НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету Інформаційних технологій	Завідувач кафедри Комп'ютерних систем, мереж та кібербезпеки
/ Болбот І.М., д.т.н., проф. /	/ Касаткін Д.Ю., к.пед.н., доц. /
підпис ПІБ, вчене звання і ступінь	підпис ПІБ, вчене звання і ступінь
«___» _____ 2025 р.	«___» _____ 2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Дослідження веб-системи для дистанційного управління пристроями з використанням протоколу WebSocket»

Спеціальність 123 «Комп'ютерна інженерія»

Освітня програма Комп'ютерні системи та мережі

Орієнтація освітньої програми _____

Гарант освітньої програми

к. фіз.-мат. н., доц. _____ Нікітенко Є. В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

к. т. н, доц. _____ Сагун А. В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Пшонік О.С.
(підпис) (ПІБ студента)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»

завідувач кафедри

комп'ютерних систем, мереж та кібербезпеки

/ Касаткін Д.Ю. к.п.н., доц. /

підпис ПІБ, вчене звання і ступінь

« » 2025 р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Пшоніку Олександру Сергійовичу

(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): 123 - Комп'ютерна інженерія
Освітня програма: комп'ютерні системи та мережі
Орієнтація освітньої програми:

Тема магістерської роботи: «Дослідження веб-системи для дистанційного управління пристроями з використанням протоколу WebSocket»

затверджена наказом ректора НУБіП України від " 29 жовтня 2024 р. № 1941 **«С»**

Термін подання завершеної роботи на кафедру: 14 листопада 2025 р.
Вихідні дані до магістерської роботи:

Перелік питань, що підлягають дослідженню: 1. Аналіз існуючих технологій та протоколів дистанційного керування. 2. Теоретичні основи та принципи роботи протоколу WebSocket. 3. Проектування веб-системи для дистанційного управління пристроєм через Web-Socket. 4. Програмна реалізація та емуляція системи дистанційного керування на WebSocket. 5. Тестування та оцінка функціонування системи.

Перелік графічного матеріалу (за потреби):

Дата видачі завдання " 29 " жовтня 2024 **р**.

Керівник магістерської кваліфікаційної роботи Сагун А.В.,
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання Пшонік О.С.
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка задачі магістерської роботи	18.10.2022	Виконано
2	Аналіз предметної області	12.07.2023	Виконано
3	Проектування системи	23.08.2023	Виконано
4	Реалізація системи	15.10.2023	Виконано
5	Тестування системи	30.10.2023	Виконано
6	Оформлення пояснювальної записки	01.11.2023	Виконано
7	Оформлення графічного матеріалу	01.11.2023	Виконано
8	Попередній захист роботи на кафедрі	09.11.2023	Виконано
9	Подача роботи для перевірки на доброчесність	10.11.2023	Виконано
10	Публічний захист роботи		

Студент _____ Пшонік О.С.
(підпис) (ініціали та прізвище)

Керівник проекту (роботи) Сагун А.В.
(підпис) (ініціали та прізвище)

ЗМІСТ

ВСТУП

Актуальність теми. Сучасні тенденції розвитку Інтернету речей (IoT) та систем автоматизації вимагають ефективних методів організації двонапрямного обміну даними між користувачем і пристроєм у реальному часі. Традиційні технології, засновані на HTTP-запитах, мають обмеження щодо швидкодії та затримок передачі.

Мета роботи полягає в розробці та дослідженні веб-системи для дистанційного управління апаратними пристроями, яка використовує протокол WebSocket для забезпечення ефективного двонапрямного обміну даними в режимі реального часу.

Основні завдання роботи: 1) аналіз існуючих методів та технологій організації дистанційного управління пристроями через мережу Інтернет, зокрема на основі протоколів HTTP, MQTT та WebSocket; 2)дослідження принципів роботи протоколу WebSocket, його архітектурні особливості, переваги та недоліки порівняно з традиційними клієнт-серверними моделями; 3) розроблення архітектури веб-системи для дистанційного управління пристроями з використанням WebSocket-з’єднання, визначити основні компоненти системи (сервер, клієнт, контролер); 4) реалізація експериментальної моделі системи, що забезпечує двонапрямний обмін даними між веб-інтерфейсом користувача та мікроконтролером (наприклад, ESP32); 5) налаштування та тестування системи в емуляційному середовищі Wokwi для перевірки стабільності та швидкодії з’єднання; 4) проведення порівняльного аналізу ефективності використання WebSocket у дистанційному керуванні пристроями відносно інших мережевих підходів.

Об'єкт дослідження - процес організації двонапрямного обміну даними між веб-клієнтом і мікроконтролером у системах дистанційного керування пристроями.

Предмет дослідження: методи, засоби та технології побудови веб-систем для дистанційного управління пристроями із використанням протоколу WebSocket.

Методи дослідження складаються з: емпіричніх методів, методів статистичного та порівняльного аналізу, методів програмної інженерії.

Наукова новизна отриманих результатів. У роботі запропоновано підхід до побудови веб-системи дистанційного керування, який поєднує використання WebSocket-протоколу, мікроконтролера ESP32 та веб-інтерфейсу, що забезпечує синхронну взаємодію між апаратною та програмною частинами системи.

На відміну від традиційних рішень, розроблена система забезпечує мінімізацію затримки передачі даних, оптимізацію мережевих ресурсів та підвищену стабільність з’єднання у реальному часі.

Практичне значення отриманих результатів.

Структура роботи. У своєму складі робота має: вступ, п’ять розділів, висновки, перелік використаних джерел з 14 найменувань. Загальний обсяг роботи становить 47 сторінок основного тексту.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ТА ПРОТОКОЛІВ ДИСТАНЦІЙНОГО КЕРУВАННЯ

1.1. Поняття та класифікація систем дистанційного управління пристроями

Тема дослідження безпосередньо пов’язана з класом веб-орієнтованих СДУП реального часу, які реалізують двосторонню взаємодію між клієнтом і пристроєм через Інтернет-протоколи нового покоління, зокрема WebSocket.

Протокол WebSocket забезпечує постійне двостороннє з’єднання між клієнтським веб-додатком і сервером, що дозволяє оперативно передавати команди керування та отримувати дані від пристроїв без необхідності повторних HTTP-запитів. Такий підхід відносить досліджувану систему до інтерактивних, подієво-орієнтованих систем дистанційного управління.

Таким чином, розробленювана система відповідатиме в тій чи іншій мірі сучасній класифікації системи дистанційного управління пристроями (СДУП) як:

- веб-орієнтована,
- реального часу,
- з використанням двостороннього протоколу обміну даними,
- хмарного типу (SaaS або PaaS),
- що забезпечує масштабованість, мобільність і мінімальні затримки у керуванні пристроями.

Системи дистанційного управління пристроями (СДУП) класифікуються за різними ознаками:

- за способом передачі даних (дротові, бездротові, комбіновані);
- за архітектурою (централізовані, децентралізовані, хмарні);
- за протоколами обміну (HTTP, MQTT, WebSocket, CoAP, AMQP тощо);

- за типом керованих об’єктів (механічні пристрої, електронні модулі, промислове обладнання, “розумні” побутові прилади);
- за режимом роботи (онлайн / офлайн, реального часу / з затримкою).

Існуючу на сьогодні класифікацію систем дистанційного управління пристроями можна показати у вигляді схеми (рис.1.1). Дану класифікація складана на базі джерел [1-4].

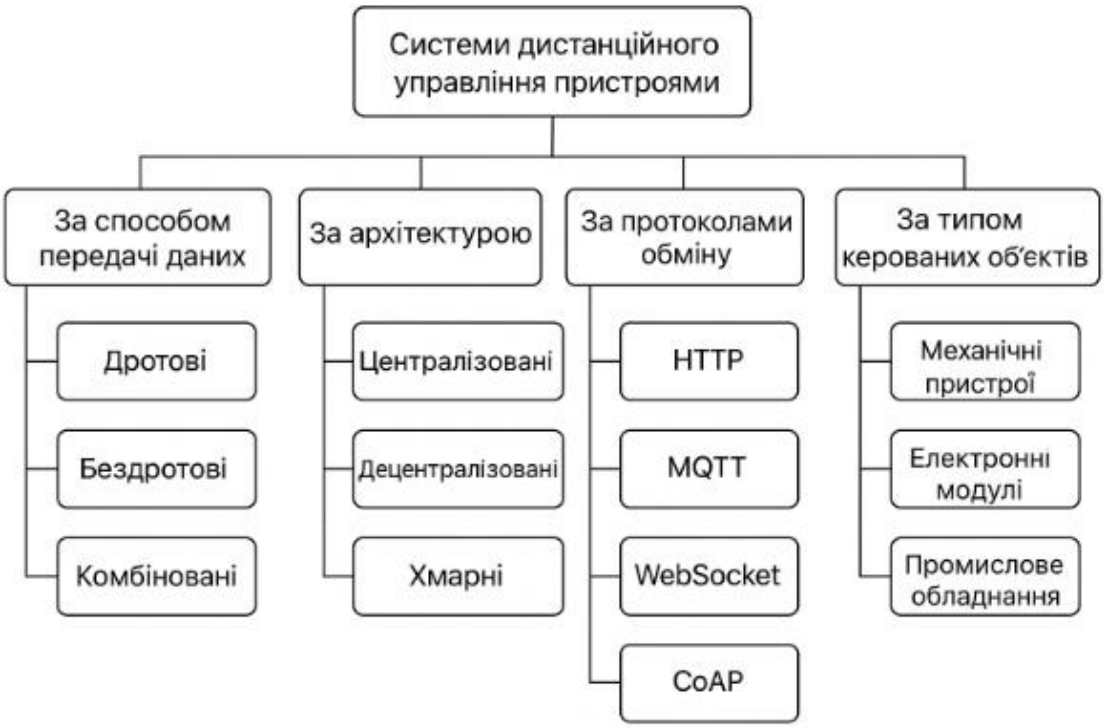

Рисунок 1.1 – класифікація систем дистанційного управління пристроями

1.2. Огляд існуючих мережевих технологій для віддаленого контролю

Для вибору оптимальної з існуючих мережевих технологій, які застосовуються для віддаленого контролю пристроями розглянемо найбільш відомі і розповсюджені: HTTP, MQTT, CoAP, WebSocket [5-8].

