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 «БАКАЛАВР»

СИСТЕМА КЕРУВАННЯ РОБОТОМ НА БАЗІ ARDUINO ТА ФРЕЙМВОРКУ REACT

ROBOT CONTROL SYSTEM BASED ON ARDUINO AND REACT FRAMEWORK

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

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-42
Вращук Роман Сергійович

(підпис)

Керівник:
к.т.н., доцент
Бортник Катерина Яківна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 04 » червня 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Тарас ТЕРЛЕЦЬКИЙ
« 10 » 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Вращуку Роману Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Система керування роботом на базі Arduino та фреймворку React*

Керівник роботи *к.т.н., доц. Бортник Катерина Яківна*

затверджені наказом закладу вищої освіти від «04» січня 2025 року № 11/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 10.06.2025р.

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 та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз проблем та наявних рішень
Вибір апаратно-програмного забезпечення
Проєктування та реалізація системи
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Джерелом розробки є науково-технічна література з мікроконтролерних систем, документація на платформу Arduino, статті про реалізацію систем дистанційного керування

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області та наявних рішень	*Бортник К.Я., доцент*		
Вибір апаратно-програмного забезпечення	*Бортник К.Я., доцент*		
Проєктування та реалізація системи керування роботом	*Бортник К.Я., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,ст.викладач*		

7. Дата видачі завдання 10.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми, аналіз предметної області та наявних рішень*	до 10.02.2025 р.	Виконано
2.	*Аналіз предметної області та наявних рішень із подальшим вибором апаратно-програмного забезпечення*	до 02.03.2025 р.	Виконано
3.	*Проєктування та реалізація системи керування роботом*	до 02.04.2025 р.	Виконано
4.	*Висновки та пропозиції*	до 10.04.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 15.04.2025 р.	Виконано
6.	*Формування додатків*	до 02.05.2025 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 10.05.2025 р.	Виконано
8.	*Нормоконтроль*	до 15.05.2025 р.	Виконано
9.	*Представлення остаточного варіанту кваліфікаційної роботи бакалавра керівникові*	до 30.05.2025 р.	Виконано
10	*Інструментальна перевірка на академічний плагіат*	до 03.06.2025 р	Виконано
11.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Вращук Р.С.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Бортник К.Я.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Вращук Р. С. Система керування роботом на базі arduino та фреймворку react. Рукопис.

Кваліфікаційна робота бакалавра 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, додатку.

У першому розділі проведено аналіз предметної області та актуальності дистанційного керування роботизованими системами. Розглянуто наявні апаратні та програмні рішення, визначено проблематику та мету дослідження, обґрунтовано вибір архітектури.

У другому розділі здійснено вибір апаратної та програмної платформи, зокрема Arduino Uno, ESP32-CAM, а також фреймворків React і Node.js. Надано огляд сенсорів, механізмів зв'язку, описано середовище розробки і використані інструменти.

Третій розділ присвячено реалізації системи. Описано архітектуру взаємодії між компонентами, програмну логіку клієнта і сервера, а також алгоритми обробки команд, передачі даних, керування рухом і відеопотоком. Проведено тестування системи в реальних умовах, проаналізовано її стабільність і потенціал для подальшого розвитку.

Ключові слова: Arduino, ESP32, WebSocket, React, Node.js, дистанційне керування, роботизована платформа.

ANNOTATION

Vrashchuk R. Robot control system based on arduino and react framework. Manuscript.

Bachelor's qualification work of the EP "Computer Engineering", specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

The qualification work consists of an introduction, three chapters, conclusions, a list of references, and an appendix.

The first chapter presents an analysis of the subject area and the relevance of remote control systems for robotic platforms. Existing hardware and software solutions are reviewed, the research problem and goal are formulated, and the choice of system architecture is justified.

The second chapter describes the selection of hardware and software platforms, including Arduino Uno, ESP32-CAM, and the React and Node.js frameworks. It provides an overview of sensors, communication mechanisms, development environments, and the tools used during implementation.

The third chapter is devoted to the implementation of the system. It describes the architecture of component interaction, the client and server logic, algorithms for command processing, data transmission, robot motion control, and video streaming. The system was tested under real conditions, its stability was evaluated, and its potential for future development was analyzed.

Keywords: Arduino, ESP32, WebSocket, React, Node.js, remote control, robotic platform.

ЗМІСТ

ВСТУП

У сучасному світі зростає попит на інтелектуальні автоматизовані системи, зокрема роботи, які можуть дистанційно виконувати різноманітні завдання. В епоху цифровізації та Інтернету речей (IoT) з'являється потреба у розробці рішень, які поєднують фізичні пристрої з веб-інтерфейсами для керування. Одним із найперспективніших напрямів є реалізація систем дистанційного керування роботизованими платформами з використанням відкритих апаратних засобів, таких як Arduino, та сучасних фреймворків веброзробки – React і Node.js. Така система дозволяє здійснювати моніторинг та контроль у реальному часі, що особливо актуально в задачах автоматизації, навчання, охорони та досліджень.

Метою роботи є розробка системи дистанційного керування мобільним роботом з використанням платформи Arduino та вебтехнологій React і Node.js, що забезпечує інтерактивне керування, обробку сенсорних даних та відеоспостереження.

Об'єкт дослідження – вбудовані системи керування мобільними роботами.

Предмет дослідження – апаратно-програмна система дистанційного керування роботом на базі Arduino з використанням React і Node.js.

Завдання, які необхідно виконати:

− реалізувати вебінтерфейс для керування роботом та обробки даних

− сенсорів;

− розробити серверну частину з обробкою websocket-запитів та відеопотоку;

− дослідити можливості інтеграції камер esp32-cam та передачі відео;

− візуалізувати показники датчиків у реальному часі;

− спроектувати і протестувати загальну архітектуру системи;

− запропонувати варіанти вдосконалення системи для підвищення функціональності, надійності та зручності використання.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА НАЯВНИХ РІШЕНЬ

1.1 Актуальність і постановка проблеми

Стрімкий розвиток інформаційних технологій, зокрема напрямку Інтернету речей (IoT), призводить до зростання інтересу до інтерактивних роботизованих систем, якими можна дистанційно керувати через вебінтерфейси. Завдяки поширенню доступних апаратних платформ, таких як Arduino, та популярних фреймворків веброзробки, як-от React і Node.js, з'являється можливість створення власних систем управління, які поєднують простоту реалізації, функціональність та зручність використання [1].

Підвищення інтересу до створення автономних та напівавтономних роботизованих пристроїв обумовлено не лише навчальними цілями, а й практичними потребами – у логістиці, аграрній сфері, охороні об'єктів та моніторингу навколишнього середовища. Однак досі існує потреба у створенні комплексного, зручного у використанні та стабільного рішення, яке забезпечувало б взаємодію між користувачем та апаратною частиною через інтуїтивно зрозумілий вебінтерфейс із підтримкою відео та сенсорної інформації.

Приклад візуалізації напрямку Інтернету речей зображенно на рисунку 1.1.

Рисунок 1.1 – Візуальний приклад інтернету речей [1]

Однією з ключових переваг розробки такої системи є її універсальність і відкритість. Вона може бути масштабована або адаптована під різні типи роботизованих платформ. Крім того, з огляду на зростаючу популярність дистанційного навчання та STEM-освіти, подібні рішення можуть бути успішно використані в освітньому середовищі для ознайомлення студентів з принципами робототехніки, вебпрограмування та архітектурою клієнт-серверних застосунків.

Таким чином, тема кваліфікаційної роботи є актуальною як з науково-прикладної, так і з освітньої точки зору.

Незважаючи на наявність численних рішень у сфері керування роботами, більшість з них мають обмеження:

− відсутність масштабованості або гнучкої архітектури;

− складність у використанні або налаштуванні;

− низька інтеграція з сучасними вебтехнологіями;

− відсутність механізмів візуалізації та зворотного зв’язку.

Це зумовлює необхідність створення власної апаратно-програмної системи, яка забезпечить:

− взаємодію через браузер з підтримкою WebSocket;

− управління рухом робота та відеопотік з камери;

− виведення даних сенсорів у реальному часі;

− розділення логіки на клієнтську і серверну частину з використанням React і Node.js.

Метою роботи є створення системи дистанційного керування роботом з передачею відео та сенсорних даних, що реалізована на базі платформи Arduino з використанням фреймворків React і Node.js.

1.2 Аналіз апаратних рішень у системах дистанційного керування

Побудова системи дистанційного керування роботизованою платформою вимагає вибору надійного та доступного апаратного забезпечення, яке б

забезпечувало підтримку необхідних функцій: управління моторами, обробки сигналів від сенсорів, передавання даних у реальному часі та взаємодії з модулем бездротового зв’язку або камерою.

Серед доступних варіантів найбільшу популярність здобули платформи Arduino та ESP32, які вирізняються гнучкістю, великою спільнотою розробників, великою кількістю бібліотек, а також сумісністю з різноманітними сенсорами та модулями.

Платформа Arduino Uno на базі мікроконтролера ATmega328P має достатню кількість цифрових і аналогових входів/виходів для реалізації базових функцій управління рухом (керування двигунами, читання даних з датчиків відстані, світла, температури тощо). Вона також підтримується різними драйверами двигунів, як-от L298N або DRV8833, що дозволяє реалізувати двонаправлений контроль моторів.

Модуль ESP32-CAM, у свою чергу, забезпечує можливість підключення камери (OV2640) для передачі відео по мережі, а також має вбудований модуль Wi-Fi, що дозволяє реалізувати як пряме підключення, так і клієнт-серверну архітектуру. Його також можна використовувати як WebSocket-клієнт або міні-сервер для стріму MJPEG, зберігання даних або надсилання зображень.

У якості датчиків у проєкті можуть бути використані:

− HC-SR04 – ультразвуковий сенсор для вимірювання відстані;

− BME680 – для зчитування температури та вологості;

− MQ-7 – для аналізу якості повітря;

− IR-сенсори та фотодатчики – для навігації або детекції перешкод.

Важливо, що всі обрані компоненти підтримують живлення в межах 3.3-5 В та мають низьке енергоспоживання, що дає можливість створити автономну або мобільну платформу.

Таким чином, апаратна частина системи складається з:

− плати Arduino як центрального контролера керування мотором;

− модуля ESP32-CAM для відеоспостереження та зв’язку;

− набору сенсорів для зворотного зв’язку;

− джерела живлення (акумуляторів типу 18650) та драйвера моторів.

Це рішення забезпечує баланс між вартістю, функціональністю та простотою реалізації для створення прототипу системи дистанційного керування роботом.

1.3 Аналіз програмних рішень та вебінтерфейсів

У сучасних системах дистанційного керування надзвичайно важливо не лише забезпечити передачу команд, а й створити зручний та ефективний спосіб взаємодії користувача з пристроєм. Це завдання вирішується за рахунок програмної реалізації клієнтської та серверної частин, які разом формують повноцінну програмну архітектуру системи керування. Такі програмні рішення мають забезпечувати обробку команд у режимі реального часу, передачу відео- та сенсорних даних, збереження стабільності з’єднання, а також гарантувати зручність інтерфейсу для користувача.

Одним з основних викликів є забезпечення реактивної взаємодії між користувачем і роботом. Для цього доцільно використовувати двосторонню комунікацію, яка дозволяє не лише надсилати команди, але й отримувати підтвердження та дані зворотного зв’язку. Найпоширенішим рішенням для цього є WebSocket-протокол, що забезпечує постійне з’єднання між клієнтом (браузером) і сервером без необхідності багаторазового запиту (як це відбувається у класичному HTTP). WebSocket дозволяє реалізувати миттєву передачу команд, що критично важливо для керування рухом роботизованої платформи.

