PUBLISH_RATE_HZ відповідає за частоту оновлення – у даному випадку сенсор

генерує й надсилає одне повідомлення на секунду. Також визначається використання NTP-сервера для синхронізації часу, а змінні GMT_OFFSET_SEC та DAYLIGHT_OFFSET_SEC задають часовий зсув та умови переходу на літній час. Це дозволяє формувати коректні часові мітки у форматі UTC у кожному JSON-повідомленні (рис. 3.23).

```
const char* WIFI_SSID     = "TP-Link_6840_5G";
const char* WIFI_PASSWORD = "vlad1709";
const char* MQTT_HOST     = "192.168.0.110";
const uint16_t MQTT_PORT  = 1883;
const char* MQTT_TOPIC    = "sensors/pulseox";
const char* DEVICE_ID     = "esp32_node_01";
const float PUBLISH_RATE_HZ = 1.0f;
const char* NTP_SERVER = "pool.ntp.org";
const long GMT_OFFSET_SEC = 0;
const int DAYLIGHT_OFFSET_SEC = 0;
```

Рисунок 3.23 – Налаштування проекту

Тут створюються глобальні об’єкти, необхідні для роботи основних модулів. Об’єкт espClient забезпечує TCP-з’єднання для MQTT-клієнта, а PubSubClient mqtt використовує його для передачі повідомлень. Далі оголошується об’єкт sensor – драйвер сенсора MAX30102/30105, який зчитує інфрачервоні та червоні компоненти PPG-сигналу. Змінні irValue та redValue слугують буферами для сирих сигналів. Значення spo2Val і bpmVal ініціалізуються середніми фізіологічними показниками, щоб уникнути «порожніх» даних у перші моменти роботи пристрою (рис. 3.24).

```
WiFiClient espClient;
PubSubClient mqtt(espClient);

MAX30105 sensor;
long irValue = 0;
long redValue = 0;

float spo2Val = 97.0f;
float bpmVal  = 75.0f;

void connectWiFi() {
  Serial.print("[WiFi] Connecting to ");
  Serial.println(WIFI_SSID);

  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  while (WiFi.status() != WL_CONNECTED) {
    delay(400);
    Serial.print(".");
  }

  Serial.println();
  Serial.print("[WiFi] Connected, IP: ");
  Serial.println(WiFi.localIP());
}
```

Рисунок 3.24 – Створення глобальний об’єктів та підключення до Wi-Fi мережі

Функція connectWifi відповідає за встановлення Wi-Fi-з’єднання. У режимі STA (station mode) ESP32 підключається до вказаної точки доступу. Поки з’єднання не встановлено, пристрій періодично перевіряє статус і виводить у консоль індикатор прогресу. Після успішного підключення контролер виводить отриману локальну IP-адресу, яка підтверджує готовність до роботи в мережі. Такий спосіб підключення є стандартним для більшості IoT-пристроїв і забезпечує необхідну стабільність перед початком обміну даними.

Функція setupTime() виконує початкове налаштування системного часу через NTP (рис. 3.25). Використовується стандартна для ESP32 функція configTime(), яка автоматично звертається до заданого NTP-сервера та синхронізує внутрішній RTC-модуль мікроконтролера. Синхронізація часу необхідна для формування коректних часових міток у форматі ISO-8601, що дозволяє серверу однозначно визначати порядок надходження даних, будувати часові графіки та уникати проблем із локальними зонами (особливо коли кілька сенсорів працюють паралельно). Простий delay() дає мікроконтролеру час на завершення процедури ініціалізації.

```
void setupTime() {
  configTime(GMT_OFFSET_SEC, DAYLIGHT_OFFSET_SEC, NTP_SERVER);
  Serial.println("[TIME] syncing...");
  delay(1000);
}

String makeIsoTimestamp() {
  struct tm timeinfo;
  if (getLocalTime(&timeinfo)) {
    char buf[32];
    strftime(buf, sizeof(buf), "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
    return String(buf);
  } else {
    return "1970-01-01T00:00:00Z";
  }
}
```

Рисунок 3.25 – Налаштування системного часу та генерація часової мітки

Функція генерує часову мітку у стандарті ISO-8601. Спочатку виконується запит поточного часу з внутрішнього RTC-буфера. Якщо система вже синхронізована з NTP, мітка формується як справжній UTC-час. Якщо ж NTP недоступний (наприклад, при поганому Wi-Fi), функція повертає базове значення «1970-01-01…», що сигналізує про відсутність часу. Сам формат ISO-8601 обраний не випадково, бо саме в такому вигляді дані очікує серверна

частина та веб-інтерфейс, який автоматично перетворює строку у часові об’єкти для побудови графіків.

