**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»

## ВЕБ-ІНТЕРФЕЙС СИСТЕМИ ІНТЕРНЕТУ РЕЧЕЙ НА БАЗІ ARDUINO

## WEB INTERFACE OF THE INTERNET OF THINGS SYSTEM BASED ON ARDUINO

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

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-42
Шульгач Іван Сергійович

(підпис)

Керівник:
ст.викладач
Міскевич Оксана Іванівна

(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

*Шульгачу Івану Сергійовичу*
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Веб-інтерфейс системи інтернету речей на базі Arduino*

Керівник роботи *ст. викладач Міскевич Оксана Іванівна*

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, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
*Вступ*
*Аналіз предметної області та наявних рішень*
*Вибір апаратної та програмної бази для проєкту*
*Проєктування та тестування системи*
*Висновки*

5. Перелік графічного (ілюстративного) матеріалу:
*Схема підключення датчика KY-016 до плати NodeMCU*
*Схема підключення датчика DHT11 до плати NodeMCU*
*Схема підключення LCD1602A і ESp8266 до Arduino Uno*
*Інтерфейс веб-додатку для керування кольором та перегляду даних*

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Аналіз проблеми та постановка завдань* | *Міскевич О. І., ст. викладач* | | |
| *Обгрунтування вибору апаратної та програмної бази* | *Міскевич О. І., ст. викладач* | | |
| *Практична реалізація* | *Міскевич О. І., ст. викладач* | | |
| *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 10.01.2024 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. | *Обгрунтування вибору апаратної та програмної бази* | до 02.03.2025 р. | Виконано |
| 3. | *Практична реалізація* | 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. | *Представлення остаточного варіанту кваліфікаційної роботи керівникові* | до 15.05.2025 р. | Виконано |
| 9. | *Нормоконтроль* | до 30.05.2025 р. | Виконано |
| 10. | *Інструментальна перевірка на академічний плагіат* | до 03.06.2025 р. | Виконано |
| 11. | *Представлення кваліфікаційної роботи бакалавра до захисту* | до 10.06.2025 р | Виконано |

**Здобувач вищої освіти** _____ (підпис) Шульгач І.С. (прізвище, ініціали)

**Керівник кваліфікаційної роботи** _____ (підпис) Міскевич О. І. (прізвище, ініціали)

# АНОТАЦІЯ

Шульгач І.С. Веб-інтерфейс системи інтернету речей на базі Arduino. 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 огляд предметної області, розглянуто актуальність теми веб-інтерфейсів для IoT, визначено основні поняття та архітектурні рівні Інтернету речей, описано переваги та сфери застосування систем керування та моніторингу на базі мікроконтролерів Arduino Uno і Arduino NodeMCU.

У другому розділі було обрано та обгрунтовано вибір апаратної і програмної бази проєкту. Описано характеристики та порівняння можливостей Arduino Uno та Arduino NodeMCU. Розглянуто основний принцип роботи та особливості застосування датчика, для вимірювання температури і вологості DHT11, RGB-лампочки KY-016 та дисплея LCD1602A.

Розглянуто протоколи обміну даними HTTP і MQTT та їхні переваги. Вибір стеку фронтенду React та Next.js із менеджером пакетів Yarn, зручність інтеграції з MQTT.

У третьому розділі описано алгоритм реалізації пристрою. Описано загальну роботу мікроконтролерів, ініціалізацію з'єднання з Wi-Fi, підключення до HiveMQ через MQTT, налаштування топіків для обміну даними.

Наведено блок-схему схему підключення компонентів. Розглянуто реалізацію коду, читання даних DHT11 та інтеграцію LCD1602A, можливості вдосконалення.

Ключові слова: Internet of Things, Arduino Uno, ESP8266, NodeMCU, MQTT, HiveMQ, React, Next.js, DHT11, KY-016, LCD1602A

# ANNOTATION

Shulhach I. Web Interface for an Internet of Things System based on Arduino. Manuscript.

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

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

In the first chapter, an overview of the subject area is provided, the relevance of web interfaces for IoT is discussed, the main concepts and architectural levels of the Internet of Things are defined, and the advantages and application domains of control and monitoring systems based on Arduino Uno and Arduino NodeMCU microcontrollers are described.

In the second chapter, the hardware and software base of the project is chosen and justified. The characteristics and comparative capabilities of the Arduino Uno and Arduino NodeMCU microcontrollers are described. The basic operating principle and application features of the DHT11 temperature and humidity sensor, the WS2812 RGB lamp, and the LCD1602A display are examined. The data exchange protocols HTTP and MQTT and their advantages are discussed. The choice of the React and Next.js frontend stack with the Yarn package manager and the convenience of integrating with MQTT are justified.

In the third chapter, the device implementation algorithm is described. The overall operation of the microcontrollers is detailed, including initializing the Wi-Fi connection, connecting to HiveMQ via MQTT, and configuring topics for data exchange. A flowchart of the microcontroller firmware is provided. The code implementation for reading DHT11 data and integrating the LCD1602A is examined, along with potential improvements.