1.2.1 Технологія MQTT

MQTT став протоколом для потокової передачі даних між пристроями з обмеженою потужністю CPU та/або часом автономної роботи, а також для

мереж з дорогою або низькою пропускною здатністю, непередбачуваною стабільністю або високою затримкою. Саме тому MQTT відомий як ідеальний транспорт для IoT. Він, в свою чергу, побудований на протоколі TCP/IP. На рисунку 1.2 показана схема взаємодії учасників технології MQTT [9].

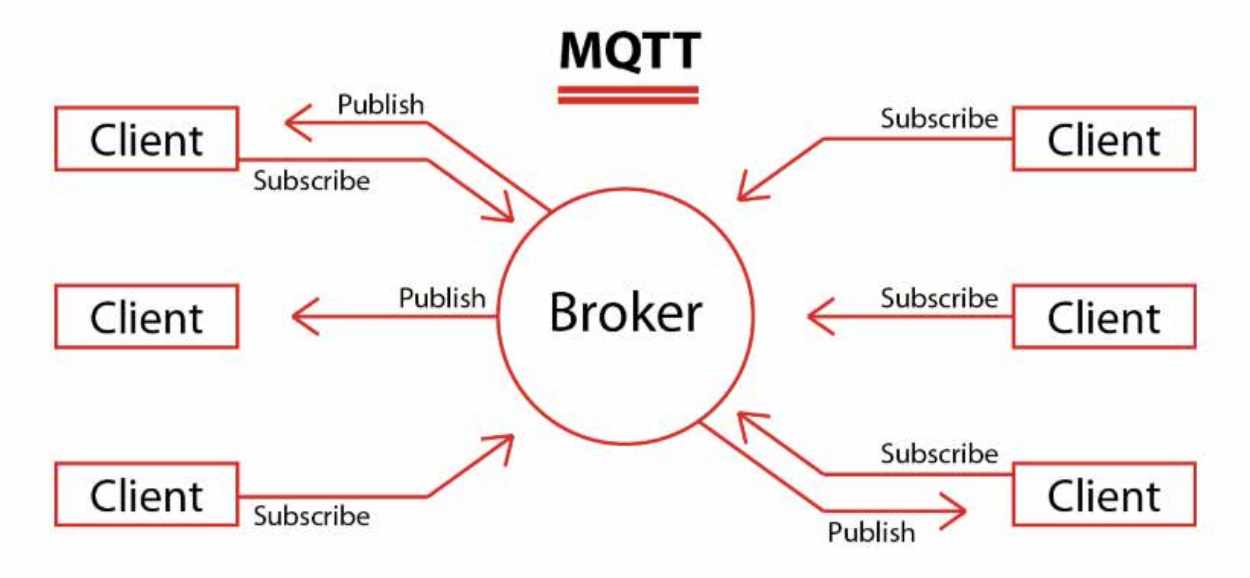

Рисунок 1.2 – Схема взаємодії учасників технології MQTT

MQTT надає спосіб створення ієрархії каналів зв'язку - свого роду гілка з листям. Щоразу, коли видавець має нові дані для поширення серед клієнтів, повідомлення супроводжується приміткою контролю доставки. Клієнти вищого рівня можуть отримувати кожне повідомлення, тоді як клієнти нижчого рівня можуть отримувати повідомлення, що стосуються лише одного чи двох базових каналів, «відгалуженим» у нижній частині ієрархії. Це полегшує обмін інформацією розміром від двох байт до 256 мегабайт.

1.2.3 Технологія CoAP

CoAP (Constrained Application Protocol) — це протокол прикладного рівня пристроїв Інтернету речей (IoT) з обмеженими ресурсами. Він розроблений за подобою HTTP, але оптимізований для роботи в мережах з

низькою пропускною здатністю та невеликим об'ємом пам'яті, використовуючи протокол UDP замість TCP.

CoAP працює за моделлю "клієнт-сервер" та підтримує основні методи REST (GET, POST, PUT, DELETE) для взаємодії з ресурсами, які представлені по URI.

На рис.1.3 показана схема взаємодії учасників технології CoAP [9].

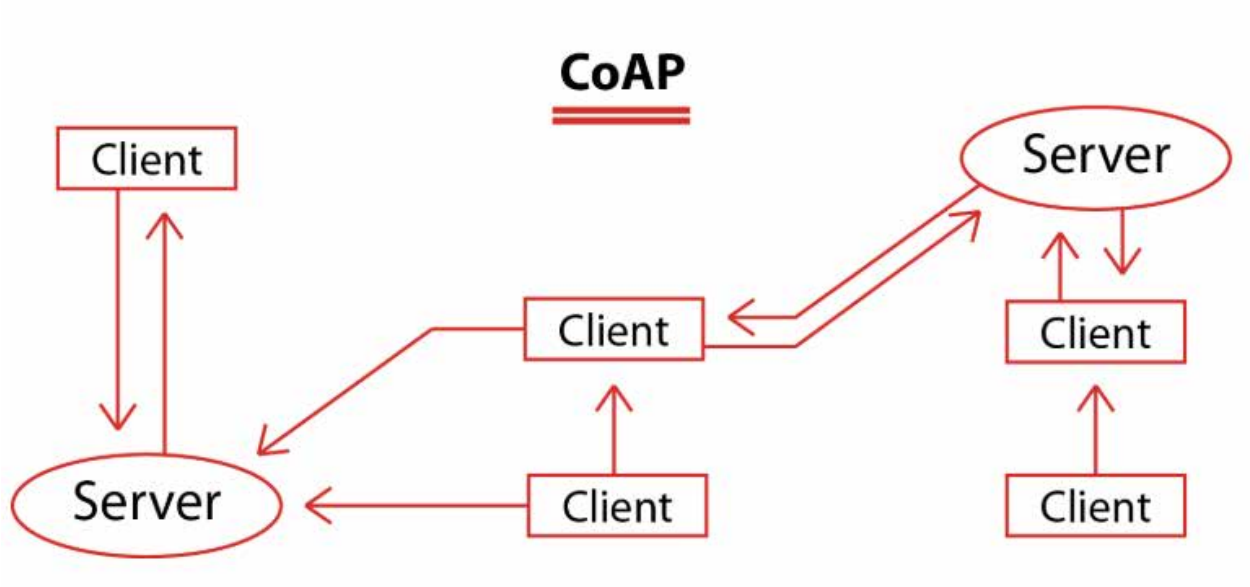

Рисунок 1.3 – Схема взаємодії учасників технології CoAP

Споживання енергії при роботі через MQTT суттєво вище, ніж при використанні CoAP. Але і кількість підтримуваних типів повідомлень в ньому більше в 4 рази. MQTT підтримує три типи сервісів QoS.

1.2.4 Механізм віддаленого доступу через WebSocket

WebSocket — це двонаправлений повнодуплексний протокол зв’язку між клієнтом та сервером. Це означає, що на відміну від HTTP-протоколу, який працює за принципом «запит від клієнта — відповідь від сервера», у WebSocket і сервер, і клієнт можуть надсилати один одному повідомлення. Кожна сторона комунікації здатна одночасно отримувати та відправляти дані. Схема такої взаємодії показана на рис.1.4 [10].

Рисунок 1.4 – принципова схема взаємодії сторін при використанні протоколу WebSocket

При задіянні Web-Socket обмін повідомленнями проходить через єдиний канал зв’язку. Він залишається відкритим протягом усієї комунікації, а за необхідності будь-яка зі сторін може його закрити.

1.3. Вибір компонент клієнт-серверних систем у середовищі веб-додатків

При виборі оптимальної мережевої технології дистанційного керування (МТДК) сформуємо критерії порівняння (табл.1) [11].

Ваговий коефіцієнт в таблиці 1.1 слід обирати, виходячи з того факту, що загальна сума вагових коефіцієнтів по всім критеріям не може перевищувати один. В зв’язку з тим, що ваги коефіцієнтів нормалізовані можна скористатися методом аналізу ієрархій (MAI) для вибору оптимальної МТДК пристроєм [12].

Таблиця 1.1 - Критерії порівняння для вибору оптимальної мережевої технології дистанційного керування

№ поз	Критерій	Позначення	Ваговий коефіцієнт*
1	Затримка передачі (Latency)	L	0.35
2	Надійність з'єднання (Reliability)	R	0.25
3	Складність реалізації (Complexity)	C	0.2
4	Підтримка реального часу (Real-Time)	RT	0.2

Для того, щоб вибір з використанням MAI міг бути практично реалізований, необхідно сформувати узагальнену матрицю парних порівнянь (табл.1.2).

Таблиця 1.2 - Узагальнена матриця парних порівнянь

Технологія	Критерії порівнянь			
	L	R	C	RT
HTTP	2	3	5	2
MQTT	4	4	3	4
CoAP	3	3	4	3
Web-Socket	5	4	2	5

Після нормалізації та формування інтегральних оцінок для кожної з порівнювальних технологій на базі сформованої в таблиці 1.2 матриці парних порівнянь отримаємо наступний результат (таблиця 1.3).

Таблиця 1.3 - Нормалізація та інтегральна оцінка для кожної МТДК

Технологія	L	R	C	RT	Σ
HTTP	0.14	0.15	0.20	0.08	**0.57**
MQTT	0.28	0.25	0.12	0.16	**0.81**
CoAP	0.21	0.19	0.16	0.12	**0.68**
Web-Socket	0.35	0.25	0.08	0.20	0.88

Як можна бачити видно зі стовпця «Σ», таблиці 1.3 який відображає інтегральну оцінку попарних порівнянь, найефективнішою технологією при виборі системи дистанційного управління пристроєм показала себе WebSocket. Вона в змозі забезпечити мінімальну затримку, постійне двонапрямне з'єднання. В графічній інтерпретації результати інтегральних попарних порівнянь для МТДК показані на рис.1.5.