З боку серверної частини ідеальним інструментом для побудови легкої та масштабованої інфраструктури є Node.js – серверне середовище виконання JavaScript-коду. Воно дозволяє створювати одночасно обробку HTTP-запитів (наприклад, зміна налаштувань камери, логування подій) та WebSocket-зв’язок

для передачі команд і сенсорних даних. Перевагою Node.js є підтримка великої кількості підключень одночасно, висока швидкодія та величезна кількість готових бібліотек, що дозволяє прискорити розробку [2]. Візуальне зображення взаємодії HTTP з структурою проекту і всіма протоколами наведено на рисунку 1.2.

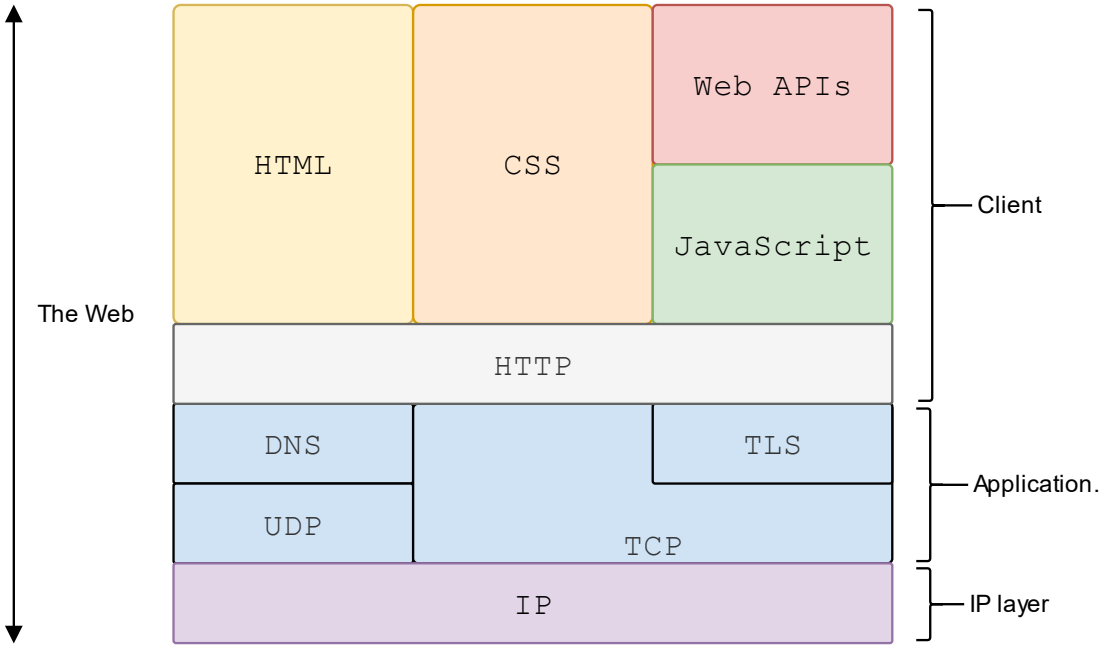

Рисунок 1.2 – Приклад взаємодії HTTP з структруою проекту [2]

З боку клієнта було використано React – популярну JavaScript-бібліотеку, яка забезпечує ефективне оновлення компонентів інтерфейсу без перезавантаження сторінки. Це дозволяє створювати інтерактивні панелі керування, де можна в реальному часі бачити стан системи, змінювати параметри, обирати режими роботи, а також переглядати відеопотік. React підтримує роботу з WebSocket, поділ на окремі компоненти та використання хуків для зручного керування станами додатку.

Існує низка схожих рішень, але більшість з них зосереджені або лише на роботі з Arduino, або лише на вебчастині без належної інтеграції. Наприклад.

Платформи на зразок Blynk або Tinkercad Circuits дозволяють симулювати прості зв’язки, але не дають повної гнучкості в UI та архітектурі.

Готові платформи типу Home Assistant орієнтовані на IoT і smart home, але вимагають значних ресурсів і не оптимізовані під мобільних роботів.

Таким чином, власна реалізація програмного середовища з використанням React і Node.js дає змогу створити цілком автономну систему, яка може працювати як у локальній мережі, так і через Інтернет (за наявності відповідної маршрутизації), що значно розширює можливості використання.

Особливу увагу в системі займає інтеграція ESP32-CAM, яка реалізує трансляцію зображення у вигляді MJPEG-потоку через HTTP-сервер. Камера дозволяє виводити відео в браузері, а також дає змогу надсилати зображення або використовувати їх у комп’ютерному зорі. Реалізація стріму вимагає мінімізації затримки, тому окрема увага приділяється буферизації, обмеженню FPS, контролю якості відео тощо.

Переваги обраного підходу:

− повна контрольованість логіки клієнт-серверної взаємодії;

− можливість масштабування або адаптації під інші апаратні платформи;

− використання сучасних фреймворків з відкритим кодом;

− гнучка архітектура, що дозволяє відділити логіку керування, виведення даних і трансляцію відео.

РОЗДІЛ 2
ВИБІР АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз апаратної платформи

У процесі розробки системи дистанційного керування роботом важливим є вибір апаратної платформи, яка поєднує доступність, достатню обчислювальну потужність, зручність для розробника та активну підтримку спільноти. Для забезпечення основних функцій – керування двигунами, обробки даних з сенсорів, отримання відеопотоку – доцільно поєднати дві незалежні апаратні платформи: Arduino (рис. 2.1) для обробки команд і сенсорів, та ESP32-CAM для відеоспостереження і передачі даних по Wi-Fi. Такий підхід дозволяє розділити завдання між двома контролерами та оптимізувати навантаження.

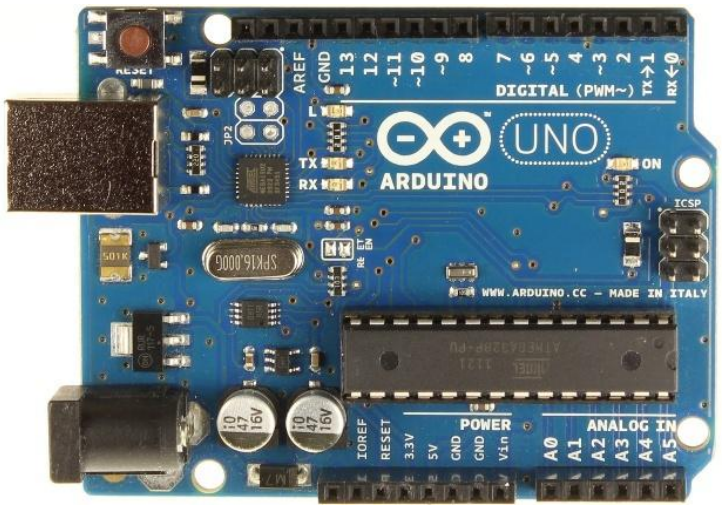

Рисунок 2.1 – Платформа Arduino Uno

Платформа Arduino Uno – це одна з найпопулярніших мікроконтролерних платформ, яка базується на мікросхемі ATmega328P (рис. 2.2). Вона має просту структуру, зручну у використанні середу розробки Arduino IDE та широку підтримку бібліотек. Зокрема, Arduino Uno має:

– 14 цифрових входів/виходів (з них 6 можуть використовуватись як ШІМ);

− 6 аналогових входів;

− послідовний порт (UART), через який здійснюється комунікація з ESP32 CAM;

− частоту 16 МГц і живлення 5 В.

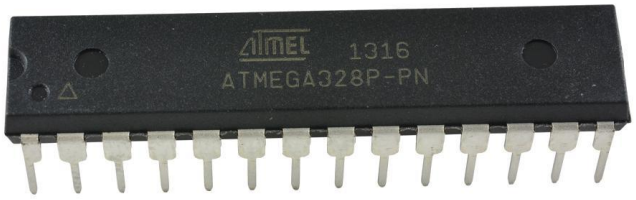

Рисунок 2.2 – Мікроконтролер ATmega328P

Перевагами Arduino є її надійність, простота прошивки, стабільна робота в реальному часі та низьке енергоспоживання. У межах цієї системи Arduino Uno використовується як головний виконавчий модуль, що:

− приймає команди з ESP32 через UART (Serial2);

− керує двигунами через драйвер (L298N або аналогічний);

− зчитує дані з ультразвукового сенсора, сенсорів світла, температури тощо.

Таким чином, Arduino виконує всі низькорівневі апаратні функції, ізолюючи їх від комунікаційної логіки.

Модуль ESP32-CAM – це плата, що поєднує можливості потужного 32-бітного мікроконтролера ESP32 із вбудованою камерою (OV2640), підтримкою Wi-Fi, Bluetooth, а також SD-картою.

Основні технічні характеристики ESP32-CAM:

− процесор з двома ядрами (240 МГц);

− до 520 КБ SRAM;

− підтримка розширеної пам’яті (PSRAM);

− роздільна здатність камери до 1600×1200 (UXGA);

− можливість трансляції MJPEG потоку;

− 9 GPIO-портів (спільних із функціями камери).

У проєкті ESP32-CAM виконує роль:

− HTTP-сервера для потокової передачі зображення з камери;

− WebSocket-клієнта для обміну командами між браузером і Arduino;

− контролера, що отримує сенсорні дані від Arduino та передає їх на фронтенд;

− пристрою для обробки запитів керування (зміна налаштувань, запуск режимів тощо).

Завдяки використанню ESP32-CAM (рис. 2.3) можливо забезпечити відеоспостереження у реальному часі, налаштування параметрів зображення (якість, яскравість, контраст), а також створити повноцінний вебінтерфейс із керуванням через локальну мережу [3].

Рисунок 2.3 – Модуль ESP32-CAM

Для забезпечення зворотного зв’язку, автоматизації частини функцій та підвищення адаптивності системи керування до умов навколишнього середовища до складу роботизованої платформи було включено низку сенсорних та допоміжних пристроїв. Сенсори відіграють важливу роль у прийнятті рішень

під час руху робота, а також у збиранні даних, які можуть бути корисними для користувача або подальшої обробки.

Приведу основні види сенсорів які використав у своїй роботі.

HC-SR04 – ультразвуковий датчик відстані (рис. 2.4), що дозволяє визначати перешкоди на шляху руху робота. Він працює за принципом відбиття ультразвукової хвилі, вимірюючи час повернення сигналу. Дані з нього використовуються для побудови умовної карти навколишнього середовища або реалізації функцій автоматичної зупинки.

Рисунок 2.4 – HC-SR04

BME680 – цифрові сенсор температури та вологості (рис. 2.5). Він дозволяє зчитувати метеоумови навколо платформи, що може бути корисним у дослідницьких або навчальних цілях та навіть обчислення якості індексу повітря IAQ. BME680 має кращу точність і ширший діапазон, тому в проєкті може застосовуватись саме він [4].

Рисунок 2.5 – Bosch BME 680

MQ-7 – газовий сенсор для виявлення чадного газу (рис. 2.6), летких органічних речовин. Такий сенсор дозволить перетворити мобільногоробота на мобільну станцію екологічного моніторингу.

Рисунок 2.6 – MQ-7

Фотодатчик – застосовуються для виявлення освітленості (рис. 2.7), ліній на поверхні (наприклад, в задачах line following) або наближення до об'єктів.

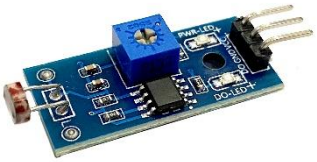

Рисунок 2.7 – Фотодатчик

Звуковий сенсор – може використовуватися для виявлення гучних подій або простого реагування на клацання/свист/голос (рис. 2.8).

Рисунок 2.8 – Звуковий сенсорї

Окрім сенсорів, до периферійних пристроїв відносять.

Драйвер двигунів L298N або DRV8833 – модуль, що дозволяє здійснювати двонаправлене керування двома двигунами з можливістю регулювання швидкості (через ШІМ-сигнали).

Сервомеханізм SG90 – використовується для зміни кута огляду камери або інших елементів, які потребують точного позиціювання.

Акумуляторний блок з BMS – живить обидві плати та виконавчі механізми, забезпечуючи захист від перенапруги та перегріву.