Keywords: Internet of Things, Arduino Uno, ESP8266, NodeMCU, MQTT, HiveMQ, React, Next.js, DHT11, KY-016, LCD1602A.

# ЗМІСТ

# ВСТУП

Актуальність теми. Сьогодні, розвиток технологій Інтернету речей, сприяє поширенню розумних систем управління та моніторингу у різноманітних сферах, від розумного будинку, до промислового контролю. Стрімке зростання кількості підключених пристроїв, створює необхідність розробки простих та зручних веб-інтерфейсів для віддаленого керування та обробки даних. Використання Arduino у поєднанні з протоколом MQTT, дозволяє забезпечити ефективну передачу даних, низьку енергоспоживаність, ефективну передачу даних і просту інтеграцію у сучасні системи. Виходячи з цього, розробка веб-інтерфейсу для IoT-систем на базі Arduino є актуальною, як із наукової, так і з практичної сторони.

Метою роботи є розробка та реалізація веб-інтурфейсу системи Інтернету речей на базі Arduino Uno і Arduino NodeMCU з використанням модулей ESP8266, DH11, LCD1602A, KY-016, протоколу MQTT через брокер HiveMQ.

Об’єкт дослідження – система Інтернету речей з веб-інтерфейсом для керування пристроями на базі мікроконтролерів та збором даних за допомогою протокола публікації та підписки.

Предмет дослідження – методи та інструменти побудови веб-інтерфейсу на основі React та Next.js з використанням MQTT-брокера HiveMQ для інтерактивного керування KY-016, збору даних із сенсора DHT11 та відображення результатів на LCD1602A.

Завдання, які необхідно виконати:

Реалізувати прошивку для ESP8266 і NodeMCU з підтримкою MQTT, для публікації даних сенсора DHT11 і прийому команд для керування KY-016

Розробити веб-інтерфейс на базі React і Next.js з підключенням до брокера HiveMQ, для керування кольором лампочки та відображенням температури і вологості у реальному часі.

Дослідити характеристики протоколів HTTP і MQTT, порівняти їхню ефективність, особливості використання.

Візуалізувати отримані дані температури та вологості у веб-інтерфейсі і LCD1602A.

Запропонувати шляхи вдосконалення проєкту, а саме впровадження TLS-з’єднання для MQTT, маштабування на кілька сенсорів, покращення мобільної адаптації інтерфейсу.

# РОЗДІЛ 1
# АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ

## 1.1 Поняття Інтернету речей та його архітектурні рівні

«Інтернет речей (Internet of Things, скорочено IoT) - це глобальна мережа підключених до Інтернету фізичних пристроїв – «речей», оснащених сенсорами, датчиками і пристроями передачі інформації» [1], ці пристрої об’єднані, завдяки підключенню до центру контролю, управління і обробки інформації. Термін «Інтернет речей» був введений Кевіном Ештоном у 1999 році, щоб описати систему у якій пристрої могли бути пов’язані мережею інтернет. Сьогодні це стало популярним терміном, для опису сценаріїв, у яких інтернет з’єднання і обчислювальна здатність поширюється на безліч об’єктів, пристроїв.

Для упорядкування функцій та забезпечення гнучкості системи у розробці IoT-рішень застосовують багаторівневу структуру, де кожен рівень відповідає за свої функції.

1.1.1 Рівень сприйняття

Цей рівень складається із сенсорів і актуаторів, «актуатори дозволяють об’єктам взаємодіяти з фізичним світом, наприклад, вмикати або вимикати пристрої» [2], що збирають та передають дані у цифровому або аналоговому форматі і відповідає за інтерфейс із фізичним середовищем.

1.1.2 Мережевий рівень

Мережевий рівень включає функції управління ресурсами мережі, забезпечує передачу даних від рівня сприйняття, до платформи обробки. На цьому рівні використовуються різноманітні канали зв’язку: протоколи низького енергоспоживання, локальні бездротові мережі і мобільний зв’язок, для віддалених об’єктів.

1.1.3 Рівень обробки

Рівень обробки відповідає за централізоване опрацювання, зберігання, аналіз та маршрутизацію отриманих даних. З метою зниження навантаження на кінцеві пристрої і підготовки інформації до подальшого аналізу

використовуються брокери повідомлень, бази даних, для зберігання даних і сервіси, що виконують первинну агрегацію або фільтрацію.

1.1.4 Прикладний рівень

На цьому рівні користувач отримує інструменти для візуалізації, моніторингу, та керування пристроями через графічні інтерфейси на базі веб-технологій, або мобільних додатків.