На рисунку 3.26 функція встановлює з’єднання з MQTT-брокером. Поки підключення не встановлено, пристрій повторює спроби з інтервалом у 2 секунди. Особливість реалізації полягає у використанні MAC-адреси ESP32 як частини clientId, що гарантує унікальність клієнта у мережі навіть при наявності кількох однакових сенсорних вузлів. Таким чином broker не відкидає повторні сесії та не створює конфліктів ідентифікаторів. Функція також виводить діагностичні повідомлення, які допомагають у налагодженні мережової частини.

```
void connectMQTT() {
  while (!mqtt.connected()) {
    Serial.println("[MQTT] Connecting...");

    String clientId = "esp32_";
    clientId += String((uint32_t)ESP.getEfuseMac(), HEX);

    if (mqtt.connect(clientId.c_str())) {
      Serial.println("[MQTT] Connected");
    } else {
      Serial.print("[MQTT] Failed, code=");
      Serial.println(mqtt.state());
      delay(2000);
    }
  }
}
```

Рисунок 3.26 – Під’єднання до MQTT-брокера

У наступному фрагменті зчитуються реальні фізіологічні сигнали із сенсора MAX30102 – інфрачервоний (IR) та червоний (RED) компоненти PPG-сигналу. Перше ж умовне правило – якщо IR < 5000, сенсор погано торкається пальця або сигнал перерваний. У цьому випадку вимірювання вважаються недійсними, і обидві величини позначаються як NAN (рис. 3.27).

```
void readRealVitals() {
  irValue = sensor.getIR();
  redValue = sensor.getRed();

  if (irValue < 5000) {
    spo2Val = NAN;
    bpmVal  = NAN;
    return;
  }

  bpmVal = heartRate.getBeatsPerMinute();

 if (bpmVal < 40 || bpmVal > 180) bpmVal = NAN;

  float ratio = (float)redValue / (float)irValue;
  spo2Val = 110.0f - 25.0f * ratio;

  if (spo2Val < 80 || spo2Val > 100) spo2Val = NAN;
}
```

Рисунок 3.27 – Зчитування фізіологічних показників

ЧСС визначається через алгоритм beat-to-beat, який входить до бібліотеки heartRate.h. Застосовується груба фільтрація: значення поза межами фізіологічної норми відкидаються.

SpO_2 оцінюється за емпіричною формулою, яка використовує співвідношення між червоним та інфрачервоним каналами – такий метод часто застосовується у простих реалізаціях. Після розрахунку знову застосовуються межові перевірки для відсікання шумів та некоректних вимірів. Це дозволяє отримати стабільні значення для подальшої передачі.

Частина зображена на рисунку 3.28 формує готове MQTT-повідомлення у форматі JSON. Поля spo2 та heart_rate можуть містити або значення, або null, якщо вимір був недійсним, це дозволяє серверу коректно обробляти такі випадки. Значення округлюються, щоб уникнути зайвих коливань на графіку та передавати тільки суттєві зміни. Після серіалізації JSON у текстовий буфер виконується публікація у вказану тему MQTT. Вивід у консоль дозволяє контролювати роботу сенсорного вузла в реальному часі.

```
void publishVitals() {
  readRealVitals();

  StaticJsonDocument<256> doc;
  doc["device_id"]  = DEVICE_ID;
  doc["timestamp"]  = makeIsoTimestamp();

  if (isnan(spo2Val)) doc["spo2"] = nullptr;
  else doc["spo2"] = (float)roundf(spo2Val * 10) / 10.0;

  if (isnan(bpmVal)) doc["heart_rate"] = nullptr;
  else doc["heart_rate"] = (int)roundf(bpmVal);

  char buffer[256];
  size_t len = serializeJson(doc, buffer);

  bool ok = mqtt.publish(MQTT_TOPIC, buffer, len);
  Serial.print("[PUB] ");
  Serial.println(buffer);
}
```

Рисунок 3.28 – Формування повідомлення

У функції setup() виконується ініціалізація всього обладнання. Починається з відкриття серіального порту для діагностики. Далі підключення до Wi-Fi та NTP. Ініціалізується MQTT-клієнт, але з’єднання буде встановлене лише у циклі loop(). Модуль MAX30102 підключений за інтерфейсом I²C на пінах 21 та 22.