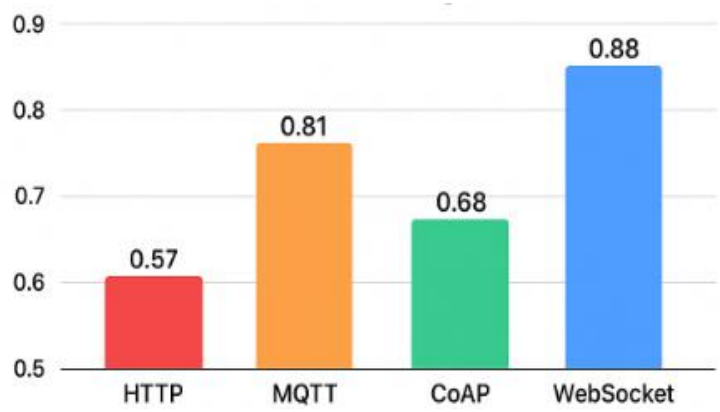

Рисунок 1.5 - Результати інтегральних попарних порівнянь для МТДК

Технологія Web-Socket добре підходить для реального часу, незважаючи на порівняно складнішу практичну реалізацію.

Під час вибору технології web-socket, як основи для системи дистанційного управління пристроями передбачається, що така система є клієнт-серверною і проектується, як веб-додаток. Такі програмні додатки

мають низку специфічних рис, пов’язаних із реалізацією ними принципів двонаправленої комунікації, продуктивністю, безпекою та реальним часом.

Таким чином, можна констатувати, що протокол WebSocket може забезпечити оптимальне співвідношення швидкодії, малої затримки та ефективності для випадку двонапрямного обміну даними в системах реального часу. Тобто, технології керування на базі протоколу WebSocket оптмальні при проектуванні web-орієнтованих систем керування та моніторингу IoT-пристроїв.

Особливості побудови клієнт-серверних систем у середовищі веб-додатків, коли йдеться про систему дистанційного управління пристроями, побудовану з використанням технології WebSocket, мають низку характерних рис, пов’язаних із принципами двонаправленої комунікації, продуктивністю, безпекою та реальним часом.

На рисунку 1.6 показано функціональну схему клієнт-серверної взаємодії системи дистанційного управління пристроями з використанням технології WebSocket.

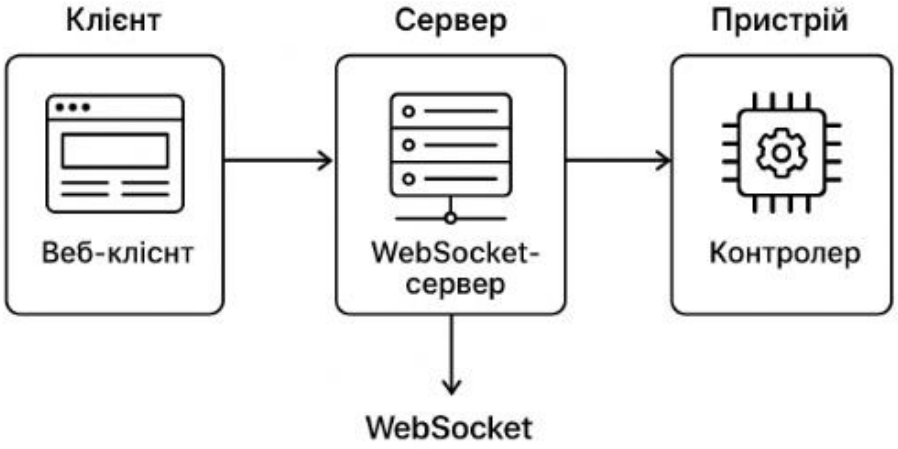

Рисунок 1.6 - функціональна схема клієнт-серверної взаємодії системи дистанційного управління пристроями **з використанням технології WebSocket**

Переваги WebSocket у системах дистанційного керування порівняно з альтернативними технологіями наведено в таблиці 1.1.

Таблиця 1.1 - Переваги WebSocket у системах дистанційного керування пристроями

Показник порівяння	**Назва технологій дистанційного керування пристроями**	
	HTTP	**Web-Socket**
Тип зв'язку	Односторонній (запит-відповідь)	Двосторонній (постійне з'єднання)
Затримка	Висока	Мінімальна
Передача подій	Обмежена	Підтримується нативно
Обсяг службових даних	Великий	Низький
Практична придатність для керування в реальному часі	Низька	Висока

З усіх, наведених в таблиці показників найбільш критичними з точки зору практичного використання є: затримка, постійна двосторонність зв'язку та практично придатність технології. В даному випадку технологія Web-Socket має найбільші переваги. Далі розглянемо наявні недолікі традиційних протоколів та переваги WebSocket для роботи в реальному часі.

Оскільки веб-сокети підтримують двонаправлений зв'язок, то вони добре підходять для ситуацій, коли необхідно забезпечити швидкий двосторонній обмін даними (онлайн-ігри, чати, фінансові застосунки, новини, обмін даними з IoT-девайсами).

Веб-сокети не використовують, якщо немає потреби в інтерактивності, немає постійного двостороннього трафіку, а натомість — існують високі вимоги щодо безпеки. Адже довготривале відкрите з'єднання додає чимало ризиків.

WebSocket протокол дає можливість створювати інтерактивну комунікацію між клієнтом і сервером без необхідності постійно пінгувати

сервер. Таким чином ми економимо час, прискорюємо роботу застосунку і можемо підтягувати зміни в режимі реального часу.

Використання технології веб-сокетів не завжди потрібне. Часом дана технологія може стати джерелом додаткових ризиків і потребує окремих знань та досвіду для правильного, безпечного застосування. Як будь-яка комунікація в мережі, комунікація по веб-сокет протоколу має певні вразливості. Такі вразливості інколи можуть мати значний вплив на дистанційне керування пристроєм, а інколи – такі вразливості не впливають принципово на об’єкт керування.

Висновки до розділу 1

Визначено, що застосування клієнт–серверних систем у середовищі веб-додатків із використанням технології WebSocket дає змогу створювати високопродуктивні системи дистанційного управління пристроями реального часу. Така архітектура відзначається низькою затримкою, мінімальними мережевими витратами, гнучкістю інтеграції з IoT-компонентами та підтримкою сучасних методів захисту даних, що робить її ефективним рішенням для побудови інтелектуальних систем управління.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ ТА ПРИНЦИПИ РОБОТИ ПРОТОКОЛУ WEB-SOCKET

2.1. Архітектура протоколу WebSocket та принцип встановлення двонапрямного з'єднання

WebSocket — це протокол, який забезпечує постійне двонапрямне з'єднання між клієнтом (наприклад, веббраузером) і сервером через один TCP-канал.

Основні компоненти, з якими співпрацює прокол WebSocket:

- Клієнт — зазвичай веб-браузер або додаток, що ініціює з'єднання.
- Сервер WebSocket — програма, що підтримує протокол (наприклад, Node.js, Python, Java).
- TCP-з'єднання — постійний канал зв'язку без повторних HTTP-запитів.

З'єднання через Web-Socket відбувається в два етапи:

Етап 1. HTTP Handshake

З'єднання починається як звичайний HTTP-запит:

```
GET /ws HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

Відповідь сервера виглядає наступним чином:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

Після цього HTTP-з’єднання оновлюється (upgrade) до WebSocket-з’єднання, і канал стає двонапрямним.

Етап 2. Обмін даними

Після успішного "upgrade", клієнт і сервер можуть вільно надсилати повідомлення у будь-якому напрямку без додаткових запитів.

Схема архітектури WebSocket у вигляді блок-схеми представлена на рис.2.1.

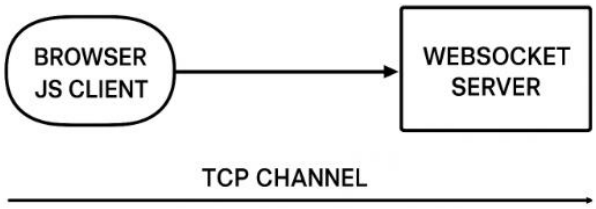

Рисунок 2.1 – Схема архітектури WebSocket

2.1.1 Формат кадрів даних, управління сесією та безпекою з’єднання

Для управління сесією зв’язку використовується правило, що WebSocket-сесія триває, поки активне TCP-з’єднання.

Основні механізми забезпечення управління сесією наступні:

- Ping/Pong — перевірка життєздатності каналу.
- Timeouts — автоматичне роз’єднання при неактивності.
- Reconnect logic — клієнтські бібліотеки часто реалізують повторне з’єднання при розриві.

Безпека з’єднання через Web-Socket забезпечується такими механізмами:

1) WSS:// — безпечна версія протоколу поверх TLS (аналог HTTPS).

2) Перевірка походження (Origin) — сервер перевіряє заголовок Origin, щоб уникнути CSRF.

3) Аутентифікація — через токени, JWT або HTTP cookies під час початкового запиту.

4) Маскування (Masking) — всі кадри від клієнта обов’язково маскуються, щоб уникнути атак типу cache poisoning.

2.2. Використання WebSocket у поєднанні з технологіями HTML5, JavaScript та Node.js

WebSocket у поєднанні з HTML5, JavaScript та Node.js використовується для створення повнодуплексних (двонаправлених) каналів зв'язку в реальному часі між клієнтом (браузером) та сервером.

Роль технології HTML5 полягає у наданні клієнтської частини з API WebSocket, а JavaScript використовується для роботи з API. Node.js, як серверне середовище на JavaScript, дозволяє керувати цими з'єднаннями на сервері, що ідеально підходить для програм, що вимагають швидкої передачі даних, наприклад, для чатів або онлайн-ігор.

Таким чином, WebSocket це потужний інструмент для створення real-time додатків.

2.3. Порівняння WebSocket із REST API, Socket.IO та ін. технологіями

В технології WebSocket використовується тип з’єднання: двонапрямне (full-duplex), постійне

Механізм діє наступний: клієнт і сервер встановлюють TCP-з’єднання, яке залишається відкритим

Призначення технології - передача даних у реальному часі (наприклад, чати, IoT, біржі, ігри)

Формат даних: текст (JSON) або двійковий. Затримка при обробці та передаванні інформації мінімальна, оскільки відсутні повторні HTTP-запити

До переваги технології можна віднести:

- миттєвий обмін даними в обох напрямках;
- зменшення навантаження на сервер;
- підтримка бінарних повідомлень.

До недоліків можна віднести складніше масштабування та необхідність підтримки з боку клієнта (браузера або пристрою) в разі, якщо потреба в такій підтримці виникає.

Якщо узагальнити результати порівняння технології WebSocket із REST API, Socket.IO та іншими популярними підходами до організації обміну даними у веб-додатках, то можна побачити наочно всі переваги та недоліки кожної з них (таблиця 2.1).

Таблиця 2.1 - результати порівняння технології обміну даними WebSocket із REST API, Socket.IO та SSE

Характеристика	**Назва технології обміну даними**			
	Web-Socket	**REST API**	**Socket.IO**	**SSE**
Тип з'єднання	Двонапрямне	Однонапрямне	Двонапрямне	Однонапрямне
Тривалість	Постійна	Короткочасна	Постійна	Постійна
Формат даних	JSON, Binary	JSON, XML	JSON, Binary	Текст
Затримка	Мінімальна	Висока	Мінімальна	Низька
Реалізація реального часу	Так	Ні	Так	Так (тільки push)
Складність	Середня	Низька	Середня	Низька

Масштабованість	Складна	Висока	Складна	Середня

Висновки до розділу 2

Внаслідок аналізу та порівняння архітектури протоколу WebSocket та принципів встановлення двонапрямного з’єднання були визначені технології обміну даними та принцип і формат передавання повідомлень в системі дистанційного керування пристроєм.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ВЕБ-СИСТЕМИ ДЛЯ ДИСТАНЦІЙНОГО УПРАВЛІННЯ ПРИСТРОЄМ ЧЕРЕЗ WEB-SOCKET

3.1. Загальна архітектура системи та компонент

Джойстик - це пристрій введення, який складається з рухливого важеля, який можна обертати чи натискати, і використовується для керування. Джойстик живиться від 5В та підключається до аналогових контактів контролера.

Біполярний крововий двигун має 4 піни підключення(2 позитивних та 2 негативних сигнали). Двигун може бути підключений напряму до контролера, а може через плату A4988. Для роботи з крововим двигуном у середовищі wokwi використовується бібліотека Stepper.h.

<Stepper.h> - бібліотека для керуванням біполярним двигуном. Має наступні функції Stepper(steps, pin1, pin2), Stepper(steps, pin1, pin2, pin3, pin4) – дозволяє підключати двигун по двох або чотирьох провідній схемі. setSpeed(rpm) – задає швидкість обертання двигуна. step(steps) – обертає двигун двигун на вказану кількість кроків.

MQTT (Message Queuing Telemetry Transport) - легкий, простий протокол передачі повідомлень, спроектований для обміну даними між пристроями в умовах обмежених ресурсів та змінюючихся мереж. Застосовується широко в Інтернеті речей (IoT) та інших сценаріях, де потрібна ефективна передача даних між пристроями.

3.2. Вибір середовища розробки та технологій (Node.js, Express, HTML/CSS, ESP32)

Одна із областей застосування MQTT – це обмін між пристроями та програмами, що підключені до Інтернет. Для цього необхідно

використовувати загальнодоступний брокер. Наприклад - test.mosquitto.org. Даний брокер є безкоштовним, тому він не гарантує безперебійну роботу сервісу. Через це його не слід використовувати для практичних рішень, що потребують надійних з'єднань та цілодобової доступності і функціональності. Брокер test.mosquitto.org рекомендується використовувати виключно в навчальних цілях.

Перед тим, як запустити готовий проект слід MQTT скористатися тестовими клієнтами та брокером MQTT (рис.3.8). Тестовим приладом буде http://test.mosquitto.org/gauge/ [13].

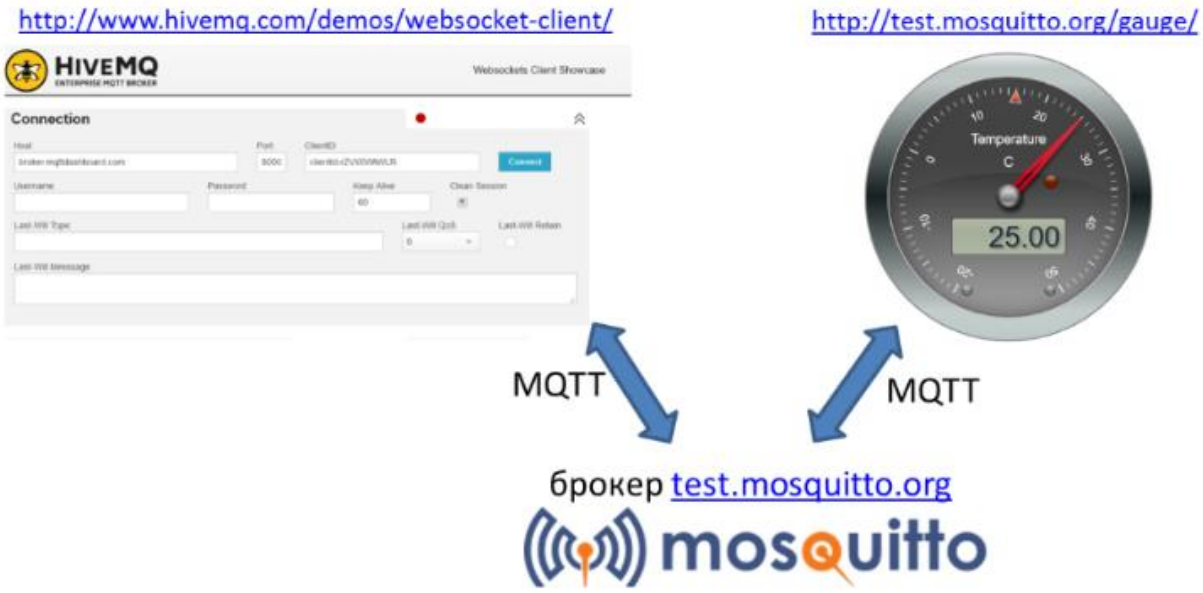

Рисунок 3.2 - схема взаємодії тестового клієнту та брокера MQTT

На сторінці WebSocket-клієнта в полі Host вводимо test.mosquitto.org , в полі Port 8080 (цей порт завжди відкритий), після чого натиснемо кнопку "Connect". Повинен з'явитися напис Connected.

У випадку відсутності зв'язку з брокером слід здійснити спробу доступу пізніше.

3.3. Розробка технології підключення пристрою та алгоритму роботи

У системах дистанційного керування та моніторингу на базі ESP32 часто поєднуються різні канали комунікації. Найпоширенішими є UART (Serial) та WebSocket. Кожна технологія вирішує свої задачі, а разом вони забезпечують як локальну взаємодію між модулями, так і хмарну/мережеву комунікацію з сервером.

Вибір UART пояснюється тим, що це найпростіший та найнадійніший спосіб передачі даних між мікроконтролерами або між ESP32 та ПК. UART є внутрішнім локальним каналом, який не залежить від мережі Wi-Fi і дозволяє отримувати дані максимально швидко та з мінімальною затримкою.

В даному проєкті UART і WebSocket у можуть працювати одночасно й виконувати різні функції.

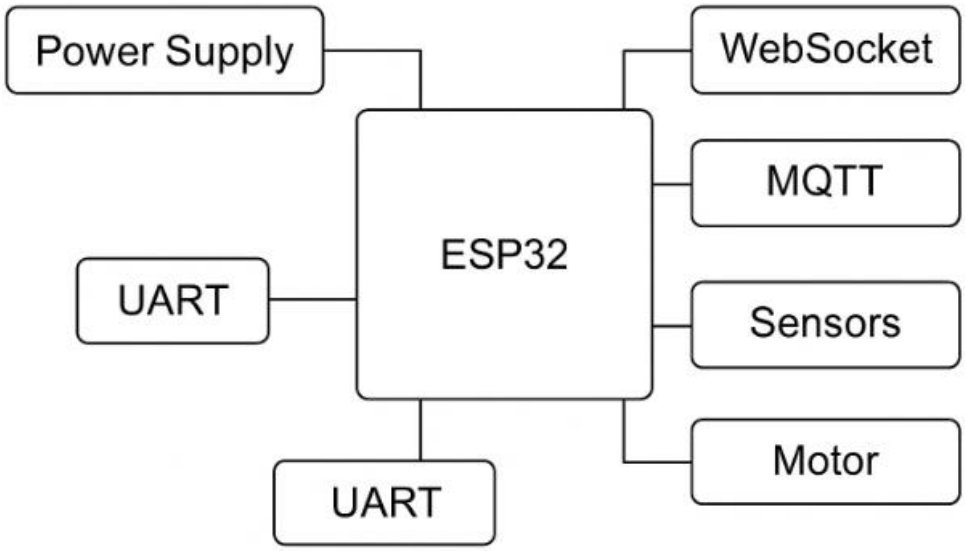

Рисунок 3.3 - Схема взаємодії компонентів системи дистанційного керування

Алгоритм роботи системи UART + WebSocket для схеми взаємодії компонентів, показаної на рис.3.3 наступна:

1. ESP32 під’єднується до Wi-Fi та встановлює WebSocket-з’єднання з сервером.
2. Через UART отримує дані від сенсорів або інших модулів.
3. Обробляє дані у FreeRTOS-задачах.
4. Через WebSocket передає результати на сервер і приймає команди (наприклад, змінити швидкість двигуна).
5. Через UART ESP32 може передавати у ПК статистику навантаження:
   - CPU load
   - free heap
   - delays
   - network events
6. У разі втрати Wi-Fi — UART продовжує працювати, WebSocket робить повторне під’єднання.

Висновки до розділу 3

В розділі визначено загальну архітектуру системи та компонент, здійснено вибір середовища розробки та технологій (Node.js, Express, HTML/CSS, ESP32) та проведено розробку технології підключення пристрою та алгоритму роботи.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕМУЛЯЦІЯ СИСТЕМИ ДИСТАНЦІЙНОГО КЕРУВАННЯ НА WEBSOCKET

4.1. Реалізація серверної частини з використанням WebSocket API

Серверна частина створеної системи дистанційного управління пристроєм включає наступні компоненти:

1. WebSocketServer (wss) — головний сервер, що керує з'єднаннями. Дана компонента має приймати всі з'єднання, розпізнавати ESP-контролер та web-браузери, які використовуються в якості front-end додатків.
2. ESP32 Client — мікроконтролер, який підключається та ідентифікується через "esp-handshake" (комунікаційний протокол).
3. Browser Client — користувацький інтерфейс (веб-додаток), який надсилає та отримує дані через WebSocket.
4. Message Router — логіка в wss.on('message'), що маршрутизує повідомлення між ESP та браузерами.

Зв'язок та принцип взаємодії між даними компонентами наведено на рис.4.1.

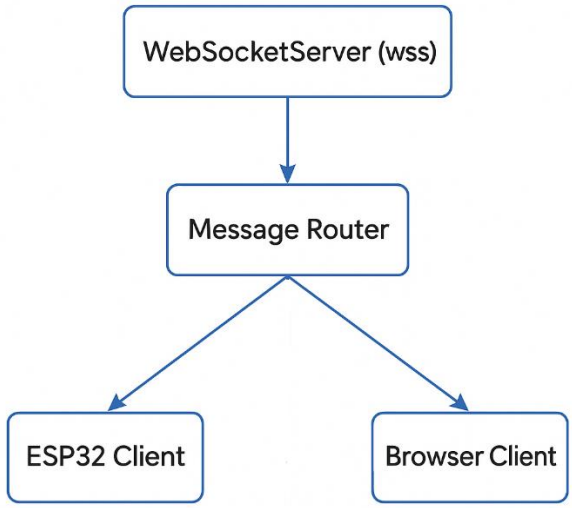

Рисунок 4.1 – Функціональна схема взаємодії між компонентами серверної частини системи дистанційного управління пристроями

Програмна логіка роботи серверної частини наступна:

1. Імпортується бібліотека ws (WebSocket для Node.js) та створюється WebSocket-сервер, який слухає порт 8080.