1.1.5 Бізнес-рівень

Бізнес-рівень відповідає за аналіз ключових показників, формування звітів, перетворення зібраних даних на стратегічні рішення. Аналітичні сервіси обробляють велику кількість інформації, застосовую інструменти BI «BI (Business Intelligence, інтелектуальний аналіз даних, бізнес-аналітика) комп'ютерні методи і інструменти для організацій, що забезпечують переклад транзакційної ділової інформації в форму, придатну для бізнес-аналізу, а також засоби для роботи з обробленою таким чином інформацією» [3], а також інтеграцію з ERP «Планування ресурсів підприємства (ERP) – це тип програмного забезпечення, за допомогою якого організації керують повсякденною господарською, комерційною, рекреаційною діяльністю, як-от бухгалтерією, закупівлями, проєктами, ризиками й дотриманням норм і правил, а також операціями в логістичному ланцюзі» [4], та CRM «CRM (Customer Relationship Management, Управління взаємовідносинами з клієнтами), це програмне забезпечення, яке допомагає організаціям перетворювати потенційних клієнтів на реальних, залучати та зрощувати ліди, а також утримувати наявних клієнтів, охоплюючи всі етапи взаємодії з ними» [5] системами.

1.1.6 Взаємодія між рівнями

Взаємодія між рівнями IoT здійснюється через строго визначені інтерфейси і протоколи. Зібрані дані транспортуються мережевим рівнем до обробки, де брокери та бази даних готують їх до прикладного рівня.

Головна перевага багаторівневої архітектури, це модульність. Кожен рівень можна розвивати, маштабувати чи змінювати незалежно від інших. У

сукупності все це дозволяє просто та швидко адаптувати систему систему до нових вимог.

## 1.2 Сфери застосування IoT-систем на базі Arduino Uno і NodeMCU

Сучасні IoT-системи, що побудовані на основі доступних і гнучких платформ Arduino Uno та NodeMCU, знайшли широке застосування у побуті, промисловості та різних галузях науки. У поєднанні з простотою апаратного забезпечення, великої кількості бібліотек і того, що ця система є відкритою, робить ці рішення невід'ємною частиною проєктів розумних приладів і систем моніторингу.

Однією із найпоширеніших сфер, де використовується IoT-системи на базі Arduino, це системи автоматизації будинку. Arduino Uno I NodeMCU реалізувати розумні світильники, що працюють відповідно до часу доби, або при присутності людей. За допомогою датчиків відкриття дверей, інфрачервоних бар'єрів та камер, формується мережа моніторингу, що сповіщає користувача про спроби несанкціонованого доступу. Інтеграція датчиків температури та вологості, дає змогу віддалено стежити за показниками у приміщеннях, також автоматично керувати кондиціонерами та обігрівачами через реле, під'єднане до Arduino.

У промисловості, потреби у зборі даних і віддаленому керуванні є критичними, для безперебійної роботи. NodeMCU у поєднанні з датчиками тиску, рівня рідини та вібрації дає змогу реалізувати віддалений моніторинг стану насосів, двигунів. Отримані дані передаються на сервер, де аналізуються у реальному часі. Використовуючи сенсори струму та вібрації, можна формувати графіки роботи двигунів і прогнозувати потребу у обслуговувані.

Боротьба з погіршенням стану довкілля вимагає розгортання дешевих і маштабованих систем спостереження. Використовуючи датчики рівня води і електропровідності, Arduino може стежити за станом водойм, чи запасів питної води. При перевищенні встановлених норм, система надсилає сповіщення екологам. У польових умовах, використовуючи датчики вологості грунту,

Arduino дозволяє автоматизувати полив, підтримуючи оптимальний рівень вологості для культур.

Також, платформи Arduino є дуже популярними у навчанні та наукових дослідженнях, через низький поріг входу. Студенти мають змогу створювати власні IoT-системи, відпрацьовуючи свої навички роботи з датчиками, мікроконтролерами. Вчені використовують NodeMCU у зборі даних у польових умовах.

Загалом, Arduino Uno і NodeMCU забезпечують доступність, швидкість і широку функціональність, що робить ці платформи універсальним вибором для побудови своєї IoT-системи. Така універсальність застосувань підкреслює важливість розробки сучасного і зручного веб-інтерфейсу.

### 1.3 Огляд існуючих засобів реалізації IoT-систем

Платформ і рішень для Інтернету речей на базі Arduino є безліч, завдяки своїй доступності і гнучкості платформа стала однією з найопулярніших, тому навколо неї створено безліч готових бібліотек, фреймворків і хмарних сервісів.

Одним із перших і найвідоміших сервісів для швидкого запуску проєктів на базі Arduino є Blynk. Мобільний застосунок надає широкий вибір віджетів, які налаштовуються візуально. Arduino Uno і NodeMCU підключаються до інтернету за допомогою бездротового зв’язку, а обмін даними здійснюється через хмарний MQTT брокер. Завдяки готовим бібліотекам поріг входу низький, що дозволяє новачкам швидко освоїтись.

Arduino IoT Cloud – це офіційний хмарний сервіс від розробників Arduino. Він забезпечує автоматизовану інтеграцію плат, без необхідності налаштовувати зовнішній брокер. Для Arduino Uno та ESP модулів потрібно додатково встановити Sketch Converter. Перевага сервісу у тому, що він автоматично підтримує TLS-з’єднання «TLS (Transport Layer Security) – це криптографічний протокол, спеціальні правила, які допомагають захистити приватність та безпеку користувачів всесвітньої мережі. Вони допомагають зробити інформацію, яка

передається через Інтернет, недоступною для зловмисників» [6], та свідоцтва автентичності.