Перевіряється коректність запуску сенсора – якщо пристрій не знайдений, мікроконтролер зупиняє роботу, що запобігає передачі порожніх або некоректних даних. Після успішного старту конфігуруються параметри світлодіодів та FIFO-буфера (через enableAFULL()) для стабільності сигналу (рис. 3.29).

Цей фрагмент відповідає за регулярне виконання основного функціоналу пристрою. Постійно контролюється стан Wi-Fi і MQTT – при розриві зв’язку контролер автоматично перепідключається. Внутрішній таймер контролює частоту публікацій: раз на секунду викликається publishVitals(). Такий підхід гарантує стабільний потік даних у реальному часі та адаптивність до мережевих збоїв (рис. 3.30).

```
void setup() {
  Serial.begin(115200);
  delay(1000);

  Serial.println("=== ESP32 MAX30102 PulseOx ===");

  connectWiFi();
  setupTime();

  mqtt.setServer(MQTT_HOST, MQTT_PORT);

  Wire.begin(21, 22);  // SDA=21, SCL=22
  if (!sensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println("[MAX30102] FAILED to initialize!");
    while (1);
  }

  sensor.setup();
  sensor.setPulseAmplitudeRed(0x1F);
  sensor.setPulseAmplitudeIR(0x1F);
  sensor.enableAFULL();
}
```

Рисунок 3.29 – Ініціалізація обладнання

```
void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();
  }

  if (!mqtt.connected()) {
    connectMQTT();
  }
  mqtt.loop();

  unsigned long now = millis();
  if (now - lastMs >= intervalMs) {
    lastMs = now;
    publishVitals();
  }
}
```

Рисунок 3.30 – Виконання основного функціоналу пристрою

3.6 Тестування проектної частини

Тестування розробленої кіберфізичної системи пульсоксиметричного моніторингу проводилося з метою перевірки коректності роботи всіх її складових: серверної частини, веб-інтерфейсу та сенсорного вузла на базі ESP32 з датчиком MAX30102. Послідовно перевірялася передача даних між компонентами, стабільність роботи MQTT-брокера, реакція інтерфейсу на появу нових пристроїв, правильність візуалізації телеметрії, обчислення статистичних параметрів і робота системи оповіщення.

Далі у відповідних підпунктах наведено результати тестування кожного етапу окремо: спочатку емулятора пульсоксиметрів, потім взаємодії веб-клієнта з брокером у реальному часі, і нарешті – перевірка роботи реального сенсорного макета на макетній платі.

3.6.1 Тестування серверної частини з емулятором

Для підтвердження стабільності MQTT-інфраструктури та коректності маршрутизації повідомлень було проведено тестування серверної частини системи за допомогою Python-емулятора багатоканальних пульсоксиметрів. Під час тестування виконувалася повна емуляція даних SpO_2 та частоти серцевих скорочень з декількох віртуальних сенсорних вузлів (рис. 3.31).

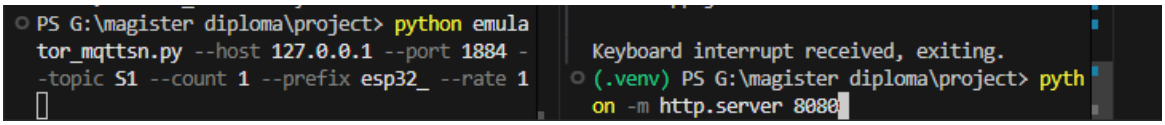

Рисунок 3.31 – Команди для запуску серверу та емулятора

Емулятор створював два незалежні пристрої (esp32_01, esp32_02), які раз на секунду передавали телеметрію у брокер EMQX [38] (рис. 3.31). Структура повідомлень включала device_id, SpO_2, ЧСС та часову мітку ISO-8601.

Для роботи вебінтерфейсу локально піднімався легкий HTTP-сервер. Після цього сторінка мониторингу відкривалася за адресою

http://127.0.0.1:8080/index.html , а підключення до брокера виконувалося через WebSocket-ендпоінт ws://127.0.0.1:8083/mqtt.

У процесі тестування було зроблено кілька скріншотів панелі лістенерів EMQX, де відображалися статуси UDP-та WebSocket-підключень, кількість активних MQTT-клієнтів та реальний потік вхідних повідомлень (рис. 3.32).