Загальна структура сенсорної підсистеми будується на Arduino, яке зчитує аналогові та цифрові значення, попередньо обробляє їх (наприклад, усередненням або перевіркою порогів), після чого передає у вигляді рядків на ESP32 через UART. ESP32, у свою чергу, передає ці значення у вебінтерфейс, де вони виводяться в зручному для користувача форматі.

Таким чином, поєднання базових сенсорів з додатковими периферійними модулями забезпечує не лише реалізацію базового керування, а й надає системі потенціал для виконання розширених функцій моніторингу та аналізу зовнішнього середовища.

2.2 Аналіз програмного середовища

Окрім апаратної складової, важливою частиною побудови системи дистанційного керування є програмне забезпечення, яке забезпечує зв’язок між користувачем і фізичним пристроєм. Саме за рахунок правильної побудови програмного середовища досягається функціональність, зручність, надійність та масштабованість системи. У даній роботі акцент зроблено на використанні відкритих вебтехнологій, які дозволяють реалізувати інтуїтивно зрозумілий інтерфейс з одночасною обробкою команд і передачею даних у реальному часі.

Програмне середовище складається з трьох основних компонентів:

− каналу двосторонньої комунікації (websocket), який забезпечує миттєвий

обмін повідомленнями між клієнтським вебінтерфейсом та сервером, а також між сервером і апаратною частиною;

− фронтенду, реалізованого на фреймворку react, що відповідає за візуалізацію інтерфейсу користувача, оновлення станів сенсорів, відображення відео з камери, кнопки керування та вхідні події (натискання клавіш, геймпад тощо);

− серверної частини на node.js, яка виступає як посередник між браузером та пристроєм esp32-cam, обробляючи websocket-з'єднання, post-запити та інші дії керування, включно з логікою автоматичного слідування або обробкою координат розпізнаних об'єктів.

Таке розділення дозволяє реалізувати архітектуру клієнт-сервер, де взаємодія з апаратною частиною є ізольованою від фронтенду, що полегшує підтримку, масштабування і відлагодження проєкту. Кожна з програмних складових має чітко визначене призначення і працює у тісній взаємодії з іншими частинами системи.

Для реалізації проєкту також використовуються супутні бібліотеки, зокрема:

− socket.io – для спрощеної роботи з WebSocket у Node.js та React;

− express – для побудови REST API та обробки запитів до сервера;

− tensorflow.js – для клієнтського розпізнавання об'єктів (опціонально).

Розгортання програмного середовища можливе як у локальній мережі (LAN), так і в глобальному середовищі (через проксі або тунельні сервіси), що робить систему універсальною та адаптивною до умов використання.

2.2.1 WebSocket як засіб двостороннього зв'язку

У традиційних вебзастосунках взаємодія між клієнтом і сервером реалізується через протокол HTTP, де кожна взаємодія вимагає окремого запиту і відповіді. Такий підхід неефективний у випадках, коли потрібна постійна синхронізація даних або миттєве реагування системи на дії користувача. У системах дистанційного керування, де важлива мінімальна затримка та

безперервний обмін інформацією, доцільно використовувати протокол WebSocket, який забезпечує постійне двостороннє з’єднання між клієнтом і сервером.

WebSocket – це мережевий протокол, який дозволяє встановити повнодуплексний канал поверх одного TCP-з’єднання (рис. 2.9). Після одноразового «рукостискання» (handshake), яке ініціюється через HTTP, відкривається постійне з’єднання, через яке сторони можуть обмінюватись повідомленнями у форматі JSON, рядків або бінарних даних без повторних запитів [5].

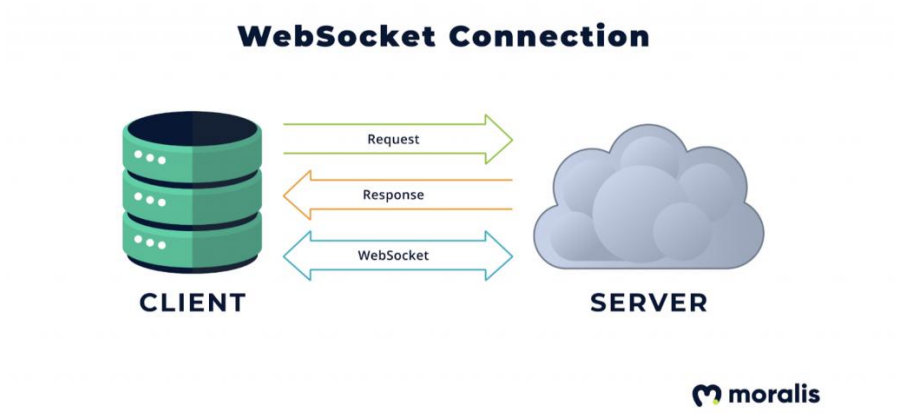

Рисунок 2.9 – WebSocket [5]

У межах даної роботи WebSocket використовується для:

− надсилання команд з браузера (клавіатура, геймпад) на сервер;

− передачі команд з сервера до ESP32;

− отримання сенсорних даних з ESP32;

− обміну службовою інформацією (наприклад, стан підключення, ping/pong).

Завдяки WebSocket реалізована низька затримка в передачі команд (менше 50 мс у локальній мережі), що дозволяє здійснювати плавне та точне керування рухом робота. Також WebSocket зменшує мережеве навантаження порівняно з постійними AJAX-запитами, оскільки з’єднання встановлюється лише один раз і підтримується доти, доки клієнт не закриє сторінку або не розірве з’єднання.

На сервері Node.js для обробки WebSocket-з’єднань використовується бібліотека socket.io, яка спрощує підключення, обробку подій (on, emit, broadcast), обробку відключень, обмін ID-клієнтів, а також дозволяє легко масштабувати проєкт.

Крім того, протокол WebSocket використовується і між ESP32 та сервером, або безпосередньо між ESP32 і клієнтом (у разі використання лише однієї точки обробки). Такий підхід дозволяє будувати як прямі моделі взаємодії, так і багатоступеневі (з проміжним сервером), що робить архітектуру більш гнучкою.

Загалом використання WebSocket стало критично важливим компонентом у реалізації цієї системи, оскільки саме цей протокол забезпечує основну комунікацію між усіма її частинами в режимі реального часу.

2.2.2 Фреймворк React для створення інтерфейсу

Інтерфейс користувача – це ключовий елемент взаємодії з системою дистанційного керування. Саме через нього користувач надсилає команди, спостерігає за рухом робота, отримує сенсорні дані та здійснює контроль над параметрами системи. Для створення сучасного, динамічного та масштабованого вебінтерфейсу в рамках цієї роботи було використано React – популярну бібліотеку JavaScript для побудови односторінкових застосунків (рис. 2.10).

Рисунок 2.10 – Приклад React коду [6]

React дозволяє реалізувати компонентний підхід, тобто розділити інтерфейс на незалежні частини (наприклад, панель керування, відеопотік, блок сенсорів, лог команд), кожна з яких оновлюється окремо без перезавантаження сторінки. Це забезпечує:

− високу швидкість роботи інтерфейсу;

− зменшення навантаження на сервер;

− легке тестування та модифікацію окремих компонентів.

Основні функції, реалізовані в інтерфейсі:

− панель керування – кнопки керування рухом (вперед, назад, ліво, право), зупинка, повороти, зміна кута огляду камери, регулювання швидкості;

− підключення геймпада – за допомогою api gamepad браузер отримує значення натискань і рухів аналогових стіків, які перетворюються у команди для робота;

− передача команд через websocket – інтегрований клієнт websocket, який дозволяє миттєво передавати дії користувача до сервера;

− відображення сенсорних даних – температура, вологість, відстань, освітленість та інші параметри, що надходять від esp32 через сервер;

− радар або міні-карта – візуалізація даних з ультразвукового сенсора у вигляді анімованого сканера;

− блок налаштувань камери – зміна параметрів зображення: яскравість, контраст, роздільна здатність, fps.

React також дозволяє працювати з хуками, зокрема useEffect (для ініціалізації WebSocket-з’єднання або відстеження змін стану) та useState (для оновлення значень сенсорів, відображення повідомлень тощо) [6].

Для стилізації інтерфейсу може використовуватись як звичайний CSS, так і бібліотеки стилів (наприклад, ShadCN UI або Bootstrap). У рамках цієї роботи зроблено акцент на чисту верстку з використанням сучасних гнучких сіток (flex, grid) та адаптивного дизайну.

Крім цього, React-інтерфейс може бути легко розширений новими модулями, такими як:

− розпізнавання об’єктів (через TensorFlow.js);

− авторизація користувача;

− збереження логів керування;

− підтримка мобільних пристроїв.

Завдяки використанню React система стає гнучкою, динамічною та розширюваною, а її візуальна частина легко адаптується під нові задачі та потреб користувача.

2.2.3 Node.js як серверна платформа

Node.js (рис. 2.11) є сучасним середовищем виконання JavaScript-коду на стороні сервера, яке побудоване на основі V8 – високопродуктивного рушія Google Chrome. Його ключовою особливістю є асинхронна обробка подій, що дозволяє ефективно обслуговувати велику кількість одночасних з’єднань із мінімальними затримками. Саме ця властивість робить Node.js оптимальним вибором для створення серверної частини в системах дистанційного керування реального часу [7].

Рисунок 2.11 – Приклад Node.js коду [7]

У межах проєкту Node.js виконує центральну координаційну функцію між клієнтським інтерфейсом, мікроконтролером ESP32-CAM і, опціонально, з

іншими джерелами або сервісами (наприклад, для обробки координат об’єкта з клієнта). Серверна частина була реалізована із застосуванням таких основних технологій:

Бібліотека socket.io – для забезпечення WebSocket-зв’язку з React-клієнтом та, за потреби, з ESP32. Це дозволяє реалізувати двосторонню передачу даних: клієнт сервер ESP32 і навпаки. Наприклад, команду %F# (рух вперед) можна передати з браузера на Arduino менш ніж за 30 мс.

Бібліотека express – для реалізації REST API. Через HTTP-запити можливо змінювати параметри системи, надсилати координати для автостеження, отримувати статуси тощо. Це зручно для інтеграції з іншими модулями або навіть зовнішніми програмами.

Обробка логіки керування – Node.js реалізує централізовану логіку: визначає, які команди від користувача надходять, які потрібно перенаправити до ESP32, обробляє помилки, втрату з’єднання, синхронізує поточний стан системи. Наприклад, при зникненні цілі (у режимі слідування) сервер ініціює зупинку руху, оновлює інтерфейс і передає новий стан назад на клієнт.

Механізм відстеження з’єднань – реалізовано постійний «ping/pong» обмін між клієнтом і сервером, а також сервером і ESP32 для контролю активності. У разі обриву зв’язку вживаються заходи: відключення потоку, повторне з’єднання, інформування користувача.

Формат повідомлень – система працює з простими командами у форматі %команда#, що полегшує парсинг на стороні Arduino та зменшує обсяг переданих даних.

Node.js також дозволяє зручно масштабувати систему: наприклад, додати реєстрацію користувачів, зберігання журналу дій, інтеграцію з хмарними сервісами, авторизацію через токени або проксирування запитів на кілька пристроїв.

Сервер можна розгорнути як на локальному комп’ютері розробника, так і на віддаленому сервері (наприклад, на Raspberry Pi, VPS або через Docker), що забезпечує гнучкість і портативність рішення.

Таким чином, Node.js виступає надійним комунікаційним та логічним ядром системи, забезпечуючи стабільну роботу між усіма компонентами: клієнтом, пристроєм і зовнішніми сервісами.

2.3 Вибір середовища розробки та інструментів

Успішна реалізація проєкту з розробки системи дистанційного керування роботом вимагає використання набору зручних, продуктивних та сумісних між собою середовищ розробки й інструментів. Правильний вибір середовища значно пришвидшує процес розробки, підвищує якість коду та спрощує налагодження як апаратної, так і програмної частини системи.

Для написання та завантаження скетчів на мікроконтролер Arduino Uno використовується Arduino IDE, який зображено на рисунку 2.12.