```
const WebSocket = require('ws');
const wss = new WebSocket.Server({ port: 8080 });
```

Виводиться повідомлення в консоль для підтвердження запуску:

```
console.log("WebSocket server listening on ws://0.0.0.0:8080");
```

Збереження поточного підключення від **ESP32** (оскільки передбачено лише один ESP-клієнт одночасно).

```
let espSocket = null;
```

Коли приходить повідомлення, сервер спочатку намагається розібрати його як JSON:

```
wss.on('connection', (ws, req) => {
const ua = req.headers['user-agent'] || '';
console.log('New connection. UA:', ua);
```

Якщо повідомлення має type: "esp-handshake", сервер вважає це сигналом, що підключився ESP32:

1) Зберігає сокет у espSocket
2) Позначає його прапорцем isESP = true.

3) зберігає espId.

Якщо повідомлення прийшло від ESP, сервер транслює його всім браузерам, що підключені (крім ESP).

Якщо повідомлення прийшло від браузера, воно пересилається на ESP, якщо той підключений.

Якщо ESP не підключений, сервер повертає повідомлення з помилкою.

При розриві з'єднання сервер: а) логує, хто саме відключився; б) якщо це був ESP, очищує espSocket.

4.2. Розроблення веб-інтерфейсу для користувача (панель керування пристроями)

Алгоритм розробки web-інтерфейсу користувача для системи дистанційного керування пристроями через Web-Socket складається з декількох етапів:

Етап 1. Аналіз вимог. Тут визначається, якими пристроями необхідно керувати (двигун, LED-освітлення, кроковий двигун тощо) та які параметри необхідно передавати (швидкість, рівень PWM, показники сенсорів). Також слід розуміти чи потрібна авторизація.

Етап 2. Проєктування архітектури. Тут визначити модель взаємодії. Наприклад:

- браузер → WebSocket-сервер → МК ESP32;
- МК ESP32 → WebSocket-сервер → браузер.

На даному етапі також визначається словник команд протоколу JSON): *cmd, value, device, status, telemetry.*

Етап 3. Процес розробки WebSocket-сервера. Відбувається запуск WebSocket-сервера (Node.js) з підтримкою одночасних підключень браузерів та відслідковується логіка маршрутизації повідомлень. Так само важливо здійснювати ідентифікацію ESP за допомогою протоколу handshake (type:"esp-handshake").

Етап 4. Реалізація WebSocket на ESP32 супроводжується налаштування Wi-Fi мережі. У випадку підвищених вимог щодо безпеки встановлюється TLS – шифрування. Тут же відбувається відправлення телеметричних даних, прийом команд та керування через протоколи GPIO/PWM/I2C.

Етап 5. Створення веб-інтерфейсу залежить від бажаного вигляду інтерфейсу управління. Як правило, використовується HTML структура панелі керування з елементами керування: кнопки, повзунки, графіки. Для більш зручного і естетичного інтерактивного управління додається JS-логіка та відбувається відкриття WebSocket наступним фрагментом програмного коду:

```
const ws = new WebSocket("ws://server:8080");
```

Зміна стану інтегрфейсу користовуача відбувається при надходженні даних з мікроконтролера ESP:

```
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  updateUI(data);
};
```

Етап 6. Тестування контролера ESP32 на предмет втрати зв’язку з мережею та тест розриву WebSocket-з’єднання.

Також відбувається перевірка навантаження при великій кількості повідомлень та тест кросбраузерної сумісності.

Етап 7. Оптимізація та безпека. Може бути реалізована WebSocket через WSS (TLS), шляхом обмеження розміру повідомлень та введенням в програмний код команд, що реалізують антиспам фільтр. Також застосовується контроль дозволених дій у браузері.

В результаті отримуємо web-інтерфейс системи дистанційного керування, наведений на рис.4.2.

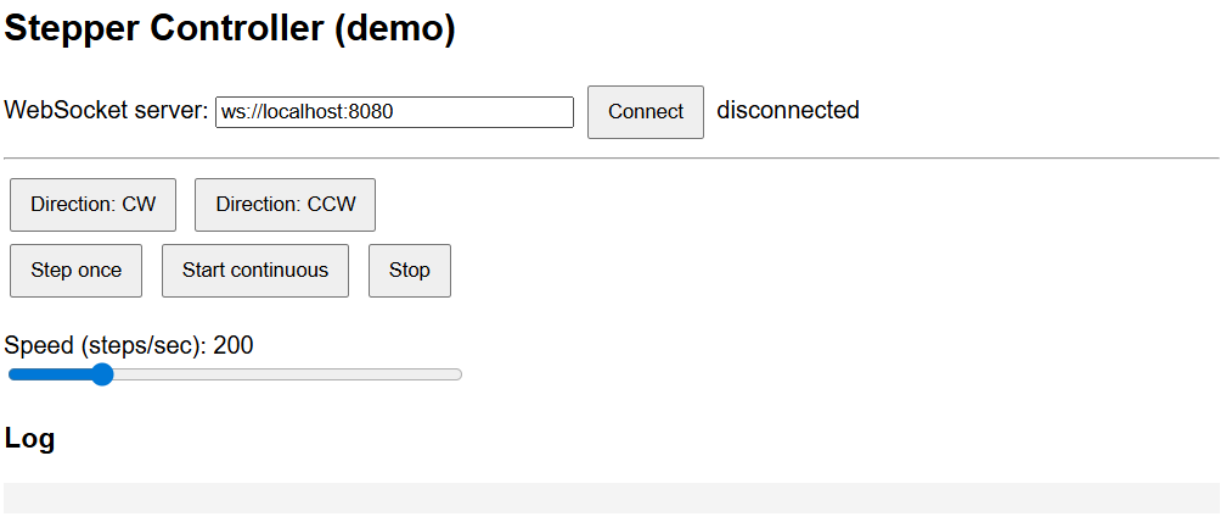

Рисунок 4.2 - web-інтерфейс системи дистанційного керування

4.3. Налаштування та тестування роботи системи у середовищі Wokwi в емуляції ESP32

Налаштування та тестування системи дистанційного управління прецензійним двигуном здійснено в середовищі Wokwi. Використання Wokwi дає певні переваги, а саме - підтримка WebSocket, що дозволяє проводити повноцінні експерименти ще до прошивки реального пристрою. Wokwi дозволяє виконувати код ESP32 повністю у браузері. Тобто WebSocket-клієнт, написаний у програмі, працює так само, як і на реальному мікроконтролері:

- встановлює TCP-з’єднання,
- виконує WebSocket-handshake,
- обмінюється повідомленнями в реальному часі.

Це дозволяє протестувати логіку взаємодії з сервером без апаратної плати.

Також середовище забезпечує повну емуляцію роботи мережі та прозору взаємодію з реальним WebSocket-сервером, візуалізацію логіки роботи та потоків даних

На рисунку 4.3 показано момент підключення до MQTT протоколу та успішну активацію передавання даних через Wi-Fi мережу контролера ESP 32.

Посилання на проект доступно за адресою: https://wokwi.com/projects/381814408446201857.

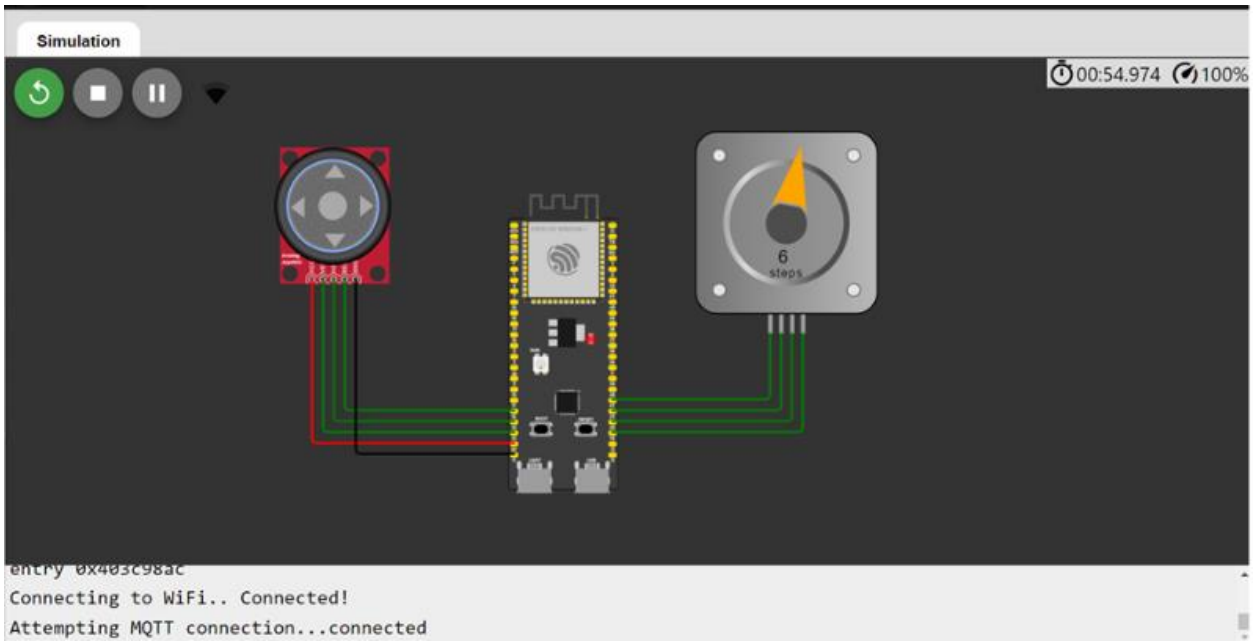

Рисунок 4.3 – Підключення до MQTT протоколу

Як видно з рисунка 4.4 розгорнутий сервер успішно відображає дані відпрацювання кроків позиціонування на програмному Web-трекері.

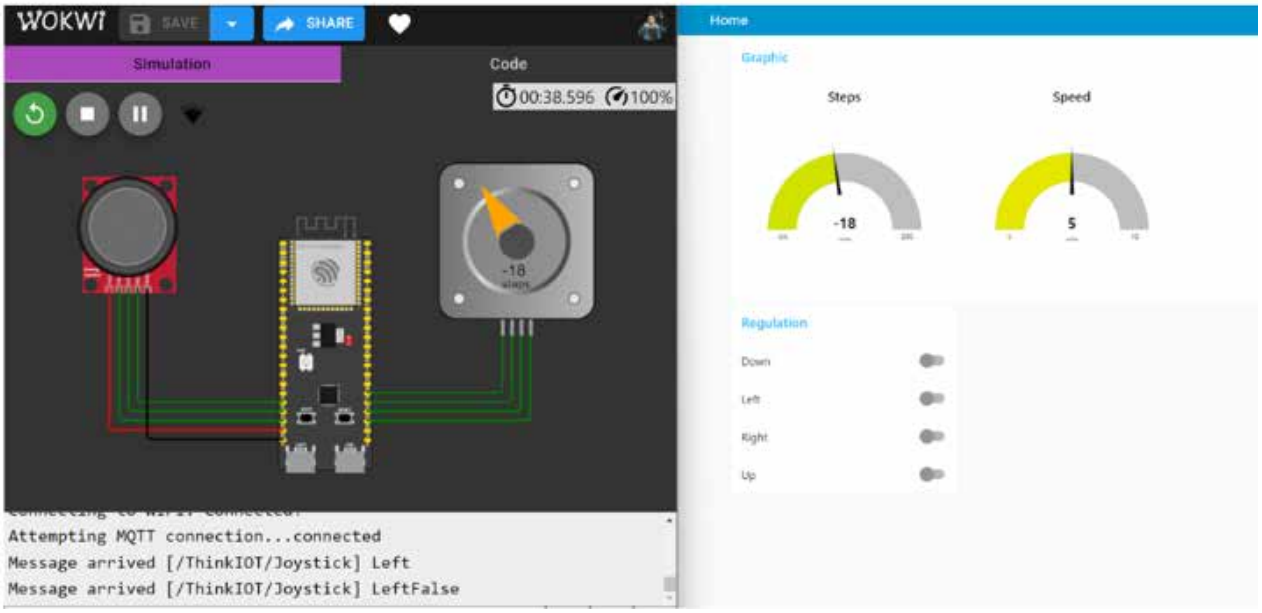

Рисунок 4.4 – один з режимів роботи програми при симуляції в WOKWI

Таким чином, при роботі проекту на ESP32 запущено програмне забезпечення, яке відкриває з’єднання WebSocket із сервером.

Мікроконтролер надсилає та приймає повідомлення через це WebSocket-з’єднання для керування, телеметрії та отримання команд.

Wokwi забезпечує “IoT Gateway”, який дозволяє ESP32-симулятору з’єднуватися з інтернетом або з іншими сервісами через TCP.

За допомогою публічного шлюзу встановлюється з’єднання з WebSocket-сервером.

Код ESP32 виводить дані в серійний порт (Serial), щоб відслідковувати події (наприклад, підключення, отримані WebSocket-повідомлення).

Висновки до розділу 4

В розділі розроблені та налаштовані окремі елементи дистанційної системи керування пристроєм на базі технології WebSocket та здійснена симуляції роботи системи на різних режимах в середовищі імітаційного моделювання WOKWI.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ОЦІНКА ФУНКЦІОНУВАННЯ СИСТЕМИ

5.1 Аналіз результатів експериментів по швидкодії, стабільності, затримкам, використанню ресурсів.

Важливим етапом є визначення на аналіз результатиів експериментальних досліджень роботи створеної системи дистанційного керування пристроєм на основі WebSocket-технології та мікроконтролера ESP32. Дослідження проводилися як у симуляторі Wokwi, так і з використанням реального обладнання. Оцінювання проводилося за такими критеріями: швидкодія**,** стабільність роботи**,** затримки передавання даних, та ефективність використання ресурсів програмно-апаратної платформи.

5.1.1 Методика зняття параметрів під час тестування навантаження ESP32

Здійснювалася оцінка роботи мікроконтролера ESP32 під різними видами навантаженням (робота з WebSocket, MQTT, сенсорами, драйвером двигуна). Для цього застосовувалися програмні та апаратні методи вимірювання. Вимірювання виконувалися для наступних показників:

1) завантаження CPU;

2) використання оперативної пам’яті;

3) час відповіді (latency);

4) стабільність мережевого з’єднання;

5) енергоспоживання (за потреби).

Тестування проводилося з використанням вбудованих механізмів FreeRTOS, системних функцій для мікроконтролера ESP-IDF та зовнішнього логування через інтерфейс UART. Реєстрація даних здійснювалася з типовою

штатною частотою роботи системи, що складає 5–10 Гц. Такий режим забезпечив достатню для аналізу пікових навантажень деталізацію.

Швидкодія системи визначалась за кількістю WebSocket-повідомлень, що можуть бути передані та оброблені у реальному часі та за реакцією мікроконтролера на команди, отримані від веб-інтерфейсу.

Таким чином, використання WebSocket-технології для швидкого дистанційного керування пристроями може бути доцільним і виправданим, оскільки технологія забезпечує високу швидкодію та низькі затримки передачі команд.

Під час експериментів оцінювалась також здатність системи підтримувати постійне з’єднання, здатність відновлюватися після втрати зв’язку та здатність коректно обробляти помилки.

В результаті було встановлено, що:

- webSocket-з’єднання між браузером та серверною частиною залишається досить стабільним протягом тривалого часу без відчутних коливань продуктивності роботи всієї системи;
- при розриванні з’єднання з боку контролера ESP32 сервер коректно ініціалізує стан системи та переходить в режим очікування нового підключення пристрою;
- у випадку зупинки симуляції у Wokwi або втрати Wi-Fi-з’єднання на реальному МК ESP32 часто фіксується зависання клієнтського з’єднання без коректного завершення сеансу. Ситуація виправляється реалізацією механізмів *heartbeat* (ping/pong) та автоматичного перепідключення;
- при підключенні кількох web-клієнтів завжди фіксувалосяться дублювання повідомлень та збільшення навантаження на серверну частину маршрутизації. Виправлення ситуації потребує обмеження кількості одночасних сесій або ж впровадження складнішої логіки управління клієнтами.

Загалом, створена система демонструє високу стабільність роботи, але потребує впровадження додаткових механізмів для підвищення надійності у випадках відмов мережі Wi-Fi.

5.1.1 Аналіз затримок передавання даних

Для системи дистанційного керування прецизійним механізмом затримки передавання даних є важливим фактором. Даний параметр оцінювався окремо для кожного етапу для таких маршрутів повідомлення:

- «браузер → сервер»;
- «сервер → контролер ESP32»;
- для повного маршруту.

В результаті були отримані наступні значення для різних каналів зв'язку (таблиця 5.1):

Таблиця 5.1 – Параметри затримок передавання даних

Канал зв'язку	**Затримка, мс**
Web-інтерфейс → Server	1–3
Server → ESP32 (Wokwi)	0,5–2
Web-інтерфейс → Server → ESP32	2–5
ESP32 реальна → Wi-Fi → Server	40
Зворотний канал ESP32 → Web-інтерфейс	2–5 (локально)

Можна відмітити, що основними чинниками, які впливають на затримки при роботі системи дистанційного керування при використанні WebSocket, є:

- застосування бездротової мережі Wi-Fi в реальному фізичному мікронтролері ESP32;
- навантаження на канали зв'язку та маршрутизуючий пристрій (теоретично);
- підключення декількох клієнтів до сервера одночасно;