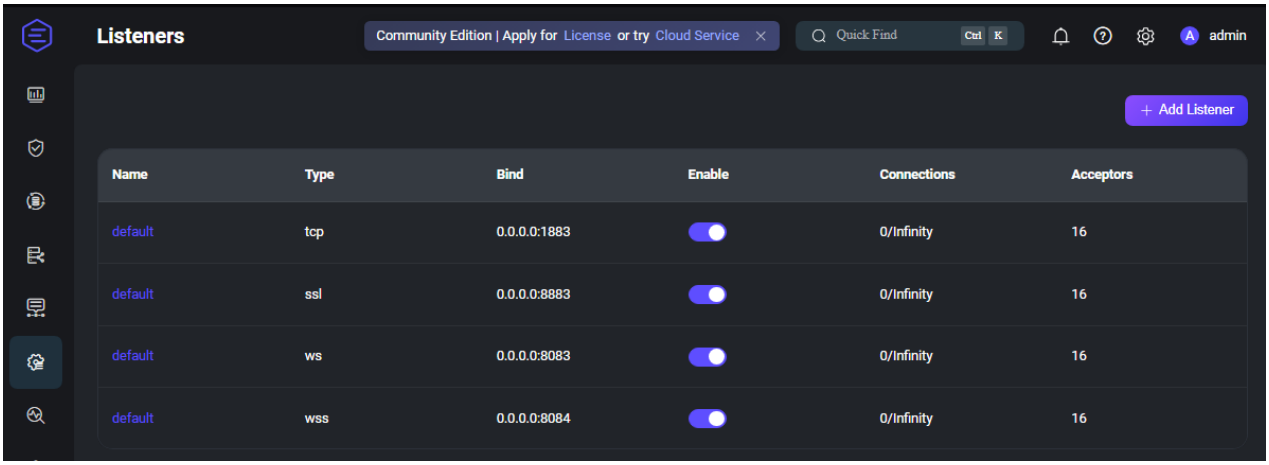

Рисунок 3.32 – Вкладка Listeners в MQTT dashboard

Окремо зафіксовано роботу вебінтерфейсу – на графіках у режимі реального часу відображалися криві зміни SpO_2 та ЧСС для кожного віртуального сенсора.

Також було перевірено автоматичне створення окремих карток при появі нових device_id, роботу лімітатора історії (MAX_POINTS) та оновлення статистики мінімумів і максимумів.

За результатами випробувань підтверджено, що серверна частина коректно приймає та обробляє MQTT-повідомлення, а вебінтерфейс стабільно візуалізує телеметрію в режимі реального часу без втрати даних. Отримані скріншоти були додані як візуальна валідація працездатності розробленої системи (рис. 3.33).

Рисунок 3.33 – Два емульованих графа

3.6.2 Тестування макету

Після перевірки роботи серверної частини та емулятора було проведено тестування системи у конфігурації, максимально наближеній до реального використання сенсорного вузла. Макет пульсоксиметра був зібраний на макетній платі та складався з плати ESP32, сенсора MAX30102, LiPo-акумулятора, модуля зарядки TP4056 та поліфюза, які забезпечували автономне живлення та базовий захист схеми. З'єднання сенсора з контролером виконувалося через інтерфейс I²C (лінії SDA та SCL), що відповідає рекомендованій схемі для MAX30102.

У межах перевірки було запущено веб-інтерфейс та згенеровано потік даних, який надходив до брокера з ідентифікатором esp32_node_01. Графічний інтерфейс коректно розпізнав новий вузол та створив окремий графік для відображення двох параметрів – SpO_2 та частоти серцевих скорочень. Поведінка інтерфейсу відповідала попереднім тестам з емулятором: графіки оновлювалися в режимі реального часу, відстежувалися локальні мінімальні та максимальні значення, а також працювала індикація стану вузла (рис. 3.34).

Рисунок 3.34 – Граф макета

Таким чином, тестування показало, що серверна частина та клієнтський моніторинг здатні працювати з апаратним вузлом без необхідності модифікації протоколу обміну даними чи логіки обробки повідомлень. Згенеровані макетом значення коректно сприймаються системою, а сам механізм передачі через MQTT підтвердив стабільність роботи.

3.7 Експериментальна частина

Для оцінки стійкості системи до зовнішнього освітлення під час експериментів додатково фіксувався рівень освітленості в зоні сенсора (в одиницях освітленості lux). Це дозволило порівняти роботу датчика MAX30102 у кількох характерних сценаріях: від повного перекриття зовнішнього світла до умов сильного освітлення. У кожному випадку до сенсора прикладався палець, а зверху додавалася або не додавалася тканина різної щільності.