Рисунок 2.12 –Структура Arduino IDE коду [8]

Це середовище має інтуїтивно зрозумілий інтерфейс, підтримує велику кількість бібліотек для сенсорів та драйверів двигунів і дозволяє безпосередньо взаємодіяти з COM-портами для моніторингу серійного виводу [8].

Ключові особливості:

− можливість завантаження коду без додаткових налаштувань;

− серійний монітор для перевірки даних UART;

− підтримка бібліотек: Servo.h, NewPing.h, SoftwareSerial, Adafruit_Sensor, DHT, MQUnifiedsensor.

Для програмування ESP32-CAM також використовується Arduino IDE з попередньо встановленою підтримкою ESP32 (через менеджер плат). Додатково потрібні бібліотеки для камери (esp_camera.h), WebSocket-клієнта (WebSocketsClient.h) і Wi-Fi.

Для налагодження ESP32 особливо важливо використовувати:

− автозавантаження через GPIO0;

− зовнішній USB-UART перехідник;

− debug-вивід через Serial або Serial2.

Також доцільним є використання PlatformIO (у середовищі VS Code) для більш складних проєктів, однак у даній роботі було зроблено акцент на Arduino IDE через її простоту.

Для серверної логіки використовувалось середовище Node.js, а розробка велась у WebStorm – потужному IDE (рис. 2.13) від JetBrains, яке має глибоку інтеграцію з JavaScript, Node.js і npm [9]. WebStorm надає такі переваги:

− інтелектуальне автодоповнення коду;

− відлагодження та профілювання HTTP-запитів;

− вбудована підтримка Git;

− зручна робота з терміналом та npm-проєктами;

− автоматичне форматування та перевірка коду.

Усі залежності (express, socket.io, cors, body-parser) встановлювались через npm, а сервер запускався локально з використанням інтегрованого терміналу WebStorm.

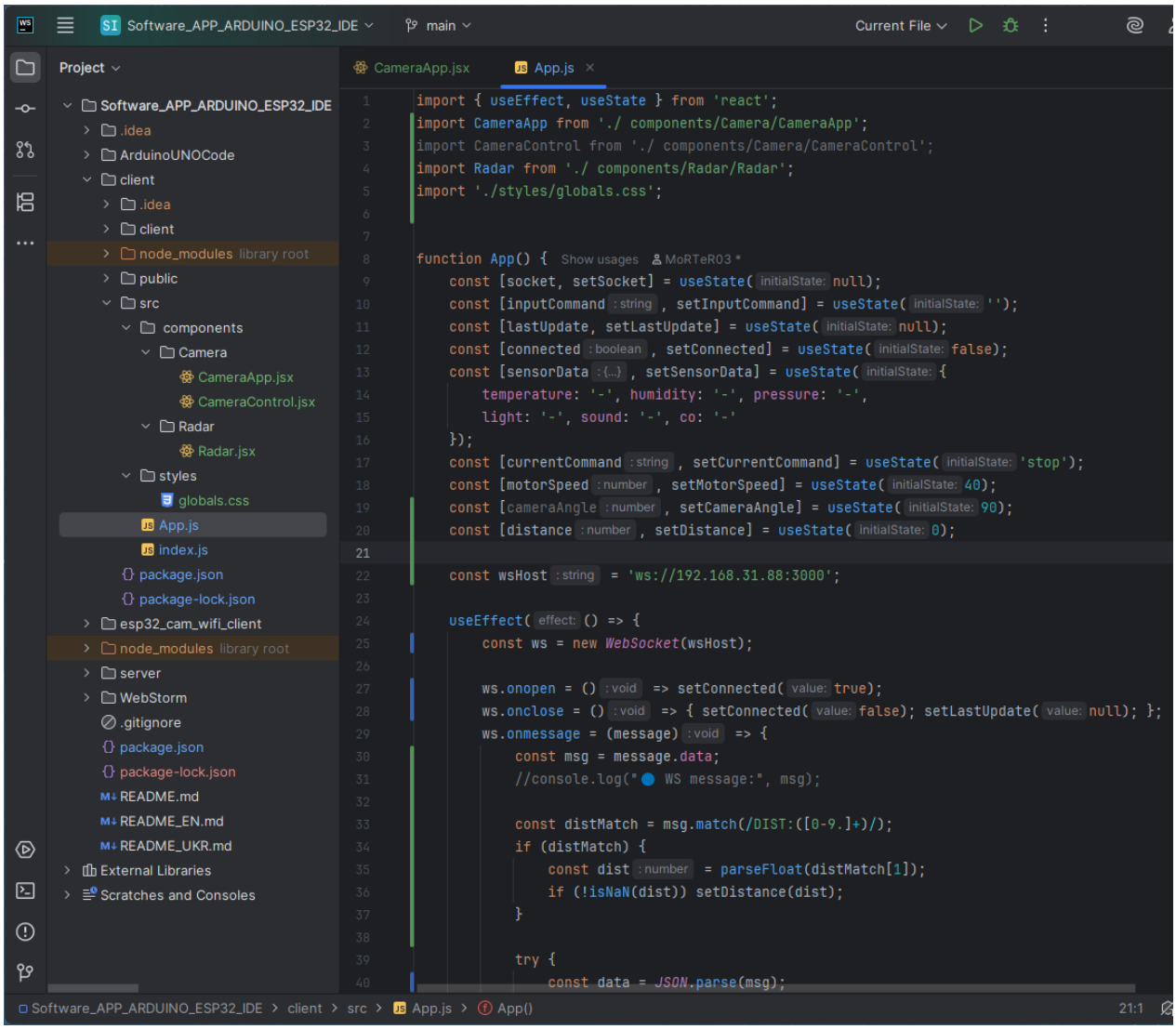

Рисунок 2.13 –Структура коду в WebStorm IDE

Фронтенд-розробка також велась у середовищі WebStorm. Проєкт створено за допомогою Create React App, що забезпечує модульну структуру, автоматичне збирання, підтримку JSX і хуків.

WebStorm забезпечив:

– зручну навігацію між компонентами;

– підтримку роботи з React-specific розширеннями;

− автоматичне оновлення сторінки під час розробки (npm start);

− візуальне відображення структури компонентів і стилів.

Можна зазначити додаткові інстрменти, які допомогли мені в роботі над проєктом:

− Git – для контролю версій та збереження прогресу;

− Fritzing / Tinkercad – для візуального проєктування схем підключення;

− ESP32 Flash Download Tool – для глибокої роботи з ESP32 у разі помилок прошивки;

− Line Notify (опціонально) – для надсилання зображень з камери в месенджери.

Завдяки поєднанню цих середовищ та інструментів була реалізована повноцінна, стабільна та масштабована система, яка охоплює всі етапи – від низькорівневого керування двигунами до клієнтського інтерфейсу з виведенням відео та сенсорних даних.

РОЗДІЛ 3
ПРОЄКТУВАННЯ І РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ РОБОТОМ

3.1 Архітектура та алгоритми роботи системи

Система дистанційного керування роботом побудована за принципом розподіленої клієнт-серверної архітектури, у якій функціональність чітко розмежована між трьома логічними рівнями: апаратним (Arduino, ESP32, сенсори, виконавчі механізми), серверним (Node.js) та клієнтським (React). Такий підхід забезпечує модульність, простоту супроводу, можливість незалежного тестування та гнучке масштабування системи.

Кожен із компонентів виконує свою визначену роль. Клієнтська частина, реалізована за допомогою React, відповідає за взаємодію з користувачем. Інтерфейс надає доступ до керування рухом робота, зміни кута камери, регулювання швидкості, перемикання режимів, а також відображає відеопотік і сенсорні дані. Керування можливе як з клавіатури, так і з геймпада, а всі команди миттєво відправляються на сервер через WebSocket-з’єднання. Крім того, клієнт може надсилати координати для режиму автослідування, а також змінювати параметри камери в реальному часі.

Сервер, побудований на Node.js, виконує функцію логічного посередника між клієнтом і апаратною частиною. Він приймає команди, обробляє їх, виконує маршрутизацію до відповідних пристроїв (ESP32 або Arduino), слідкує за стабільністю зв’язку, відправляє сенсорні дані у вебінтерфейс, а також керує поведінкою робота у разі втрати об'єкта або обриву з'єднання. Крім WebSocket, сервер також обробляє HTTP-запити, зокрема для прийому координат (/api/coords) та зміни режимів.

ESP32-CAM виконує одразу кілька функцій: транслює відео у форматі MJPEG через HTTP-потік, приймає та перенаправляє команди, які надходять із сервера, слухає UART-з’єднання з Arduino, а також передає на сервер сенсорні дані. Крім того, він приймає налаштування камери (контраст, якість, FPS тощо)

від клієнта і застосовує їх у реальному часі, призупиняючи трансляцію на час зміни параметрів.

Arduino Uno є головним виконавчим елементом. Він зчитує команди, що надходять через UART від ESP32, інтерпретує їх і керує моторами, сервомеханізмами та іншими виконавчими пристроями. До нього також підключено сенсори, зокрема HC-SR04, MQ-7, фотодатчики тощо. Arduino періодично зчитує значення з усіх сенсорів, формує відповідь у вигляді текстового рядка або JSON-структури і передає її на ESP32, звідки вона потрапляє у вебінтерфейс користувача.

Загальний алгоритм взаємодії системи виглядає наступним чином: користувач активує дію в інтерфейсі (наприклад, рух вперед), React формує відповідну команду у форматі %F100# і надсилає її через WebSocket на сервер. Node.js отримує цю команду, обробляє її та передає на ESP32, який ретранслює її на Arduino. Мікроконтролер виконує дію, а паралельно зчитує значення сенсорів і передає їх назад тим самим шляхом — через ESP32, сервер і WebSocket у React. Одночасно з цим браузер відображає потік з камери ESP32 у реальному часі через тег <img>, що підвантажує MJPEG-відео за HTTP. Якщо користувач увімкнув режим автослідування, то координати об’єкта надсилаються серверу, де відбувається логічна обробка і приймається рішення щодо руху або повороту камери.

Таким чином, система побудована як узгоджена взаємодія незалежних компонентів, які обмінюються інформацією у режимі реального часу, забезпечуючи стабільне дистанційне керування, адаптивність та можливість подальшого розширення. Кожен з модулів може бути вдосконалений або замінений без порушення загальної логіки роботи, що робить архітектуру гнучкою й довготривало підтримуваною.

3.1.1 Алгоритми обробки команд та даних

Для системи дистанційного керування важливою є здатність оперативно реагувати на команди користувача незалежно від типу пристрою введення. У

реалізованій системі передбачено підтримку керування як з клавіатури, так і з геймпада. Команди, що надходять з клієнтської частини, передаються на сервер, далі на ESP32, і вже звідти на Arduino, який виконує конкретні дії. Уся система побудована таким чином, щоб забезпечити мінімальну затримку, стабільність передачі та логічну розділеність відповідальності між модулями.

Для передачі команд використовується уніфікований текстовий протокол, зручний для обробки без потреби складного парсингу. Формат повідомлень виглядає як %<КОД_КОМАНДИ><ПАРАМЕТР>#. Наприклад, %F100# означає рух вперед зі швидкістю 100, %S# – повна зупинка, %C120# – зміна кута камери, %V80# – оновлення глобального параметра швидкості. Такий підхід дозволяє легко інтерпретувати дані як на сервері, так і безпосередньо на мікроконтролері Arduino.

На стороні клієнта реалізовано обробку введення з клавіатури та геймпада. За допомогою стандартного API браузера зчитуються натискання кнопок (W, A, S, D тощо) або зміщення стіків та курків на контролері. Після аналізу введення формується текстова команда, яка передається через WebSocket на сервер. У React ця логіка реалізується у вигляді хуків useEffect, useRef та setState, що дозволяє динамічно оновлювати інтерфейс і реагувати на зміни без перезавантаження сторінки.