- процес обробки JSON-повідомлень на стороні сервера.

При симуляції у Wokwi затримки є мінімальними, що дозволяє використовувати дану систему в якості системи реального часу: керування сервоприводами, LED-індикацією, передавання телеметрії тощо.

5.2 Тестування використання ресурсів системи при роботі

Вимірювання завантаження CPU визначалося за допомогою системних функцій [14]:

- vTaskGetRunTimeStats() — для збору статистики часу, який займають окремі задачі;
- uxTaskGetSystemState() — для визначення стану та пріоритетності задач;
- UART-логування частоти виконання подій WebSocket/MQTT.

На основі експериментальних даних було побудовано графік зміни завантаження CPU мікроконтролера протягом 60 секунд (рис. 5.2).

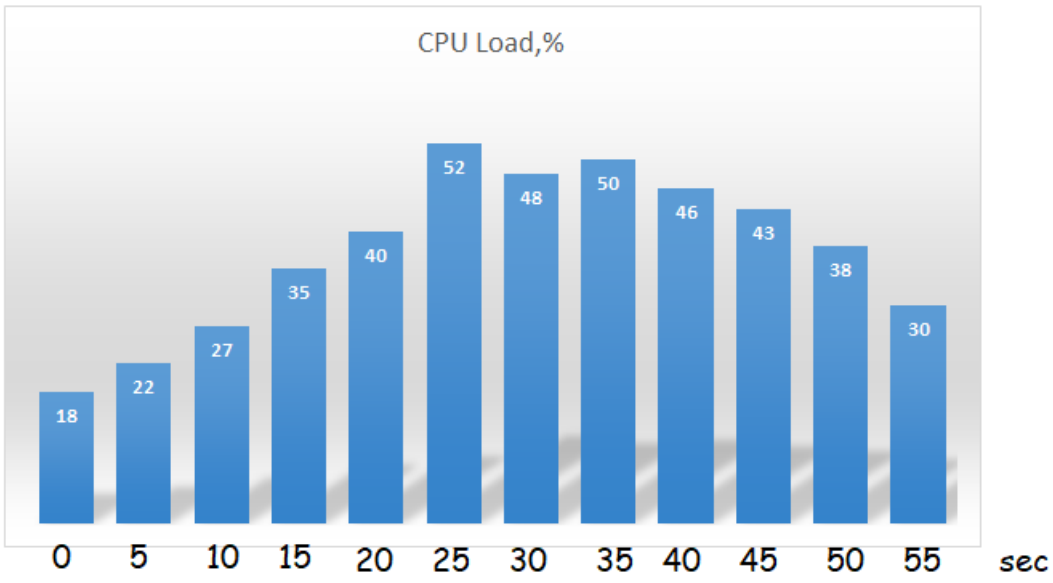

Рисунок 5.2 – Графік завантаження процесора ESP32

Оцінка ресурсоспоживання проводилась для двох основних компонентів: серверної частини, яка створена з використанням технології Node.js та мікроконтролера ESP32.

Також вимірювався рівень використання оперативної пам’яті мікроконтролера. Для такого аналізу використовувались системні функції:

- xPortGetFreeHeapSize() — загальна кількість вільної пам’яті;
- heap_caps_get_free_size() — пам’ять за типами;
- uxTaskGetStackHighWaterMark() — аналіз використання стеку задач.

Графік на рис. 5.3 демонструє зміну доступної heap-пам’яті протягом роботи системи.

Рисунок 5.3 – Динаміка використання heap-пам’яті ESP32

Для серверної частина частині на тестовій платформі визначено, що процес Node.js споживає наступні ресурси:

- процесор: 1–5% при активній передачі повідомлень,
- оперативна пам’ять: 25–60 МБ,

Пропускна здатність необхідна мінімальна, оскільки передаються компактні JSON-пакети з даними.

Проведена оцінка показує, що серверна частина економно споживає ресурси і може обслуговувати декілька сотень одночасних підключень без суттєвих витрат апаратних ресурсів.

5.2.1 Пряме тестування навантаження на мікроконтролер ESP32

При тестуванні навантаження на ресурси мікроконтролера ESP32 він демонструє такі показники:

- використання оперативної пам’яті WebSocket-клієнтом: 15–25 КБ,
- завантаження процесора: <5% у типових сценаріях,
- основне енергоспоживання припадає виключно на Wi-Fi-модуль.

Таким чином, мікроконтролер ESP32 має достатній запас апаратних ресурсів для виконання паралельних задач таких, як: обробки сенсорів, робота з PWM/ADC, логікою керування та організація та підтримання обміну даними.

Реальний мікроконтролера ESP32, підключений через Wi-Fi, демонструє затримку реакції у межах від 1 до 5 мс, що є типовим для бездротових мікропроцесорних платформ.

Висновки до розділу 5

Під час експериментів по тестуванню системи встановлено ряд параметрів роботи системи: динаміка використання пам’яті та ресурсів CPU мікроконтролера ESP32, що використовується на різних режимах роботи, параметри затримок даних.

ЗАГАЛЬНІ ВИСНОВКИ

В рамках проведеного дослідження веб-системи для дистанційного управління пристроями з використанням протоколу WebSocket було здійснено:

- вибір та підтвердження переваг технології Web-Socket, як оптимальної технології двонапрямного обміну даними для системи дистанційного управління пристроями, що використовують в якості фізичного середовищі комунікацій мережу Ethernet та Internet;
- здійснено програмну реалізацію та симуляцію роботи системи дистанційного керування на базі технології Web-Socket.

Під час експериментів по тестуванню системи встановлено ряд параметрів роботи системи:

– середню швидкість обміну текстовими WebSocket-повідомленнями при з’єднанні між клієнтом та сервером, що складає від 300 до 800 повідомлень за сек у середовищі локального виконання;

– отримані результати експериментів по швидкодії, стабільності, затримкам, використанню ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Seong, J., Ranjan, R., Kye, J., Lee, S., & Lee, S. (2023). Enhancing industrial communication with Ethernet/Internet Protocol: A study and analysis of real-time cooperative robot communication and automation via Transmission Control Protocol/Internet Protocol. Sensors, 23(20), 8580. https://doi.org/10.3390/s23208580

2. Корж, О. О. (2022). Комп'ютерно-інтегрована система керування переміщенням мобільних стелажів: технічні рішення для дистанційного управління через мережу Ethernet [Магістерська кваліфікаційна робота]. Київ: КНУТД. https://er.knutd.edu.ua/handle/123456789/22927

3. Посашков, О., & Цимбал, О. (2024). Розроблення архітектури системи віддаленого доступу до навчального лабораторного обладнання з використанням автоматизованих рішень. Innovative Technologies and Scientific Solutions for Industries, № 3 (29). https://itssi-journal.com/index.php/ittsi/article/view/508/461

4. Yang, Y. H., Zhang, B. Y., Zhang, N., & Gao, C. (2013). Design of remote control equipment based on industrial Ethernet. Applied Mechanics and Materials, 397-400, 1785-1789. https://doi.org/10.4028/www.scientific.net/AMM.397-400.1785

5. Diaconu, A. (2022). The WebSocket Handbook (v2.0). Ably Realtime. Retrieved from https://pages.ably.com/hubfs/the-websocket-handbook.pdf pages.ably.com

6. Kulshrestha, A. (2021). An empirical study of HTML5 WebSockets and their cross-browser behaviour. International Journal of Computer Applications, 82(6). Retrieved from https://research.ijcaonline.org/volume82/number6/pxc3892221.pdf

7. Бражиненко, М. (2019). IoT low-bandwidth networks: протоколи M2M-зв'язку (CoAP тощо). Системи управління, навігації та зв'язку, 1(159). https://journal-hnups.com.ua/index.php/soi/article/download/727/

8. IoTMadLab. (2023). Performance evaluation of CoAP and MQTT with realistic network scenarios. Retrieved from https://iotmadlab.es/wp-content/uploads/2023/10/Performance_CN_2021.pdf

9. PickData. (2019, October 21). MQTT vs CoAP, the battle to become the best IoT protocol. PickData. https://www.pickdata.net/news/mqtt-vs-coap-best-iot-protocol

10. Єленська, В. (2023, 30 травня). WebSockets: навіщо потрібні та як з ними працювати. ProIT. https://proit.ua/websockets-navishcho-potribni-ta-iak-z-nimi-pratsiuvati-2/

11. Saaty, Thomas L. Mathematical Principles of Decision Making (Principia Mathematica Decernendi) (2009). Pittsburgh: RWS. ISBN 1-888603-10-0. «Comprehensive coverage of the AHP, its successor the ANP, and further developments of their underlying concepts»

12. Sahun, A. V., & Pshonik, O. S. (2025). *Pidkhid do vyboru tekhnolohii obminu danymy dlia systemy dystantsiinoho keruvannia prystroiamy v merezhi Ethernet ta Internet*. Innovations of Modern Science and Education, Proceedings of the 2nd International Scientific and Practical Conference, Perfect Publishing, Vancouver, Canada, 364–369. URL: https://sci-conf.com.ua/ii-mizhnarodna-naukovo-praktichna-konferentsiya-innovations-of-modern-science-and-education-29-31-10-2025-vankuver-kanada-arhiv/

13. jpmens. (2020, March 5). tempgauge [Computer software]. GitHub. https://github.com/jpmens/tempgauge

14. FreeRTOS. (n.d.). uxTaskGetSystemState() (API reference). Retrieved November 16, 2025, from https://freertos.org/Documentation/02-Kernel/04-API-references/03-Task-utilities/01-uxTaskGetSystemState

ДОДАТКИ

Додаток А - Програмний код серверної частини додатку - server.js