Cayenne, ще один інструмент, що пропонує веб-конструктор дашбордів і готові шаблони, для управління реле, освітленням, зчитування температури та вологості. У конструкторі задаються правила автоматизації, а Arduino, через MQTT підписується на власний брокер.

Node-RED – це візуальна платформа для створення процесів за допомогою блоків. Arduino підключається до сервера через MQTT або HTTP, а дані надходять у потоки.

ThingsBoard, безкоштовна IoT-платформа з відкритим кодом, що підтримує MQTT, CoAP «CoAP – це протокол для взаємодії між пристроями IoT (Інтернет речей), які мають обмежені ресурси» [7] і HTTP. Після розгортання на сервері, користувач отримує редактор дашбордів, механізми правил та інструменти для керування пристроями. Web UI дозволяє формувати багаторівневі карти з різними типами віджетів і створювати складні ланцюжки обробки повідомлень.

# РОЗДІЛ 2
# ОБГРУНТУВАННЯ ВИБОРУ АПАРАТНОЇ ТА ПРОГРАМНОЇ БАЗИ

## 2.1 Характеристики та порівняння Arduino Uno і Arduino NodeMCU

Розглянемо дві основні платформи, які використовуються у рамках практичної частини дипломної роботи, а саме Arduino Uno і NodeMCU. Вибір був здійснений в залежності від апаратних потреб модулів системи, а саме LCD1602A, KY-016, DHT11. Порівняння основних параметрів дозволяє обгрунтувати вибір кожної плати з точки зору архітектури, ресурсів пам’яті, кількості та типу вводів і виводів, а також мережевих можливостей енергоспоживання.

### 2.1.1 Arduino Uno

Arduino Uno базується на 8-бітному ATmega328P з сімейства AVR «AVR – родина восьмибітових мікроконтролерів фірми Atmel. Мікроконтролери AVR мають гарвардську архітектуру (програма і дані розташовані в різних адресних просторах) і систему команд, близьку до ідеології RISC» [8]. Він має свою архітектуру, у якій команда та дані розміщуються в окремих сегментах пам’яті, що спрощує декодування інструкцій і пришвидшує роботу. Робоча тактова частота складає 16 МГц, чого достатньо для опитування цифрових та аналогових ввідних сигналів, обробки простих алгоритмів, керування й обміну даними.

У ATmega328P є у наявності 32 КБ флеш-пам’яті, для розміщення коду та самої програми, 2 КБ SRAM «SRAM – це інший тип оперативної пам’яті, який зберігає дані у статичному стані, без потреби у постійному оновленні» [9] для динамічних змінних і стеку, також 1 КБ EEPROM «Це тип пам’яті ПЗУ, тобто енергонезалежна пам’ять, в якій дані будуть зберігатися постійно, навіть якщо джерело живлення вимкнено» [10] для зберігання конфігураційних параметрів між перезавантаженнями. Обмежена SRAM вимагає обережності при роботі з великими буферами або числовими масивами, але її достатньо для проєктів із невеликою кількістю одночасно активних даних.

На Arduino Uno є 14 цифрових виводів, що можна побачити на рисунку 2.1, 6 з яких підтримують ШІМ «ШІМ є цифровим сигналом прямокутної форми, в якому рівень сигналу перемикається між логічним нулем та одиницею» [11] із роздільною здатністю 8 біт, корім того, на платі знаходиться 6 каналів аналогового вводу, який дозволяє вимірювати напругу.


Рисунок 2.1 – Arduino Uno [12]

Кожен аналоговий вхід може використовуватись, як цифровий, тому загальна кількість каналів дорівнює 20. Плату оснащено лінійним регулятором, що перетворює напругу в 7, або 12 В в 5 В і 3,3 В. USB-порт може забезпечити живлення до 500 мА, а лінійний стабілізатор вимикає надлишок напруги, що робить плату надійною. Для зв’язку із периферією використовуються UART, «UART – це зазвичай окрема мікросхема чи частина мікросхеми, що використовується для з’єднання через комп’ютерний чи периферійний

послідовний порт» [13] SPI, «SPI (англ. Serial Peripheral Interface) – це послідовний синхронний інтерфейс, який є чотирьох-провідним, та призначений для повнодуплексного обміну даними між ведучим та підлеглими пристроями» [14] I2C «I2C – послідовна шина даних для зв’язку інтегральних схем, розроблена фірмою Philips на початку 1980-х як проста шина внутрішнього зв’язку для створення керуючої електроніки» [15].

2.1.2 NodeMCU

NodeMCU базується на 32-бітному мікроконтролері Tensilica L106 з тактовою частотою 160 МГц, який є інтегрованим у ESP8266. На платі знаходиться повноцінний Wi-Fi-модуль, що дозволяє виконувати підключення, як до існуючої мережі, так і створювати власну точку доступу. Програмний стек взаємодіє з апаратними блоками, що забезпечує швидку обробку з’єднань і мінімальне навантаження на процесор.