Node.js-сервер виконує функцію обробника усіх команд. Він отримує повідомлення через події socket.on, валідує формат, визначає напрямок команди (ESP32 або Arduino), надсилає її за потрібним каналом і при потребі – активує таймер зупинки. Ця логіка дозволяє реалізувати функції автозупинки у разі втрати сигналу, затримки відповіді або закінчення режиму слідування.

Модуль ESP32-CAM виступає як проміжний вузол, що поєднує WebSocket-сервер, камеру, а також UART-зв’язок з Arduino. Він приймає команди від Node.js, передає їх Arduino через Serial2, а також прослуховує UART на предмет надходження зворотних повідомлень. Прийняті рядки, якщо вони мають правильну структуру, передаються назад на сервер, а той – на клієнт.

Arduino Uno є виконавчим компонентом, який обробляє командні рядки у вигляді %F100# або %C90#, аналізуючи код дії та параметр. Залежно від типу команди мікроконтролер запускає або зупиняє мотори, змінює ШІМ-сигнали, керує сервомотором для регулювання кута камери. У циклі loop() Arduino також опитує сенсори, формує відповідь у вигляді текстового рядка або JSON-структури (рис. 3.1) і передає її назад на ESP32.

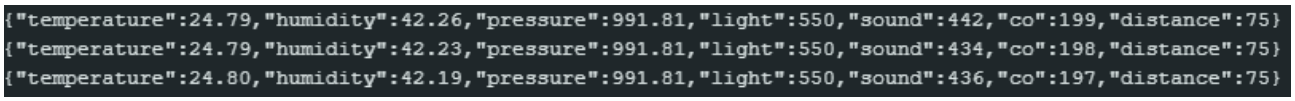

Рисунок 3.1 – JSON-структура у моніторі задач Arduino

Зворотній зв’язок у системі реалізовано повноцінно. Дані з сенсорів передаються від Arduino через ESP32 до сервера, і з нього – в React. Завдяки використанню WebSocket оновлення значень на інтерфейсі відбувається майже миттєво. Якщо сенсорна інформація не надходить протягом визначеного періоду, клієнтська частина змінює відображення – наприклад, виводить «–» або повідомлення про втрату з’єднання. Для покращення надійності система використовує також таймери затримки, логування втрат кадрів, а у критичних ситуаціях – автоматичну зупинку.

Загалом така архітектура обробки команд і зворотного зв’язку демонструє надійність, масштабованість і адаптивність. Вона дозволяє розширювати функціональність системи за рахунок додавання нових команд, сенсорів або режимів без потреби у суттєвому переписуванні коду на стороні сервера чи клієнта.

3.1.2 Взаємодія з сенсорами

Сенсорна система є ключовим елементом проєкту, адже саме вона забезпечує зворотний зв’язок між роботом і навколишнім середовищем. Завдяки цьому користувач отримує інформацію про температуру, вологість, наявність перешкод, рівень газів, а також інші фактори, що дозволяє не лише візуалізувати

ці параметри у вебінтерфейсі, а й реалізувати автоматичні дії, наприклад, зупинку перед перешкодою або автокорекцію положення камери.

До системи інтегровано кілька типів сенсорів, які взаємодіють із мікроконтролером Arduino Uno. Найбільш активно використовується ультразвуковий датчик HC-SR04, що дозволяє визначати відстань до об’єктів шляхом вимірювання часу відбиття сигналу. Його значення відображаються у вигляді цифрового числа або у формі візуального радара. Для вимірювання кліматичних показників використовується сенсор BME680, який забезпечує точні дані температури та вологості з інтервалом оновлення кожні кілька секунд. Його значення передаються у JSON-форматі й виводяться у відповідному блоці на інтерфейсі. Для більш складного аналізу атмосфери використовуються сенсори так само BME680 і MQ-7, вони здатні виявляти чадний газ, леткі органічні речовини, а також визначати тиск і загальний індекс якості повітря. Додатково до системи можуть бути підключені ІЧ-сенсори, фотодатчики або мікрофони для виявлення об'єктів збоку, звукових подій чи контролю освітленості.

Передача даних сенсорів побудована на простій, але надійній моделі: Arduino зчитує значення в основному циклі loop() і формує з них JSON-рядок, наприклад { "temp": 22.5, "hum": 48, "dist": 76 }. Цей рядок передається через UART на ESP32, який, у свою чергу, пересилає його серверу Node.js. Сервер не виконує складної обробки, а лише перевіряє формат і перенаправляє дані через WebSocket на клієнт. У React-інтерфейсі значення автоматично відображаються у відповідних компонентах, що забезпечує безперервний зворотний зв’язок з користувачем.

У системі передбачено обробку можливих збоїв. Якщо повідомлення не відповідає формату JSON або є пошкодженим, воно ігнорується. У разі відсутності нових даних протягом визначеного часу (2-3 секунди) відповідні поля в інтерфейсі стають сірими, а користувач бачить повідомлення про втрату з’єднання. Для критичних випадків, як-от коли відстань до перешкоди менше 15

см, система автоматично надсилає команду на зупинку руху, навіть якщо користувач продовжує натискати кнопку руху вперед.

Важливою перевагою такого підходу є простота й масштабованість: усі дані передаються у вигляді звичайного текстового JSON, а клієнтська логіка побудована так, що може автоматично оновлювати відображення нових типів сенсорів без зміни загальної структури коду. Крім цього, реалізація WebSocket-зв’язку дозволяє отримувати дані без затримок і без потреби у постійних HTTP-запитах, що суттєво знижує навантаження на сервер. У перспективі сенсорні дані можуть бути збережені або передані для подальшої обробки – наприклад, аналізу трендів або виводу на зовнішні аналітичні панелі.

Таким чином, реалізована модель сенсорного зворотного зв’язку повністю відповідає потребам інтерактивного керування. Вона дозволяє користувачу ефективно орієнтуватись у реальному середовищі робота, приймати рішення на основі об’єктивних показників або делегувати частину логіки автоматичним алгоритмам.

3.2 Апаратна реалізація та підключення

Реалізація фізичної частини системи починається з побудови повноцінної апаратної архітектури, яка включає в себе модуль керування, виконавчі пристрої, систему сенсорів, камеру та живлення. Головною керуючою одиницею виступає мікроконтролер Arduino Uno, який відповідає за обробку команд, керування моторами та зчитування даних з підключених сенсорів. Цей контролер було обрано завдяки його простоті, надійності та наявності великої кількості бібліотек, що прискорює розробку. До Arduino під’єднано драйвер двигунів L298N, який дозволяє здійснювати двонаправлене керування двома колекторними моторами з можливістю регулювання швидкості через широтно-імпульсну модуляцію (ШІМ). Самі мотори розміщено на компактній мобільній платформі, яка забезпечує маневреність та стійкість робота.

Приклад підключення основних переферійних пристроїв до Arduino UNO зображено на рисунку 3.2.

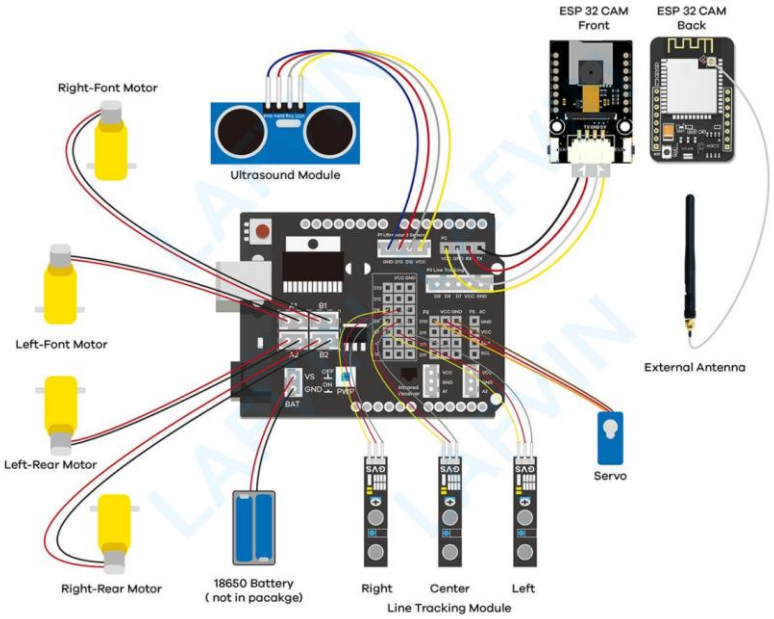

Рисунок 3.2 – Приклад підключення основних переферійних пристроїв

Камера та модуль зв’язку реалізовані через ESP32-CAM, який виконує функції збирання зображення, передавання відео по Wi-Fi, а також ретрансляції команд до Arduino через UART. Між Arduino та ESP32-CAM встановлено послідовне з’єднання за допомогою Serial2, де ESP32 виконує роль посередника, що передає команди від сервера на Arduino, і зчитує зворотну інформацію про стан системи чи значення сенсорів.

До Arduino було підключено датчики: ультразвуковий HC-SR04 для визначення відстані до об’єктів перед роботом, температурно-вологісний BME680 для моніторингу навколишнього середовища, та опціонально сенсор CO MQ-7, який може використовуватись для аналізу забруднення повітря. Всі сенсори живляться від 5 В через Arduino, і зчитуються з використанням цифрових або аналогових входів.

Живлення системи реалізовано через акумуляторний блок на базі літієвих елементів 18650 (рис. 3.3), з’єднаних послідовно. Для захисту акумуляторів

використовується BMS-модуль (Battery Management System), що забезпечує контроль напруги на кожній комірці, а також запобігає перезарядженню та надмірному розряду. Додатково в ланцюг живлення включено понижувальний стабілізатор напруги для захисту ESP32-CAM, який потребує 3.3 В живлення.

Рисунок 3.3 – Літієвий елемент 18650

З’єднання між компонентами реалізовано за допомогою макетної плати, клемних з’єднувачів або пайки. Для створення електричної схеми підключення використовувався Fritzing, що дозволило наочно відобразити топологію системи. Особливу увагу було приділено правильному розведенню маси між ESP32 та Arduino, щоб уникнути шумів у сигнальних лініях UART.

Кріплення камери ESP32-CAM реалізоване на сервомоторі SG90, що дозволяє змінювати кут огляду по вертикалі. Це дає змогу в режимі слідування автоматично піднімати або опускати камеру залежно від положення цілі у кадрі, що покращує стабільність відстеження.

Загалом апаратна реалізація була виконана з урахуванням розділення функціональності між модулями, що дозволило спростити налагодження та забезпечити стабільну роботу всієї системи. Кожен компонент може бути замінений або вдосконалений без суттєвої перебудови всього пристрою.

Побудова електричних з’єднань між модулями є критичним етапом у фізичній реалізації проєкту. Всі компоненти були пов’язані в єдину систему з

урахуванням номіналів напруги, логічних рівнів, допустимих струмів та особливостей живлення. Arduino Uno, як основний виконавчий модуль, забезпечує живлення для більшості сенсорів. Плата ESP32-CAM потребує живлення у 3.3 В, тому окремо для неї використовується стабілізатор типу AMS1117 або понижувальний модуль типу LM2596.

UART-з'єднання між ESP32 і Arduino виконано за допомогою TX-RX пар із перехресним підключенням, при цьому важливо було уникнути конфлікту логічних рівнів, оскільки ESP32 працює на 3.3 В, а Arduino – на 5 В. Для вирівнювання рівнів використовувався простий резистивний подільник або логічний перетворювач. Датчики відстані та температури підключені до цифрових і аналогових пінів Arduino відповідно до рекомендацій виробника.

Для керування моторами було використано драйвер L298N, який підключено до PWM-виходів Arduino. Живлення мотора подається окремо, а керування здійснюється цифровими пін-напрямками. Камера ESP32-CAM кріпиться на сервомоторі SG90, який керується через один PWM-пін Arduino. Для перевірки правильності підключення використовувались макетна плата, мультиметр, а також тестовий скетч, що дозволяє перевірити кожен окремий модуль перед повним збиранням конструкції.