Перший експеримент проводився при мінімальному рівні зовнішньої засвітки 0-5 lux, що відповідає щільному оптичному перекриттю пальця

непрозорою тканиною. У таких умовах зовнішнє світло практично не потрапляє до фотодіода. Отримані графіки демонструють стабільні показники SpO_2 та рівну, малошумну хвилю ЧСС (рис. 3.35).

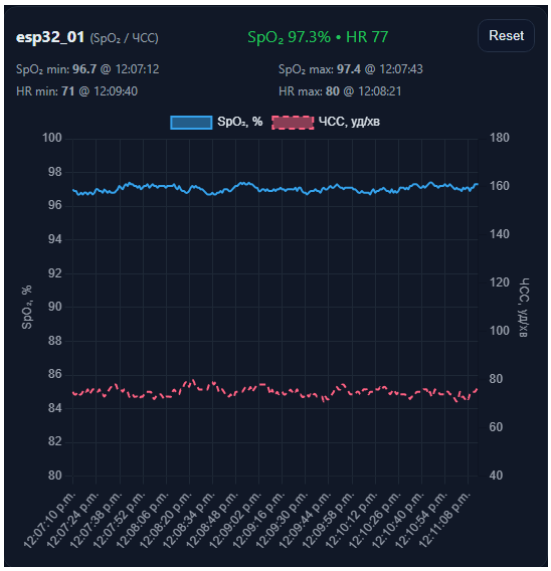

Рисунок 3.35 – Граф пульсоксометра в умовах темряви

Другий експеримент виконано із марлею поверх пальця, що частково пропускає розсіяне світло. Виміряна освітленість перебувала в межах 10-50 lux, що характерно для слабкого кімнатного світла. Тут невелика кількість стороннього світла все ж потрапляла на сенсор, тому на графіку з’явилися легкі флуктуації амплітуди. Водночас рівень SpO_2 залишався коректним, а ЧСС впізнаваною та стабільною. Це демонструє, що система витримує помірний шум освітлення (рис. 3.36).

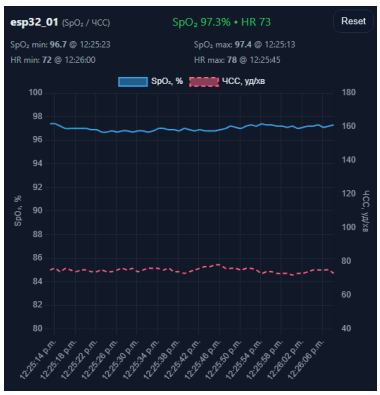

Рисунок 3.36 – Граф пульсоксометра в умовах мінімального освітлення

Третій експеримент моделював ситуацію без оптичного перекриття, але з низьким рівнем засвітки 100-300 lux (звичайне розсіяне світло кімнати без прямих джерел). У цьому випадку SpO_2 опустився нижче норми, оскільки частина інфрачервоного променя губилася через відсутність додаткового оптичного каналу тканини. Однак хвиля ЧСС залишилась майже ідеально чистою, оскільки зовнішнє світло було слабким і не створювало значних шумів. Така картина характерна для неправильного прикладання пальця (рис. 3.37).

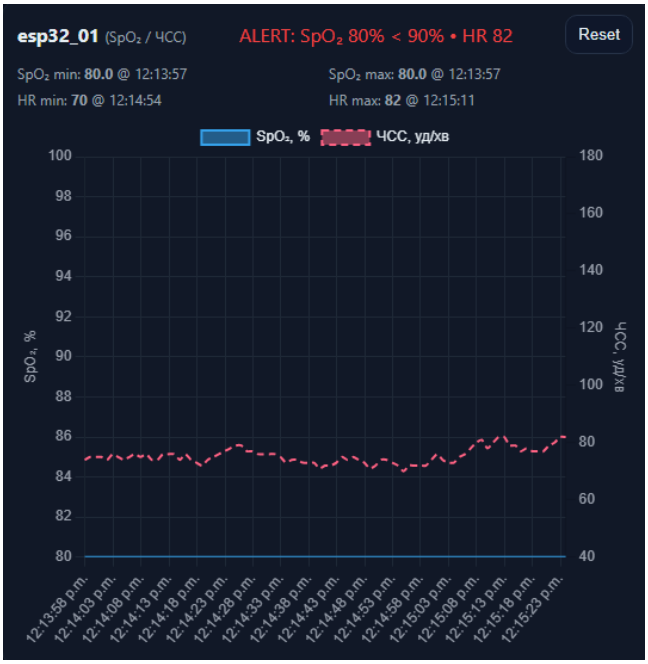

Рисунок 3.37 – Граф пульсоксометра в умовах легкого освітлення

Четвертий експеримент проводився в умовах інтенсивного освітлення 1000-5000 lux, що відповідає або яскравому денному світлу біля вікна, або прямому штучному світлу. За таких умов фотодіод MAX30102 отримує значну кількість паразитного світла, що істотно погіршує співвідношення червоного та інфрачервоного каналів. На графіках це виражено різко зниженим SpO_2 та високошумною хвилею ЧСС із локальними стрибками та «завалами». Це найбільш несприятливий сценарій для роботи пульсоксиметра (рис. 3.38).