NodeMCU комплектується зовнішньою SPI-флеш пам’яттю об’ємом 4 Мб, у якій знаходиться прошифка, SPIFFS «SPIFFS (SPI File System) – це легка файлова система, розроблена для вбудованих систем, де іноді необхідно працювати з файлами для частого читання та запису, таких як реєстрація даних, збереження кодів доступу або в веб-сервісах» [16], OTA-оновлення «Оновлення по повітрю (OTA) є бездротова доставка нового програмного забезпечення, прошивки або інших даних на мобільні пристрої» [17].

На платі розведено 10 універсальних виводів, які можна програмно налаштовувати, використовувати для вводу та виводу сигналів, що робить їх надзвичайно гнучкими для підключення різноманітних периферійних пристроїв, таких, як датчики, кнопки, реле та індикатори, також на платі присутній один аналоговий вхід, який здатен вимірювати напругу до 1 В, вбудовані інтерфейси SPI та I2C, які ідельно підходять для роботи з численними датчиками температури, вологості, тиску, для підключення зовнішніх пристроїв, зображення плати на рисунку 2.2.

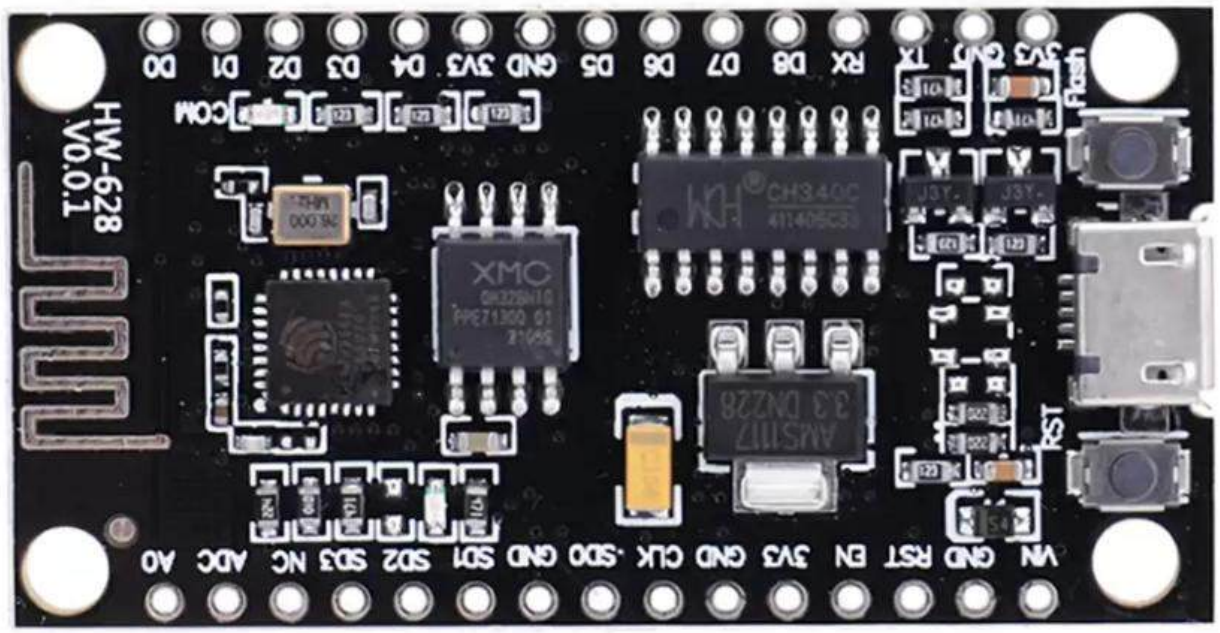


Рисунок 2.2 – NodeMCU [18]

У кожного вивода є можливість переналаштовування для різних функцій, що робить плату дуже універсальною і гнучкою у використанні.

Мікроконтролер підтримує 3 режими роботи, для зниження енергоспоживання, що дозволяє використовувати його у автономних пристроях.

У режимі modem sleep, Wi-Fi з’єднання підтримується, але передача даних зупинена, у цьому режимі споживання становить 15 мА. Режим light sleep вимикає частину периферійних блоків, не впливаючи на роботу ядра, споживання становить від 1 до 4 мА. Deep sleep, більшість системних компонентів, включно з ядром, вимикаються, активним залишається тільки годиннк реального часу, який може періодично вмикати пристрій, споживання струму у цьому режимі від 20 до 60 мікроампер.

На основі технічних характеристик та можливостей Arduino Uno та NodeMCU, можна зробити висновок про доцільність їх використання у рамках єдиної IoT-системи. Обидві платформи мають свої переваги та недоліки і вибір кожної з них обумовлений специфікою завдань, які виконуються у проєкті.

Переваги Arduino Uno, це стабільність, простота підключення і живлення 5 В, що робить її оптимальним вибором для підключення LCD1602A. Дисплей

потребує кількох цифрових ліній для передачі даних і сигналів керування. Arduino має достатню кількість виводів для прямого підключення дисплея, без використання розширювачів, що значно спрощує реалізацію. Постійна напруга 5 В гарантує стабільну роботу підсвітки та контрасту. Завдяки бібліотекам, таким як LiquidCrystal і підтримці середовища Arduino IDE, вивід текстової інформації може бути реалізований без великого досвіду програмування.