Програмне забезпечення для Arduino та ESP32-CAM розроблялось у середовищі Arduino IDE із застосуванням бібліотек для сенсорів, WebSocket, UART та керування сервомоторами. На стороні Arduino головною задачею є приймання команд з ESP32, інтерпретація їх у дії (рух, зупинка, поворот, зміна кута камери), зчитування сенсорних даних та формування зворотного повідомлення. Команди обробляються у форматі простого парсингу: перевірка початку символом % і закінчення символом #, після чого виконується дія згідно вказаного коду.

Програма має структуру, наближену до стану автомату, де в основному циклі loop() перевіряється наявність нових UART-команд, зчитуються сенсори, оновлюється стан двигунів. Для уникнення блокувань було мінімізовано

використання функції delay(), натомість застосовано millis() для відслідковування часу. Окремо реалізовано буфер прийому даних із ESP32, що дозволяє обробляти як JSON-рядки, так і прості текстові команди.

На стороні ESP32-CAM основним завданням є запуск вебсерверу для трансляції відеопотоку та підтримка WebSocket-з'єднання з сервером. Крім того, мікроконтролер постійно слухає UART і надсилає дані у WebSocket клієнтам, виконуючи роль комунікаційного шлюзу між апаратною та вебчастиною.

Процес налагодження апаратної частини включав кілька етапів: перевірку окремих модулів, їхнє поєднання, перевірку зв'язку між мікроконтролерами та синхронізацію логіки на всіх рівнях. На початковому етапі Arduino та ESP32 тестувалися окремо за допомогою простих скетчів – рух вперед, зчитування температури, передача JSON через UART. Особливу складність становила стабільність WebSocket-з'єднання між ESP32 та сервером, що вирішувалось використанням регулярних ping-повідомлень, таймаутів і вбудованого watchdog.

Важливою частиною було налагодження UART-протоколу: обробка спеціальних символів, видалення артефактів, розділення команд і повідомлень. Для цього застосовувався монітор порту, логування з серверної сторони, а також фільтрація некоректних повідомлень. Проблеми з передачею відео вирішувались за рахунок обмеження FPS, зміни якості зображення та оптимізації налаштувань камери.

Також виникали типові апаратні труднощі – втрата живлення при одночасному увімкненні двигунів та сервомотора, шум на лініях живлення та підвисання ESP32. Всі ці проблеми вирішувались за рахунок покращення стабілізації напруги, відокремлення живлення для мотора і логіки, додавання конденсаторів та застосування фільтрів.

В результаті було досягнуто стабільної взаємодії всіх компонентів, а налагоджена архітектура дозволяє в майбутньому безболісно додавати нові сенсори або функції до проєкту без порушення цілісності системи.

3.3 Реалізація програмної частини

Програмна частина системи дистанційного керування роботом є ключовим елементом, що забезпечує зв'язок між апаратною платформою, сервером та користувацьким інтерфейсом. Вона формує логіку обробки команд, передачі сенсорних даних, трансляції відео та відповідає за взаємодію всіх компонентів у реальному часі. Реалізація цієї частини базується на принципах розподіленої архітектури, де кожен модуль виконує окреме завдання, але водночас залишається частиною єдиного логічного середовища.

На рівні Arduino було реалізовано програмну підтримку всіх операцій і обчислень які описані в додатку А.

На рівні клієнта був реалізований динамічний інтерфейс за допомогою React, який надає користувачу можливість керувати роботом через кнопки, геймпад або клавіатуру. Інтерфейс оновлюється в реальному часі, відображаючи дані сенсорів, відеопотік з камери, поточний стан підключення та виконувані команди. Реалізація відбувається через компоненти, які отримують дані через WebSocket та оновлюються за допомогою хуків React (useEffect, useState).

Серверна частина, написана на Node.js, виконує функції диспетчера. Вона приймає команди з клієнта, передає їх до ESP32-CAM, а також обробляє відповіді, надсилаючи їх назад у браузер. Крім того, сервер відповідає за координацію розпізнавання об'єктів, таймінг зупинок, обробку втрати цілі, обробку координат і керування режимами. Завдяки використанню socket.io реалізовано надійний WebSocket-зв'язок із клієнтами, що працює у двох напрямках: командний трафік і сенсорний зворотній зв'язок.

На рисунку 3.4 зображено структуру клієнської і серверної частини проекту, які виконуються в одному середовищі розробки.

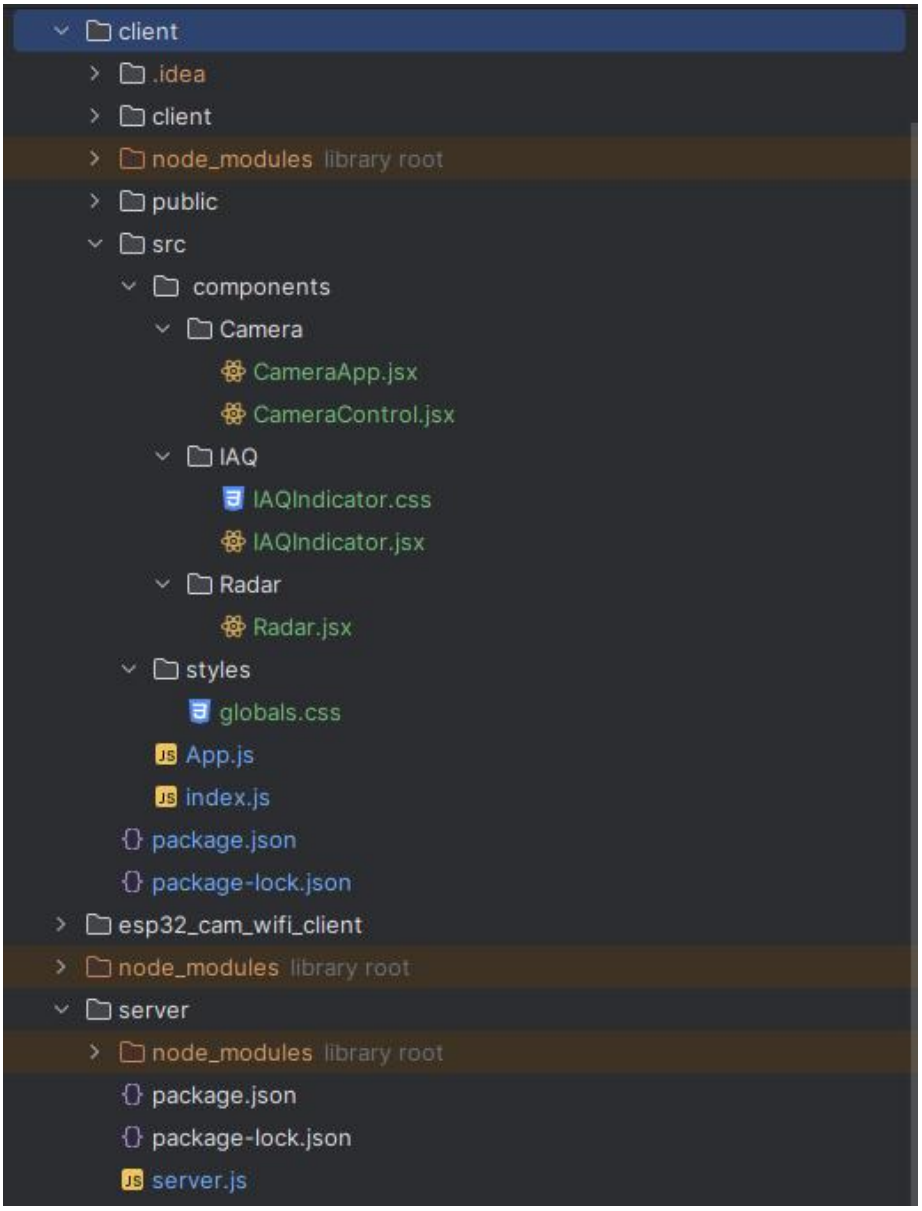

Рисунок 3.4 – Структура клієнської і серверної частини

Програмування мікроконтролерів виконується з урахуванням ресурсних обмежень. Для ESP32-CAM розгорнуто вебсервер, що транслює MJPEG-відео, обробляє HTTP-запити та пересилає команди на Arduino через UART. Arduino Uno працює як виконавчий пристрій, що обробляє текстові команди, керує моторами, сервоприводами і обробляє сигнали з сенсорів, формуючи відповіді у вигляді JSON або простих рядків.

Таким чином, програмна частина охоплює всі рівні взаємодії – від введення користувача до фізичної дії робота, і навпаки – від фізичного стану

системи до візуалізації на екрані.Це забезпечує ефективну роботу системи у режимі реального часу та дозволяє легко масштабувати її функціональність. Реалізація програмної частини для серверу і клієнта описана в додатку Б.

3.3.1 Клієнтська частина: інтерфейс користувача

Клієнтська частина відіграє роль центрального засобу взаємодії користувача з системою. Саме через інтерфейс у веббраузері відбувається передача команд, контроль за станом робота, моніторинг сенсорних даних та перегляд відеопотоку в режимі реального часу. Для реалізації фронтенду обрано бібліотеку React, яка дозволила створити гнучкий, динамічний та масштабований інтерфейс з підтримкою компонентного підходу.

Інтерфейс складається з кількох ключових зон: панелі керування, поля відео, блоків сенсорних показників, секції журналу команд, а також радару, який візуалізує показники відстані. Вся логіка керування зібрана у функціональні компоненти React, які взаємодіють між собою через хуки стану. Оновлення даних відбувається без перезавантаження сторінки завдяки використанню useState і useEffect.

Керування роботом реалізується як через клавіатуру, так і через геймпад, що значно розширює зручність у роботі з платформою. При натисканні відповідних клавіш (наприклад, W – рух вперед, S – назад), або при натисканні на курки чи стіки геймпада, формується команда у заданому форматі (%F#, %B#, %L#, %S# тощо), яка передається через WebSocket на сервер. Кожна дія користувача миттєво реєструється в інтерфейсі, а у разі втрати зв'язку виводиться повідомлення.

Реалізовано також блок відображення сенсорних даних, де виводиться температура, вологість, рівень CO, освітленість, звуковий рівень та дистанція. Дані надходять у форматі JSON з сервера, і обробляються відповідними компонентами. Якщо дані не надходять протягом визначеного часу, в інтерфейсі з'являється повідомлення про втрату зв'язку або значення змінюються на «–».

Окрему роль відіграє відеоінтерфейс, що побудований на вставці MJPEG-потоку з ESP32-CAM. Потік реалізовано через тег <img> із прямим посиланням на HTTP-сервер камери. Це дозволяє уникнути складної обробки відео у клієнта, натомість забезпечує швидку передачу зображень у режимі реального часу. Додатково реалізовано керування параметрами камери (роздільна здатність, якість, яскравість, контраст, FPS) через набір випадаючих меню або слайдерів.

Для покращення взаємодії було реалізовано автоматичне масштабування інтерфейсу, адаптивну верстку (через CSS Grid та Flexbox), а також базову темну тему для зручності використання в умовах слабкого освітлення. Реалізація компонування та логіки відбувалась у середовищі WebStorm, що надало зручність у роботі зі структурою проєкту, модульними імпортами та контролем помилок.

Таким чином, клієнтська частина не лише виконує роль керування, а й повноцінно забезпечує зворотний зв'язок, інформування, візуалізацію і створює комфортний досвід керування для кінцевого користувача.

3.3.2 Серверна частина: обробка команд і даних

Серверна частина є логічним ядром системи, яке забезпечує взаємодію між клієнтським інтерфейсом та апаратною платформою. Саме тут обробляються команди управління, синхронізуються дії, контролюється стабільність з’єднань, обробляються координати цілі в режимі слідування та забезпечується передача сенсорних даних. У межах проєкту сервер реалізовано на основі Node.js**,** що дозволило використати асинхронну модель обробки подій, зменшити затримки та підвищити продуктивність системи в цілому [10].