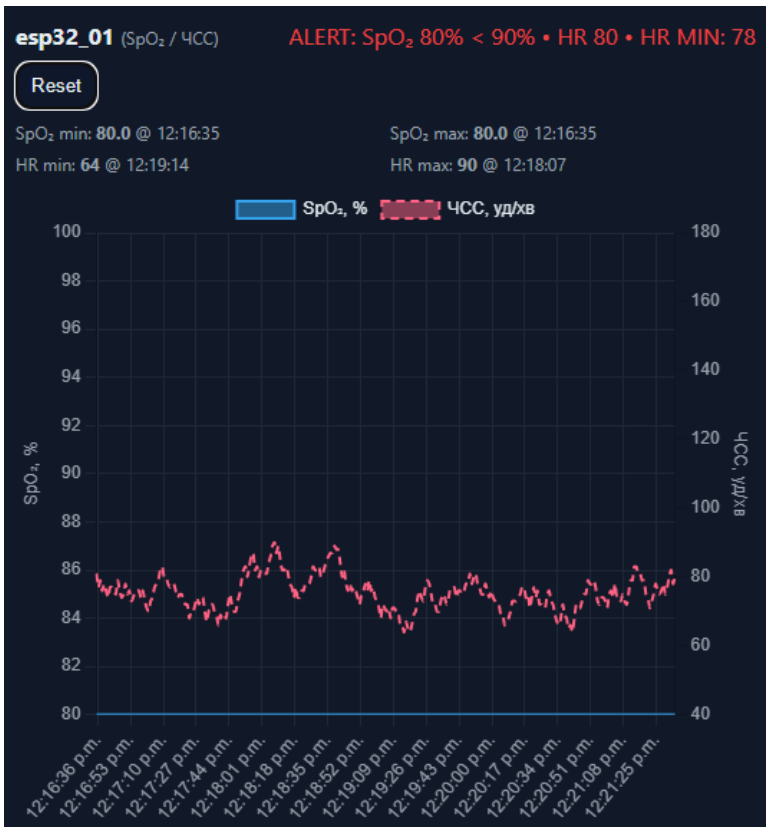

Рисунок 3.38 – Граф пульсоксометра в умовах повної освітлиності

Зміна кривих SpO_2 і ЧСС під час цих експериментів узгоджується зі зміною освітленості: менші значення lux забезпечують стабільний сигнал, тоді як сильне світло ускладнює роботу датчика через шум зовнішнього походження. Таким чином, результати підтверджують як працездатність системи, так і її адекватну реакцію на реальні експлуатаційні умови.

ВИСНОВКИ

У межах виконаного дослідження було реалізовано всі завдання, сформульовані у вступі, що дозволило комплексно оцінити можливості побудови системи моніторингу фізіологічних параметрів пацієнтів на основі мережі пульсоксиметрів.

Проведений аналіз сучасних технічних рішень для моніторингу фізіологічних показників показав, що ринок медичних IoT-систем активно розвивається, проте більшість готових рішень є або надто дорогими, або обмеженими у масштабованості. Це підтвердило актуальність створення доступного та гнучкого прототипу, здатного функціонувати у реальному часі.

Дослідження принципів роботи пульсоксиметрів та порівняння типових сенсорів дало можливість обґрунтовано обрати датчик MAX30102. Він забезпечує прийнятну точність, низьке енергоспоживання, компактність та сумісність з різними мікроконтролерами, що робить його оптимальним для побудови сенсорних вузлів у системах медичного спостереження.

Було визначено апаратну платформу та розроблено схему сенсорного модуля на основі ESP32, яка поєднує можливості Wi-Fi/BLE-зв’язку з достатньою обчислювальною потужністю для попередньої обробки сигналів. Створена апаратна конфігурація довела здатність стабільно зчитувати дані з пульсоксиметра та передавати їх бездротовими каналами.