NodeMCU на базі ESP8266 є потужнішим у обробці даних і має вбудований Wi-Fi модуль, що значно спрощує підключення до MQTT-брокера. Це робить його ідеальним вибором для реалізації сенсорно-виконавчого блоку системи. Керування лампочкою потребує швидкого реагування на команди у реальному часі. NodeMCU дозволяє не лише приймати команди, а ще й формувати їх у зручному форматі і обробляти, завдяки чудовій тактовій частоті і наявності достатнього обсягу оперативної пам’яті.

Важливим пунктом є можливість енергозбереження. Підключений сенсор температури і вологості DHT11 опитується з періодичністю в 10 секунд, після чого пристрій може перейти у режим сну, споживаючи мінімальний струм.

Підсумовуючи, об’єднання Arduino Uno та NodeMCU у межах однієї системи, дозволяє досягти балансу, стабільності та енергоефективності, простототи реалізації рішень та потужної обробки мережевих запитів.

### 2.2 Датчик DHT11, принцип роботи та особливості застосування

Найпоширеніший у бюджетному сегменті сенсорів для вимірювання температури та вологості є DHT11. Цей модуль, що зображений на рисунку 2.3, відзначається простотою використання, низьким енергоспоживанням та широкою підтримкою. DHT11 – це цифровий сенсор, який має термістор, для визначення температури і ємнісний сенсор для вимірювання вологості повітря. Перевага модуля полягає у його здатності генерувати цифрові дані на виході, що значно спрощує взаємодію з мікроконтролерами та усуває потребу в додатковому аналого-цифровому перетворенні.


Рисунок 2.3 – DHT11 [19]

DHT11 дозволяє вимірювати температуру в діапазоні від 0 до 50° С з точністю 2° С та відносну вологість від 20 % до 90 % з точністю 5 %. Частота оновлення даних складає 1 Гц, тобто передача інформації про температуру та вологість відбувається раз на секунду. Такої швидкості достатньо для більшості задач, від моніторингу клімату, до керування системами поливу. Датчик перетворює фізичні параметри середовища на електричні сигнали.

Комунікація між DHT11 і мікроконтролером здійснюється через однопровідний протокол зв’язку. Зазвичай модуль DHT11 виготовляється у вигляді трививідного або чотирививідного блоку, що містить сам сенсор, резистор та вбудований стабілізатор напруги.

Сенсор є енергоефективним, його споживання не перевищує 2.5 мА у робочому режимі і близьке до нуля у режимі очікування. Завдяки цьому DHT11 підходить для автономних IoT-пристроїв, які працюють від батареї або в режимі сну.

До переваг DHT11 можна віднести низьку вартість, простоту підключення та програмування, цифровий вихід, який не потрібно калібрувати, висока сумісність із Arduino і NodeMCU.

Серед недоліків невисока точність, вузький діапазон температур і вологості, повільне оновлення даних, чутливість до таймінгів у протоколі обміну.

DHT11 використовується разом із платою NodeMCU для збору даних, які передаються через MQTT-протокол на веб-інтерфейс, таким чином, користувач має змогу дивитись значення температури та вологості у режимі реального часу через браузер.

DHT11 – це доступне та просте у використанні рішення для моніторингу навколишнього середовища. Саме тому DHT11 широко застосовується у навчальних цілях, побутових системах контролю клімату та базових IoT-проєктах.

### 2.3 KY-016, конструкція та методи керування кольором

Візуальна індикація стану пристроїв є невід’ємною частиною сучасних систем Інтернету речей, тому значне місце посідають RGB-світлодіоди. Одним із найзручніших, це модуль KY-016, який поєднує три світлодіоди в одному корпусі і дозволяє відтворювати широкий спектр кольорів шляхом змішування основних компонентів. Цей модуль, що зображений на рисунку 2.4 широко використовується у системах, зокрема на базі Arduino та NodeMCU, ефективно використовується для створення світлових індикаторів різного призначення: від сповіщень про стан системи та тривожних сигналів до декоративного підсвічування та забезпечення зворотного зв’язку з користувачем через кольорові сигнали. В основі модуля KY-016 лежить конструкція зі спільним анодом, позитивний вивід є спільним для всіх трьох основних кольорів, для трьох окремих світлодіодів, укладених у єдиний прозорий корпус. Модуль оснащений чотирма виводами, один загальний анод та три індивідуальні катоди для кожного колірного каналу.