Ключовим компонентом серверної логіки є бібліотека socket.io, яка забезпечує постійне WebSocket-з’єднання між браузером користувача та сервером. Через цей канал надходять усі команди керування (рух, поворот, зупинка, зміна кута камери тощо), а також передаються назад сенсорні значення з ESP32. Крім того, сервер підтримує одночасне з’єднання з кількома клієнтами,

що дозволяє організувати багатокористувацьке середовище або розширити систему на майбутнє.

Команди, які надходять з React-клієнта, потрапляють у Node.js, де вони проходять попередню перевірку формату, і далі перенаправляються або до ESP32, або безпосередньо через UART до Arduino. Для цього застосовується логіка маршрутизації команд залежно від їх префіксу або типу. Усі команди реалізовані у вигляді простих текстових конструкцій (наприклад, %F100#), що дозволяє мінімізувати обсяг переданих даних і спростити обробку на рівні мікроконтролера.

Крім команд, сервер приймає сенсорні дані з ESP32**,** які надходять у форматі JSON або окремих рядків. Node.js обробляє ці повідомлення, розподіляє дані по відповідних клієнтах і оновлює їхній інтерфейс у режимі реального часу. Реалізовано також обробку таймаутів – якщо дані не надходять протягом визначеного часу, сервер передає клієнту сигнал про втрату зв’язку або автоматично зупиняє робота, щоб уникнути аварійних ситуацій.

Окрему роль у серверній логіці відіграє модуль обробки координат у режимі автослідування. Через HTTP POST-запит клієнт надсилає серверу координати положення об'єкта на зображенні. Node.js аналізує відхилення об'єкта від центру кадру, приймає рішення про зміну кута камери або активацію руху робота у відповідному напрямку. При зникненні об'єкта з кадру протягом певного часу активується зупинка, а інтерфейс інформує користувача про втрату цілі.

Усі модулі серверної частини реалізовані в одному програмному середовищі – WebStorm, що дозволило підтримувати чітку структуру проєкту, використовувати інтегрований термінал для запуску, налагодження й логування, а також зручно працювати з залежностями через npm.

Завдяки асинхронній архітектурі, модульності та ефективному WebSocket-зв’язку серверна частина системи є надійним комунікаційним посередником,

який гарантує стабільну роботу системи, швидке реагування на дії користувача та контроль за всіма процесами на фізичному рівні.

3.4 Тестування

Після завершення реалізації апаратної та програмної частин системи було проведено повноцінне тестування з метою перевірки працездатності всіх модулів, стабільності взаємодії між ними, а також коректної реакції на команди користувача в режимі реального часу. Особлива увага приділялася стабільності з'єднань, обробці граничних ситуацій, втратам зв'язку та взаємодії з сенсорами.

Тестування проводилось у кількох етапах: спочатку окремо перевірялась робота кожного модуля – Arduino, ESP32-CAM, Node.js сервер, клієнт React. Після цього система запускалась у повному складі в локальній мережі для інтеграційної перевірки. При керуванні з клавіатури та геймпада затримки у виконанні команд в середньому не перевищували 50 мс, що є цілком прийнятним для ручного дистанційного управління. Реакція на повороти, зміни напрямку та зміни швидкості була стабільною. Геймпад працював коректно як при аналоговому керуванні (через курки L2/R2), так і при дискретному (через кнопки).

Сенсори стабільно передавали значення з Arduino на ESP32, далі через WebSocket – у React-клієнт. Збої виникали лише у випадках розрядженого акумулятора або перевантаження живлення, що підтверджує потребу в окремому джерелі для двигунів. Значення температури, вологості та відстані відображались у реальному часі з оновленням кожні 500-1000 мс.

Під час тестування відеопотоку з ESP32-CAM було помічено, що при встановленні надто високої роздільної здатності (UXGA) затримка потоку значно зростає, а стабільність зменшується. Оптимальними параметрами виявилися CIF (400x296) або VGA (640x480) при частоті оновлення до 10 FPS.

Зміна яскравості, контрасту, якості кадру працювала без збоїв, що підтверджує коректну роботу API налаштування камери.

У режимі автослідування сервер успішно приймав координати об’єкта, визначав напрямки зміщення та надсилав відповідні команди на ESP32 і Arduino. Якщо об’єкт виходив за межі кадру або не змінював положення, активувався таймер зупинки. Реакція на втрату цілі була стабільною, із зупинкою руху і поверненням камери у нейтральне положення.

Загалом тестування показало, що реалізована система є функціонально повною, стабільною в умовах локальної мережі та здатна реагувати на події в реальному часі. Виявлені проблеми стосувались здебільшого енергоживлення або мережевих нюансів, які були усунені в процесі налагодження.

Крім перевірки працездатності, у процесі тестування були також визначені потенційні напрями для вдосконалення системи. Зокрема, спостереження за поведінкою компонентів у різних режимах дозволило сформувати технічні рішення щодо підвищення стабільності та зручності використання. Було виявлено, що доцільним є впровадження функцій збереження сенсорних даних, адаптації інтерфейсу для мобільних пристроїв, а також розширення системи автономного реагування.

ВИСНОВКИ

За результатами виконання кваліфікаційної роботи можна зробити відповідні висновки:

Реалізовано вебінтерфейс для керування роботом, створений на основі React. Інтерфейс забезпечує зручне дистанційне керування в режимі реального часу, підтримує як клавіатуру, так і геймпад, містить візуальні блоки керування, сенсорні показники, MJPEG-потік камери та елементи налаштувань. Застосовано компонентний підхід, гнучку стилізацію та використано WebSocket-клієнт для зв’язку з сервером.

Розроблено серверну частину системи з використанням Node.js, яка відповідає за обробку WebSocket-запитів, координацію обміну даними між клієнтом і апаратною частиною, а також підтримку HTTP-запитів для розпізнавання об'єктів. Завдяки асинхронній архітектурі реалізовано швидке реагування, обробку команд у текстовому форматі та надійне з'єднання з клієнтом і ESP32.

Досліджено можливості інтеграції ESP32-CAM у системи керування, що включає реалізацію відеопотоку через HTTP, налаштування параметрів зображення, передавання даних з UART та ретрансляцію команд до Arduino. Тестування підтвердило ефективність роботи камери при роздільній здатності до VGA із затримкою в межах 150-300 мс.

Візуалізовано сенсорні показники в інтерфейсі користувача, зокрема температуру, вологість, відстань до перешкод, рівень CO, освітленість та звук. Дані передаються через послідовне з’єднання з Arduino, обробляються на ESP32 та передаються клієнту через сервер. Реалізовано обробку втрати зв’язку та зупинку руху при критичних значеннях.

Спроектовано та протестовано повну архітектуру системи, що охоплює взаємодію між Arduino, ESP32-CAM, Node.js сервером та React-клієнтом. Архітектура побудована за модульним принципом, що дозволяє легко

масштабувати систему або замінювати окремі її компоненти. Система пройшла повне тестування в умовах локальної мережі й продемонструвала стабільну роботу при передачі даних і команд у реальному часі.

Запропоновано кілька напрямків для вдосконалення створеної системи з метою підвищення її функціональності, надійності та зручності у використанні. Зокрема, доцільним є впровадження збереження сенсорних даних у базу для подальшого аналізу, реалізація мобільного застосунку з адаптованим інтерфейсом, а також додавання автономних режимів на основі розпізнавання об’єктів або маршруту. Також розглядається можливість інтеграції хмарного зберігання відеопотоку, покращення стабільності з’єднання в умовах нестабільного Wi-Fi, а також розширення можливостей налаштування безпосередньо через інтерфейс користувача.

Отримано практичні результати, які свідчать про ефективність використання сучасних вебтехнологій у поєднанні з доступними мікроконтролерними платформами для побудови автономних роботизованих систем. Реалізація підтвердила доцільність поєднання React і Node.js і ESP32 і Arduino як універсальної основи для створення інтерактивних інженерних рішень.

Розроблену систему можна використовувати для навчальних, демонстраційних та дослідницьких цілей, а також як базу для розширення в напрямках автоматизованого моніторингу, охоронних роботів або мобільних платформ зі штучним інтелектом на борту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is the Internet of Things (IoT)?. CloudBlue. URL: https://surl.li/vhlohw (дата звернення: 11.03.2025).

2. An overview of HTTP and Components of HTTP-based systems. Mdn web docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview (дата звернення: 14.03.2025).

3. ESP32 WebSocket Server: Display Sensor Readings. Random Nerd Tutorials. URL: https://surl.lu/jvlhgx (дата звернення: 15.03.2025).

4. Gas sensor BME680. Bosch. URL: https://surl.li/aqjuzd (дата звернення: 17.04.2025).

5. WebSockets Explained – What is WebSocket?. Moralis. URL: https://surl.li/wdtmeb (дата звернення: 17.03.2025).

6. Вступ в React, якого нам не вистачало. DevZone. URL: https://surli.cc/pqcuje (дата звернення: 21.03.2025).

7. How to Start a Node.js Server Quick and Easy Guide. RunJS. URL: https://runjs.app/blog/how-to-start-a-node-server (дата звернення: 24.03.2025).

8. Getting Started with Arduino IDE. Arduino. URL: https://surl.li/kseyqh (дата звернення: 25.03.2025).

9. The JavaScript and TypeScript IDE. .JetBrains URL: https://www.jetbrains.com/webstorm/ (дата звернення: 21.04.2025).

10. Що таке Node.js? Основи серверної розробки на JavaScript. DevZone. URL: http://devzone.org.ua/post/shcho-take-nodejs-osnovy-servernoyi-rozrobky-na-javascript (дата звернення: 14.04.2025).