Розроблене програмне забезпечення для збору та передачі даних реалізувало повний цикл обробки – від отримання сирих фотоплетизмографічних сигналів до формування структурованих MQTT-повідомлень. Використання протоколу MQTT забезпечило мінімальні затримки, надійність доставки та можливість масштабування мережі сенсорів без зміни архітектури.

Створено моніторинговий веб-сервіс підтвердив ефективність запропонованого підходу: система здатна отримувати та відображати дані в

режимі реального часу, будувати графіки, фіксувати зміни та забезпечувати наочність для кінцевого користувача – медичного працівника або оператора.

Під час тестування система була перевірена як із програмним генератором сигналів, так і з реальним сенсорним вузлом. Це дозволило оцінити точність, затримку передачі, стабільність зв’язку та відмовостійкість. Результати засвідчили, що прототип коректно функціонує в різних умовах та може бути адаптований для подальшого використання у більш масштабних або критичних сценаріях.

Поставлена мета дослідження була досягнута. Розроблений прототип довів, що побудова доступної телемедичної системи на основі мережі пульсоксиметрів є технічно реалістичною та доцільною. Архітектура забезпечує автоматизований збір і передачу фізіологічних показників, своєчасне відображення інформації та можливість розширення системи без суттєвих змін у її структурі. Отримані результати підтверджують, що такі рішення мають значний потенціал для застосування у медичних закладах, мобільних госпіталях та інших середовищах, де важливими є автономність, швидкість реагування та мінімальна участь персоналу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Трофімов В, Поліщук М. Аналіз надійності бездротової системи моніторингу життєвих показників на основі мережі пульсоксометрів. Міжнар. наук.-практ. конф. молодих вчених та студентів (м. Луцьк, 6 травня 2025 р). Луцьк, ЛНТУ, 2025. С. 180-182.

2. Ковальчук П., Поліщук М. Реалізація мереж пульсоксиметрів IoT для моніторингу стану пацієнтів у реальному часі. Програмно-визначена мережева архітектура для Інтернету речей: концепція та виклики. Науковий часопис «Технічні вісті». № 1(59), 2(60). Львів, 2025. С. 77-79.

3. Chan E., Chan M., Chan M. Pulse oximetry: Understanding its basic principles facilitates appreciation of its limitations. *ScienceDirect*. URL: https://www.sciencedirect.com/science/article/pii/S095461111300053X (дата звернення: 10.08.2025).

4. Sahu M. L., Atulkar M., Ahirwal M. K., Ahamad A. *Vital Sign Monitoring System for Healthcare Through IoT Based Personal Service Application* Wireless Personal Communications. 2021. URL: https://www.researchgate.net/publication/354161165_Vital_Sign_Monitoring_System_for_Healthcare_Through_IoT_Based_Personal_Service_Application (дата звернення: 12.08.2025)

5. Задорожний К. Біологія. Повторне видання. 8 клас : підручник. Харків : Ранок, 2021. 240 с.

6. Цифровий пальцевий пульсоксиметр Датчик кисню в крові Насичення РК-міні Монітор SpO2 Частота пульсу – купити найкращі товари в інтернет-магазині Ayzeze. Ayzeze. URL: https://www.ayzeze.com/uk/products/683549785f3d9701397f7b00?variant_id=683549785f3d97e3397f7b02 (дата звернення: 15.08.2025).

7. Garmin HRM 200 – пульсометр для тренувань з ANT+ та Bluetooth | 4Garmin. Все для Гармін - 4Garmin.com. URL: https://4garmin.com/datchyk-sertsevoho-rytmu-garmin-hrm-200-xs-s/ (дата звернення: 15.08.2025).

8. Види пульсоксиметрів – які бувають пульсоксиметри?. BabyMedical. URL: https://babymedical.com.ua/uk/vidy-pulsoksimetrov-kakie-byvayut-pulsoksimetry/?srsltid=AfmBOoobyp2uxLbVqx1md7irBMr1hSIdgGZ02AgvY9dxK3wKOYvRwaBF (дата звернення: 15.08.2025).

9. Wearpulse Baby Finger Oximeter. Vibeat. URL: https://vibeatstore.com/en-de/products/baby-oxygen-meter-finger-pulse-oximeter (date of access: 15.08.2025).

10. Nedis Smartlife Bluetooth pulse oximeter. Smartify. URL: https://smartify.pt/en/products/nedis-smartlife-bluetooth-pulse-oximeter-3?srsltid=AfmBOoqmbhvdBmCctGe78kqXELFlv-FQ0Ec4k7ML9_naqP68JnAkpCqq (дата звернення: 17.08.2025).