```
// Simple WebSocket relay server: browser <-> server <-> esp32
const WebSocket = require('ws');

const wss = new WebSocket.Server({ port: 8080 });
console.log("WebSocket server listening on ws://0.0.0.0:8080");

let espSocket = null;

wss.on('connection', (ws, req) => {
  const ua = req.headers['user-agent'] || '';
  console.log('New connection. UA:', ua);

  ws.on('message', (msg) => {
    try {
      const data = JSON.parse(msg);
      if (data && data.type === 'esp-handshake') {
        console.log('ESP connected:', data.id);
        espSocket = ws;
        ws.isESP = true;
        ws.espId = data.id;
        return;
      }
    } catch (e) { /* not JSON or other message */ }

    if (ws.isESP) {
```

```
        // message from ESP -> broadcast to all non-ESP clients
        wss.clients.forEach((client) => {
          if (client !== ws && client.readyState === WebSocket.OPEN &&
!client.isESP) {
            client.send(msg);
          }
        });
      } else {
        // message from browser -> forward to ESP if present
        if (espSocket && espSocket.readyState === WebSocket.OPEN) {
          espSocket.send(msg);
        } else {
          ws.send(JSON.stringify({ type: 'error', message: 'No ESP connected' }));
        }
      }
    });

    ws.on('close', () => {
      console.log('Connection closed', ws.isESP ? '(ESP)' : '');
      if (ws.isESP) {
        espSocket = null;
      }
    });
  });
```

Додаток Б – програмний код фронт-енд частини

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>Stepper WebSocket Controller (Demo)</title>
  <style>
    body { font-family: Arial; padding: 20px; max-width:800px; }
    button { margin: 4px; padding: 8px 12px; }
    input[type=range]{ width: 300px; }
    pre { background:#f4f4f4; padding:10px; }
  </style>
</head>
<body>
  <h2>Stepper Controller (demo)</h2>
  <div>
    <label>WebSocket server: <input id="url" value="ws://localhost:8080"
size="30"></label>
    <button id="connect">Connect</button>
    <span id="status">disconnected</span>
  </div>

  <hr>
  <div>
    <button id="dirCW">Direction: CW</button>
    <button id="dirCCW">Direction: CCW</button>
    <br>
    <button id="stepOnce">Step once</button>
    <button id="start">Start continuous</button>
```

```
  <button id="stop">Stop</button>
  <br><br>
  <label>Speed (steps/sec): <span id="speedVal">200</span></label><br>
  <input type="range" id="speed" min="10" max="1000" value="200">
</div>

<h3>Log</h3>
<pre id="log"></pre>

<script>
  let ws = null;
  const status = document.getElementById('status');
  const log = document.getElementById('log');
  document.getElementById('connect').onclick = () => {
    const url = document.getElementById('url').value;
    ws = new WebSocket(url);
    ws.onopen = () => {
      status.textContent = 'connected';
      appendLog('Connected to ' + url);
    };
    ws.onclose = () => {
      status.textContent = 'disconnected';
      appendLog('Disconnected');
    };
    ws.onmessage = (ev) => {
      appendLog('From server: ' + ev.data);
    };
    ws.onerror = (e) => appendLog('Error: ' + e);
  };
```

```
        function appendLog(s){
          log.textContent += s + "\n";
          log.scrollTop = log.scrollHeight;
        }

        function send(obj){
          if (ws && ws.readyState === WebSocket.OPEN)
ws.send(JSON.stringify(obj));
          else alert('Not connected to server');
        }

        document.getElementById('dirCW').onclick = () => send({cmd:'setDir',
dir:'CW'});
        document.getElementById('dirCCW').onclick = () => send({cmd:'setDir',
dir:'CCW'});
        document.getElementById('stepOnce').onclick = () => send({cmd:'step',
steps:1});
        document.getElementById('start').onclick = () => {
          const s = document.getElementById('speedVal').textContent;
          send({cmd:'start', speed: Number(s)});
        };
        document.getElementById('stop').onclick = () => send({cmd:'stop'});
        document.getElementById('speed').oninput = (e) => {
          document.getElementById('speedVal').textContent = e.target.value;
        };
      </script>
    </body>
    </html>
```

Додаток В – програмний код мікроконтролера керування

```
/*
 esp32_stepper.ino
 Requires: arduinoWebSockets library and ArduinoJson (for ESP32)
*/

#include <WiFi.h>
#include <WebSocketsClient.h>
#include <ArduinoJson.h>

const char* ssid     = "WOKWI-GUEST";
const char* password = "";
const char* ws_host  = "demo.wokwi.test"; // placeholder: replace with your server address or ngrok URL
const uint16_t ws_port = 8080;
const char* ws_path = "/";

WebSocketsClient webSocket;

// Pins for stepper driver (simulate)
const int PIN_STEP = 18;  // STEP
const int PIN_DIR  = 19;  // DIR
const int PIN_EN   = 5;   // ENABLE or LED for visualization

volatile bool running = false;
volatile int direction = 1; // 1 = CW, -1 = CCW
volatile int steps_to_do = 0;
volatile int step_delay_us = 5000; // microseconds between steps (controls speed)
```

```
hw_timer_t * timer = NULL;

void IRAM_ATTR onTimer(){
  static bool level = false;
  if (running || steps_to_do > 0) {
    digitalWrite(PIN_STEP, level ? HIGH : LOW);
    level = !level;
    if (!level) {
      if (steps_to_do > 0) steps_to_do--;
    }
  } else {
    digitalWrite(PIN_STEP, LOW);
  }
}

void webSocketEvent(WStype_t type, uint8_t * payload, size_t length) {
  switch(type) {
    case WStype_DISCONNECTED:
      Serial.println("[WS] Disconnected");
      break;
    case WStype_CONNECTED:
      Serial.println("[WS] Connected to server");
      {
        DynamicJsonDocument doc(128);
        doc["type"] = "esp-handshake";
        doc["id"] = "esp32-1";
        String out; serializeJson(doc, out);
        webSocket.sendTXT(out);
      }
```

```
        break;
      case WStype_TEXT:
        {
          DynamicJsonDocument doc(256);
          DeserializationError err = deserializeJson(doc, payload, length);
          if (err) {
            Serial.println("[WS] JSON parse error");
            break;
          }
          const char* cmd = doc["cmd"];
          if (!cmd) break;
          if (strcmp(cmd, "setDir")==0) {
            const char* d = doc["dir"];
            if (strcmp(d,"CW")==0) { direction = 1; digitalWrite(PIN_DIR, HIGH);
}
            else { direction = -1; digitalWrite(PIN_DIR, LOW); }
            Serial.printf("Set dir: %s\n", d);
          } else if (strcmp(cmd, "step")==0) {
            int s = doc["steps"] | 1;
            steps_to_do += s;
            Serial.printf("Queued steps: %d\n", s);
          } else if (strcmp(cmd, "start")==0) {
            int sp = doc["speed"] | 200;
            if (sp <= 0) sp = 1;
            step_delay_us = (1000000 / sp) / 2;
            running = true;
            timerAlarmWrite(timer, step_delay_us, true);
            Serial.printf("Start continuous, speed=%d sps, delay=%d us\n", sp,
step_delay_us);
          } else if (strcmp(cmd, "stop")==0) {
```

```
        running = false;
        Serial.println("Stop continuous");
       } else if (strcmp(cmd, "setSpeed")==0) {
        int sp = doc["speed"] | 200;
        if (sp <= 0) sp = 1;
        step_delay_us = (1000000 / sp) / 2;
        timerAlarmWrite(timer, step_delay_us, true);
        Serial.printf("Speed set %d sps\n", sp);
       }
      }
      break;
    default:
      break;
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN_STEP, OUTPUT);
  pinMode(PIN_DIR, OUTPUT);
  pinMode(PIN_EN, OUTPUT);
  digitalWrite(PIN_STEP, LOW);
  digitalWrite(PIN_DIR, HIGH);
  digitalWrite(PIN_EN, LOW);

  WiFi.begin(ssid, password);
  Serial.print("WiFi connecting");
  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 40) {
   delay(200);
```

```
    Serial.print('.');
    attempts++;
  }
  Serial.println();
  if (WiFi.status() == WL_CONNECTED) {
    Serial.print("WiFi connected, IP: "); Serial.println(WiFi.localIP());
  } else {
    Serial.println("WiFi not connected - continue (Wokwi simulation)");
  }

  webSocket.begin(ws_host, ws_port, ws_path);
  webSocket.onEvent(webSocketEvent);
  webSocket.setReconnectInterval(5000);

  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, step_delay_us, true);
  timerAlarmEnable(timer);
}

void loop() {
  webSocket.loop();
  delay(10);
}
```

ABSTRACT

Explanatory note: 53 pages, 15 figures, 6 tables, 2 appendix in 10 pages, 14 sources.

REMOTE CONTROL, WEBSOCKET, PRECISION ENGINE, ESP-32, WOKWI

Modern trends in the development of the Internet of Things (IoT) and automation systems require efficient methods for organizing real-time bidirectional data exchange between a user and a device. Traditional technologies based on HTTP requests have limitations in terms of performance and transmission latency.

The aim of this work is to develop and investigate a web-based system for remote control of hardware devices that uses the WebSocket protocol to ensure efficient real-time bidirectional data communication.

The object of investigation is the process of organizing bidirectional data exchange in remote storage systems.

The first chapter summarize that the combination of client-server systems with average web applications based on the use of WebSocket technology makes it possible to create highly productive remote control systems for real-time devices.

The second chapter tells that as a result of the analysis and comparison of the architecture of the WebSocket protocol and the principles of establishing a bidirectional connection, data exchange technologies and the principle and format of message transmission in the system of remote control of the device were determined.

The third chapter were selected the overall system architecture and components were determined, the development environment and technologies (Node.js, Express, HTML/CSS, ESP32) and the device connection technology and work algorithm were developed.

In the fourth chapter were developed and configured individual elements of the remote device control system based on WebSocket technology and simulations of the system's operation in various modes were carried out in the WOKWI simulation environment.

In the fifth chapter a number of parameters of the system were set: the dynamics of the use of memory and CPU resources of the ESP32 microcontroller used in different modes of operation, parameters of data delays.

During this study the technology was justified as the optimal solution for real-time bidirectional data exchange over Ethernet and Internet networks. A software implementation and simulation of the system were developed, followed by experimental testing. The experiments made it possible to determine key system parameters, including the 300-800 msg/s of text WebSocket message exchange, as well as results related to performance, stability, latency, and resource usage.