ДОДАТКИ

Додаток А

Програмна частина прошивки для Arduino UNO

```
#include "IR_remote.h"
#include "keymap.h"
#include <Servo.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#define TRIG_PIN 12
#define ECHO_PIN 13

IRremote ir(3);
Servo servo_10;
Adafruit_BME680 bme;
unsigned long warmupStart = 0;
const unsigned long warmupDuration = 5UL * 60 * 1000; // 5 хвилин
unsigned long lastSensorSend = 0;
const unsigned long sensorInterval = 500;
String uartBuffer = "";
int motorSpeed = 40;
volatile float V_Servo_angle = 90;
volatile int Front_Distance = 0;
volatile char IR_Car_Mode = ' ';
volatile boolean IR_Mode_Flag = false;
void updateSensorsAndSend() {
  if (millis() - lastSensorSend < sensorInterval) return;
  lastSensorSend = millis();
  int lightValue = analogRead(A0);
  int soundValue = analogRead(A1);
  int mq7Value = analogRead(A2);
  float temp = 0, press = 0, hum = 0, gasRes = 0;
  if (bme.performReading()) {
    temp = bme.temperature;
    press = bme.pressure / 100.0;
    hum = bme.humidity;
    gasRes = bme.gas_resistance / 1000.0;  // в кОм
  }
  Front_Distance = checkdistance();
  // Вивід у форматі JSON з додатковим полем "gas"
  Serial.print("{\"temperature\":"); Serial.print(temp, 2);
  Serial.print(",\"humidity\":"); Serial.print(hum, 2);
  Serial.print(",\"pressure\":"); Serial.print(press, 2);
  Serial.print(",\"light\":"); Serial.print(lightValue);
  Serial.print(",\"sound\":"); Serial.print(soundValue);
  Serial.print(",\"co\":"); Serial.print(mq7Value);
  Serial.print(",\"distance\":"); Serial.print(Front_Distance);
  Serial.print(",\"gas\":"); Serial.print(gasRes, 2);
  Serial.println("}");
}
void Move_Forward(int s) { digitalWrite(2, HIGH); analogWrite(5, s);
digitalWrite(4, LOW); analogWrite(6, s); }
void Move_Backward(int s) { digitalWrite(2, LOW); analogWrite(5, s);
digitalWrite(4, HIGH); analogWrite(6, s); }
void Rotate_Left(int s) { digitalWrite(2, LOW); analogWrite(5, s); digitalWrite(4,
LOW); analogWrite(6, s); }
void Rotate_Right(int s) { digitalWrite(2, HIGH); analogWrite(5, s);
digitalWrite(4, HIGH); analogWrite(6, s); }
void STOP() { digitalWrite(2, LOW); analogWrite(5, 0); digitalWrite(4,
HIGH); analogWrite(6, 0); }
float checkdistance() {
```

```
  digitalWrite(12, LOW); delayMicroseconds(2);
  digitalWrite(12, HIGH); delayMicroseconds(10);
  digitalWrite(12, LOW);
  return pulseIn(13, HIGH) / 58.00;
}
void Ultrasonic_Avoidance() {
  Front_Distance = checkdistance();
  if (Front_Distance <= 10) {
    Move_Backward(100); delay(200);
    (random(1, 100) <= 50) ? Rotate_Left(100) : Rotate_Right(100); delay(500);
  } else if (Front_Distance <= 20) {
    STOP(); delay(200);
    (random(1, 100) <= 50) ? Rotate_Left(100) : Rotate_Right(100); delay(500);
  } else {
    Move_Forward(70);
  }
}
void Ultrasonic_Follow() {
  Front_Distance = checkdistance();
  if (Front_Distance <= 5) Move_Backward(80);
  else if (Front_Distance <= 10) STOP();
  else Move_Forward(100);
}
void IR_remote_control() {
  switch (IR_Car_Mode) {
    case 'b': Move_Backward(motorSpeed); delay(300); STOP(); break;
    case 'f': Move_Forward(motorSpeed);  delay(300); STOP(); break;
    case 'l': Rotate_Left(motorSpeed);   delay(300); STOP(); break;
    case 'r': Rotate_Right(motorSpeed);  delay(300); STOP(); break;
    case 's': STOP(); break;
    case     '+':     V_Servo_angle     =     min(180,     V_Servo_angle     +     3);
servo_10.write(round(V_Servo_angle)); break;
    case     '-':     V_Servo_angle     =     max(0,     V_Servo_angle     -     3);
servo_10.write(round(V_Servo_angle)); break;
  }
  IR_Car_Mode = ' ';
  byte code = ir.getIrKey(ir.getCode(), 1);
  if      (code == IR_KEYCODE_UP)    IR_Car_Mode = 'f';
  else if (code == IR_KEYCODE_LEFT)  IR_Car_Mode = 'l';
  else if (code == IR_KEYCODE_DOWN)  IR_Car_Mode = 'b';
  else if (code == IR_KEYCODE_RIGHT) IR_Car_Mode = 'r';
  else if (code == IR_KEYCODE_OK)    IR_Car_Mode = 's';
  else if (code == IR_KEYCODE_2)     IR_Car_Mode = '+';
  else if (code == IR_KEYCODE_8)     IR_Car_Mode = '-';
  else if (code == IR_KEYCODE_1)       { motorSpeed = min(120, motorSpeed + 10);
Serial.print("Швидкість: "); Serial.println(motorSpeed); }
  else if (code == IR_KEYCODE_3)       { motorSpeed = max(0, motorSpeed - 10);
Serial.print("Швидкість: "); Serial.println(motorSpeed); }
}
void handleSerialCommand(String fullCmd) {
  fullCmd.trim();
  int idx = 0;
  while ((idx = fullCmd.indexOf('%')) != -1) {
    fullCmd = fullCmd.substring(idx);
    int end = fullCmd.indexOf('#');
    if (end == -1) {
      Serial.println("⛔ Команда не завершена символом #");
      return;
    }
    String cmd = fullCmd.substring(0, end + 1);
    fullCmd = fullCmd.substring(end + 1);
    Serial.print("[Arduino] Отримано команду: "); Serial.println(cmd);
    if (!cmd.startsWith("%") || !cmd.endsWith("#")) {
```

```
      Serial.println("⛔ Команда не відповідає формату %...#");
      continue;
    }
    char command = cmd.charAt(1);
    int value = -1;
    if (cmd.length() > 3 && isDigit(cmd.charAt(2))) {
      String valStr = cmd.substring(2, cmd.length() - 1);
      value = valStr.toInt();
    }
    if (command == 'V') {
      motorSpeed = constrain(value, 0, 120);
      Serial.print("☑ Нове motorSpeed з геймпада (R3): ");
      Serial.println(motorSpeed);
      continue;
    }
    Serial.print("🔧 Команда: "); Serial.print(command);
    Serial.print(" | Швидкість: ");
    if (value != -1) Serial.println(value);
    else Serial.println(motorSpeed);
    switch (command) {
      case 'F': Move_Forward(value != -1 ? value : motorSpeed); break;
      case 'B': Move_Backward(value != -1 ? value : motorSpeed); break;
      case 'L': Rotate_Left(value != -1 ? value : motorSpeed); break;
      case 'R': Rotate_Right(value != -1 ? value : motorSpeed); break;
      case 'S': STOP(); break;
      case 'H': V_Servo_angle = min(180, V_Servo_angle + 4);
servo_10.write(round(V_Servo_angle)); break;
      case 'G': V_Servo_angle = max(0, V_Servo_angle - 4);
servo_10.write(round(V_Servo_angle)); break;
      case 'A': Ultrasonic_Avoidance(); break;
      case 'Z': Ultrasonic_Follow(); break;
      case 'C':
        if (value >= 0 && value <= 180) {
          V_Servo_angle = value;
          servo_10.write(value);
          Serial.print("🎯 Установка кута серво (камера): ");
          Serial.println(value);
        }
        break;
    }
  }
}
void setup() {
  Serial.begin(9600);
  Wire.begin();
  if (!bme.begin()) {
    Serial.println("Не знайдено BME680!"); while (1);
  }
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setGasHeater(320, 150);
  servo_10.attach(10);
  servo_10.write(V_Servo_angle);
  pinMode(2, OUTPUT); pinMode(4, OUTPUT);
  pinMode(5, OUTPUT); pinMode(6, OUTPUT);
  pinMode(12, OUTPUT); pinMode(13, INPUT);
}
void loop() {
  updateSensorsAndSend();
  IR_remote_control();
  while (Serial.available() > 0) {
    char c = Serial.read();
```

```
    if (c == '\n') {
      uartBuffer.trim();
      if (uartBuffer.length() > 0) {
        if (uartBuffer == "PONG") {
          Serial.println("ESP32 відповів: PONG");
        } else {
          handleSerialCommand(uartBuffer);
        }
      }
      uartBuffer = "";
    } else {
      uartBuffer += c;
    }
  }
}
```

Додаток Б

Код серверної частини Node.js і клієнтської частини React app

```
const express = require('express');
const WebSocket = require('ws');
const http = require('http');
const bodyParser = require('body-parser');
const cors = require('cors');
let lastTurnTime = 0;
const TURN_COOLDOWN = 300; // ⏱️ 300 мс між поворотами
const app = express();
app.use(cors());
app.use(bodyParser.json());
app.use('/upload-frame', express.raw({ type: 'application/octet-stream', limit:
'2mb' }));
let lastDeltaX = 0; // 📦 останнє горизонтальне відхилення
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });
let esp32Connection = null;
let browserClients = [];
let currentCameraAngle = 90;
let lastObjectTimestamp = 0;
let autoFollowEnabled = true;
let stopTimeout = null;
let latestFrame = null;
const MIN_DISTANCE = 20; // см — гранична дистанція під'їзду
const translateCommand = (cmd) => {
  switch (cmd) {
    case "forward": return "%F#";
    case "backward": return "%B#";
    case "left": return "%L#";
    case "right": return "%R#";
    case "stop": return "%S#";
    default: return cmd;
  }
};
// ======= 🌐 WebSocket логіка =======
wss.on('connection', (ws) => {
  console.log("New WebSocket connection");
  ws.on('message', (message) => {
    const msgStr = message.toString().trim();
    console.log("[WS received]:", msgStr);
    if (msgStr === "ESP32") {
      esp32Connection = ws;
      console.log("☑️ ESP32 підключено");
```

```
      return;
    }
    try {
      const parsed = JSON.parse(msgStr);
      if (parsed.type === "sensor") {
        browserClients.forEach(client => {
          if (client.readyState === WebSocket.OPEN) {
            client.send(JSON.stringify({  type:  "sensor",  payload:  parsed.payload
}));
          }
        });
      } else if (parsed.temperature !== undefined) {
        browserClients.forEach(client => {
          if (client.readyState === WebSocket.OPEN) {
            client.send(JSON.stringify(parsed));
          }
        });
      }
    } catch (e) {
      if (esp32Connection && esp32Connection.readyState === WebSocket.OPEN) {
        const translated = translateCommand(msgStr);
        console.log("☑️ Передаю команду на ESP32:", translated);
        esp32Connection.send(translated);
      }
    }
  });
  ws.on('close', () => {
    console.log("❌ З'єднання закрито");
    if (ws === esp32Connection) {
      console.log("⚠️ ESP32 втрачено");
      esp32Connection = null;
    }
    browserClients = browserClients.filter(c => c !== ws);
  });
  browserClients.push(ws);
});
// ======= 🌎 Root endpoint =======
app.get('/', (req, res) => {
  res.send('Robot server is running.');
});

// ======= 🚀 Старт сервера =======
server.listen(3000, () => {
  console.log('🚀 Сервер запущено на порту 3000');
}); return (
        <div className="app">
            <h1 className="title">🚗 Управління Роботом</h1>
            <div className="main-panel">
                <div className="left-panel">
                    <div className="control-panel">
                        <h2>🕹️ Контроль руху</h2>
                        <p>Швидкість: <strong>{motorSpeed}</strong></p>
                        <div className="movement-buttons">
                            <button onClick={() => sendCommand('%F#')}>⬆️</button>
                            <button onClick={() => sendCommand('%S#')}>⏹️</button>
                            <button onClick={() => sendCommand('%B#')}>⬇️</button>
                            <button onClick={() => sendCommand('%L#')}>⬅️</button>
                            <button onClick={() => sendCommand('%R#')}>➡️</button>
                        </div>
                        <CameraApp />
                        <div className="custom-command">
```

```
                            <h3>💬 Ручна команда:</h3>
                            <input
                                type="text"
                                value={inputCommand}
                                onChange={(e) => setInputCommand(e.target.value)}
                                placeholder="Введіть команду"
                            />
                            <button onClick={handleSendCustom}>Відправити</button>
                        </div>
                        <p className={connected ? 'connected' : 'disconnected'}>
                            {connected ? `🟢 Підключено: ${lastUpdate}` : "🔴 Немає
з'єднання"}
                        </p>
                    </div>
                </div>
                <div className="right-panel sensor-panel">
                    <div style={{ display: 'flex', flexDirection: 'column', gap:
'20px' }}>
                        <div>
                            <h2>📊 Показники сенсорів</h2>
                            <ul>
                                <li>🌡️ Температура: {sensorData.temperature !== "-
" ? sensorData.temperature.toFixed(1) + " °C" : "-"}</li>
                                <li>💧 Вологість: {sensorData.humidity !== "-" ?
sensorData.humidity.toFixed(1) + " %" : "-"}</li>
                                <li>🏔️ Тиск: {sensorData.pressure !== "-" ?
sensorData.pressure.toFixed(1) + " hPa" : "-"}</li>
                                <li>💡 Освітленість: {sensorData.light}</li>
                                <li>🔊 Звук: {sensorData.sound}</li>
                                <li>🧵 CO (MQ-7): {sensorData.co}</li>
                                <li>
                                    🔥 Статус сенсора:
                                    <span style={{ color: isWarmedUp ? 'limegreen'
: 'crimson', fontWeight: 'bold' }}>
                                        {isWarmedUp ? "Прогріто" : "Не прогріто"}
                                    </span>
                                    {!isWarmedUp && <span> —
{formatTime(warmupTime)}</span>}
                                </li>
                                <IAQIndicator iaq={sensorData.iaq} />
                            </ul>

                            <h3>🔁 Поточна команда:</h3>
                            <p><strong>{currentCommand.toUpperCase()}</strong></p>
                        </div>

                        <div>
                            <h3>📡 Радар</h3>
                            <Radar distance={distance} />
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
}
```