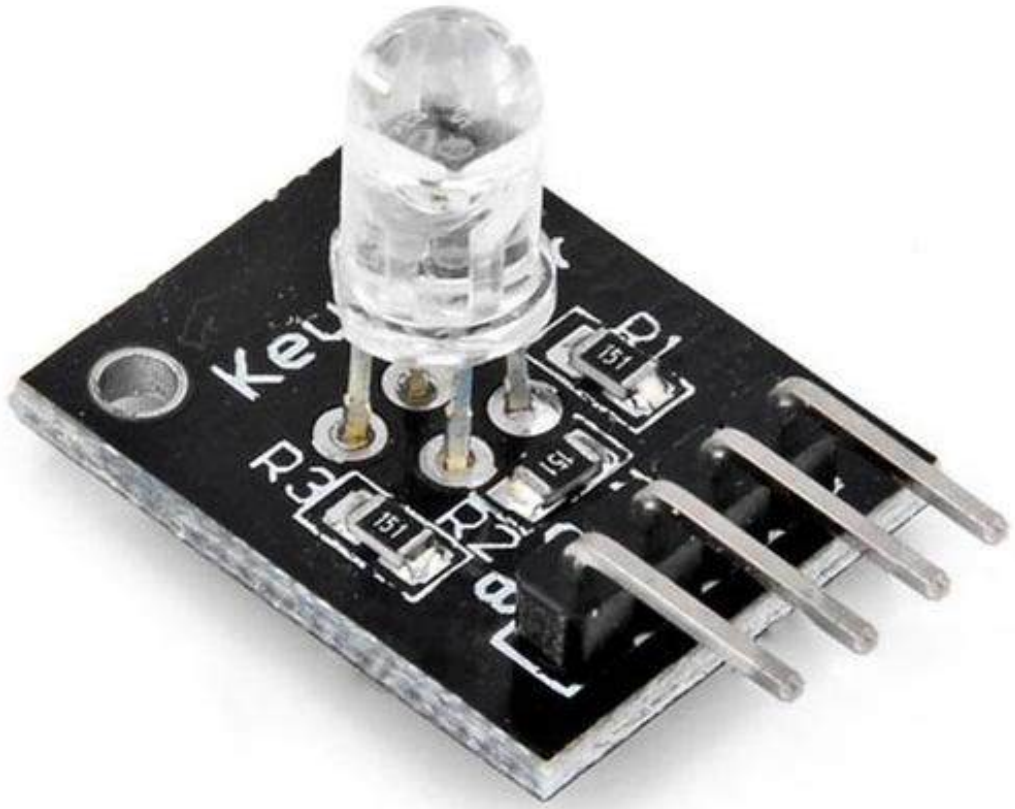

Рисунок 2.4 – KY-016 [20]

Така конфігурація дозволяє незалежно керувати кожним світлодіодом, активуючи його подачею сигналу низького рівня на відповідний катод.

Світлодіоди модуля KY-016 не мають вбудованих резисторів, тому використання зовнішніх резисторів є обов’язковим для уникнення перегріву або виходу з ладу компонентів. Резистори підбираються відповідно до напруги живлення та бажаної яскравості. Світло, яке випромінює модуль KY-016, формується шляхом змішування трьох базових кольорів, а саме червоного, зеленого та синього. У залежності від сили струму, що протікає через кожен світлодіод, інтенсивність відповідного кольору змінюється. У результаті змішування можна отримати понад 16,7 млн відтінків, хоча реальна кількість кольорів залежить від роздільної здатності керуючого сигналу.

Існує кілька основних методів керування світінням KY-016 у системах. Найпростіший спосіб це підключення кожного з каналів до окремого цифрового

виводу Arduino або NodeMCU, встановлюючи стан HIGH або LOW, можна вмикати або вимикати відповідний світлодіод. Такий спосіб підходить для відображення фіксованих кольорів, але не дає змоги гнучко змінювати яскравість або формувати плавні переходи між відтінками.

Для точного регулювання яскравості кожного каналу використовують ШІМ «Широтно-імпульсна модуляція (ШІМ) – це технологія управління яскравістю моніторів, заснована на швидкому перемиканні підсвічування екрану шляхом регулярного ввімкнення та вимкнення світлодіодів, що формують підсвічування» [21]. У цьому режимі мікроконтролер генерує сигнали з певним циклом, який визначає середню потужність, що подається на світлодіод. На платі Arduino Uno 6 цифрових виводів підтримують ШІМ, для NodeMCU ШІМ доступне на більшості GPIO, хоч і реалізується програмно. Завдяки ШІМ можна створити плавну анімацію, динамічні ефекти або реагування на зміну зовнішніх параметрів, таких як температура, час доби або значення з датчиків.

У складніших проєктах кольором можна керувати віддалено через інтерфейс користувача. На веб-сторінці, користувач вибирає бажаний колір через колірну палітру, а значення RGB надсилаються на мікроконтролер через MQTT-брокер. Таким чином, значення яскравості кожного кольору передаються через мережу, а мікроконтролер виконує їх на відповідних ШІМ-виводах, змінюючи інтенсивність світіння.

## 2.4 Дисплей LCD1602A, характеристики та особливості підключення

Найпоширеніший спосіб виводу текстової інформації в системах на базі Arduino є використання дисплея LCD1602A на основі контролера HD44780. Даний модуль має низьке енергоспоживання, просто підключається, а також повністю сумісний з більшістю платформ, зокрема Arduino, ESP8266, ESP32 та STM32 «STM32 – це сімейство 32-х бітних мікроконтролерів на базі процесорів ARM з архітектурами ядер Cortex-M7F, Cortex-M4F, Cortex-M3, Cortex-M0+ або Cortex-M0» [22]. LCD1602A широко застосовується в проєктах,

де необхідно відображати дані від сенсорів, повідомлення користувачу або поточні параметри роботи пристрою.