11. MQTT–SN Protocol Explained For Sensor Networks. *ZedIoT*. URL: https://zediot.com/blog/mqtt-sn-protocol-explained (дата звернення: 17.08.2025).

12. The WebSocket API (WebSockets) - Web APIs MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення:: 20.08.2025).

13. The Arduino UNO R3 Datasheet. *Arduino*. URL: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf (дата звернення: 21.08.2025).

14. Arduino Pro Mini Datasheet. *HandsOn Tech – Open Source Electronics Platform*. URL: https://www.handsontec.com/dataspecs/module/Arduino%20Pro-Mini.pdf (дата звернення: 21.08.2025).

15. Arduino Nano ESP32 Datasheet. *Arduino*. URL: https://docs.arduino.cc/resources/datasheets/ABX00083-datasheet.pdf (дата звернення: 21.08.2025).

16. MAX30102 Datasheet. *Analog*. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX30102.pdf (дата звернення: 21.08.2025).

17. HTML: HyperText Markup Language Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 03.09.2025).

18. CSS: Cascading Style Sheets MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 05.09.2025).

19. Overview of DevTools - Microsoft Edge Developer documentation. Microsoft Learn: Build skills that open doors in your career. URL:

https://learn.microsoft.com/en-us/microsoft-edge/devtools/overview (дата звернення: 05.09.2025).

20. What is Wi-Fi?. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/computer-networks/what-is-wi-fiwireless-fidelity (дата звернення: 10.09.2025)

21. Introduction to Bluetooth Low Energy. *Cdn–learn*. URL: https://cdn-learn.adafruit.com/downloads/pdf/introduction-to-bluetooth-low-energy.pdf (дата звернення: 11.09.2025).

22. Chart.j Open source HTML5 Charts for your website. URL: https://www.chartjs.org/docs/latest (дата звернення: 13.09.2025).

23. Parser for command-line options, arguments and subcommands. Python documentation. URL: https://docs.python.org/3/library/argparse.html (дата звернення: 15.08.2025).

24. JSON encoder and decoder. Python documentation. URL: https://docs.python.org/3/library/json.html (дата звернення: 17.09.2025).

25. Generate pseudo-random numbers. Python documentation. URL: https://docs.python.org/3/library/random.html (дата звернення: 18.09.2025).

26. Time access and conversions. Python documentation. URL: https://docs.python.org/3/library/time.html (дата звернення: 20.09.2025).

27. System-specific parameters and functions. Python documentation. URL: https://docs.python.org/3/library/sys.html (дата звернення: 25.09.2025).

28. Thread-based parallelism. Python documentation. URL: https://docs.python.org/3/library/threading.html (дата звернення: 05.10.2025).

29. Basic date and time types. Python documentation. URL: https://docs.python.org/3/library/datetime.html (дата звернення: 07.10.2025).

30. Using The Python MQTT-SN Client. URL: http://www.steves-internet-guide.com/python-mqttsn-client/ (дата звернення: 09.10.2025).

31. How LiPo Batteries Work? Typhon Energy-RC Battery Manufacturer, RC Aircraft Batteries, RC Helicopter Battery, RC Car Battery, rc plane battery. Typhon Energy-RC Battery Manufacturer, RC Aircraft Batteries, RC Helicopter Battery, RC

Car Battery, rc plane battery. URL: https://www.typhonenergy.com/how-lipo-batteries-work-explained-with-diagrams-expert-tips (дата звернення: 11.10.2025).

32. Wifi. URL: https://docs.arduino.cc/libraries/wifi (дата звернення: 13.10.2025).

33. PubSubClient. URL: https://docs.arduino.cc/libraries/pubsubclient (дата звернення: 15.10.2025).

34. BenoitBlanchon. JsonDocument. ArduinoJson. URL: https://arduinojson.org/v7/api/jsondocument (дата звернення: 17.10.2025).

35. Wire. URL: https://docs.arduino.cc/language-reference/en/functions/communication/wire (дата звернення: 19.10.2025)

36. max30105 Reference. Particle Welcome to the Particle docs. URL: https://docs.particle.io/reference/device-os/libraries/m/max30105 (дата звернення: 20.09.2025).

37. Time. URL: https://docs.arduino.cc/libraries/time (дата звернення: 21.10.2025).

38. Get Started with EMQX Broker EMQX Platform Docs. EMQX Documentation. URL: https://docs.emqx.com/en/cloud/latest/quick_start/introduction.html (дата звернення: 25.10.2025).