LCD1602A, який зображений на рисунку 2.5, це символьний рідкокристалічний дисплей, призначений для відображення 2 рядків по 16 символів. Кожен символ формується у сітці 5×8 пікселів. Загалом модуль підтримує відображення до 224 різних ASCII-символів, включаючи букви, цифри, знаки пунктуації та деякі псевдографічні елементи.

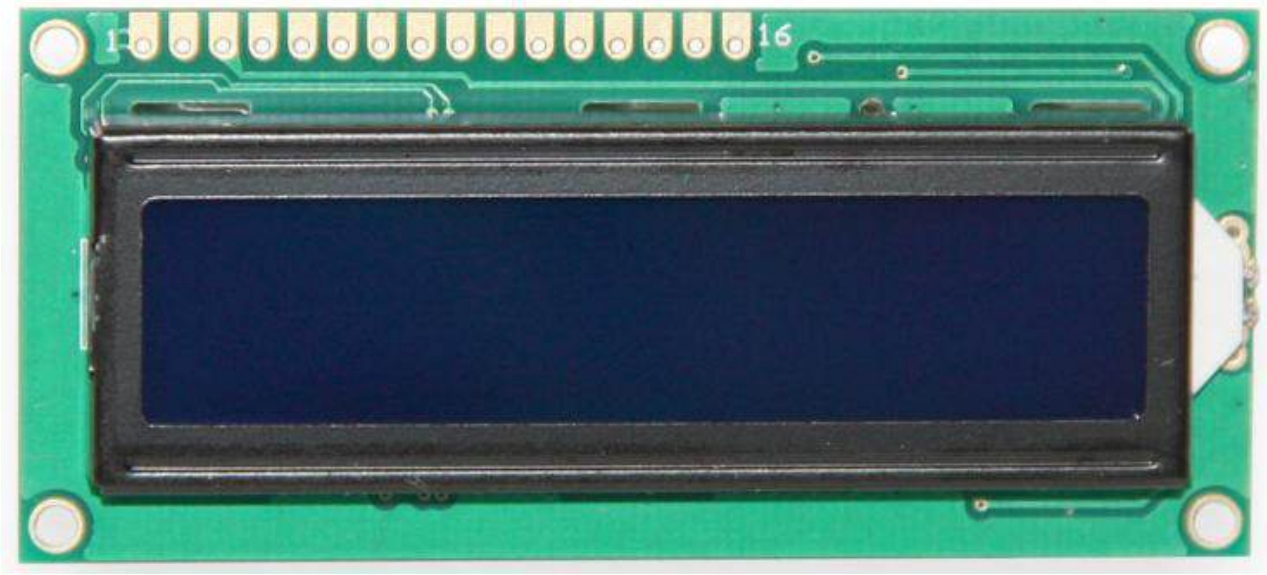


Рисунок 2.5 – DHT11 [23]

Живлення дисплея стандартно становить 5 В. Підсвітка вмикається за допомогою окремих пінів, а рівень контрасту регулюється через змінний резистор, підключений до виводу V0. Для зменшення кількості використовуваних виводів зазвичай застосовується 4-бітний режим. Поширеним способом підключення є пряме з’єднання з Arduino Uno або NodeMCU. Для економії портів часто використовують модуль розширення I2C–LCD, який базується на чипі PCF8574. Такий модуль дозволяє підключити дисплей через шину I2C, що значно зменшує кількість задіяних виводів і спрощує підключення до ESP8266, де кількість вільних GPIO обмежена.

Дисплей LCD1602A часто використовується у системах моніторингу, де потрібно виводити текстову інформацію без участі комп’ютера чи

веб-інтерфейсу. У поєднанні з Arduino, він ідеально підходить для автономних пристроїв.

## 2.5 Протоколи обміну даними HTTP і MQTT: переваги та сфери використання

HTTP – це клієнт-серверний протокол прикладного рівня, розроблений для передачі гіпертексту. Він працює за принципом «запит-відповідь», клієнт надсилає запит, а сервер повертає відповідь. HTTP переважно використовується у традиційній веб-розробці, REST API та односторонніх трансакціях.

MQTT – це легковаговий протокол публікації-підписки, оптимізований для мереж із низькою пропускною здатністю та пристроїв із обмеженими ресурсами. Беручи до уваги таблицю 2.1, можна дійти до висновку, що протокол чудово підходить для взаємодії великої кількості IoT-пристроїв, які потребують мінімального енергоспоживання та швидкого обміну короткими повідомленнями.

Таблиця 2.1 – Порівняння HTTP і MQTT

| Характеристика | HTTP | MQTT |
|---|---|---|
| Модель взаємодії | Клієнт-сервер (запит-відповідь) | Публікація-підписка (через брокера) |
| Тип з’єднання | Зазвичай короткочасне, встановлюється для кожного запиту/відповіді | Може бути довготривалим |
| Накладні витрати (заголовки) | Відносно великі | Дуже малі |
| Призначення | Передача гіпертексту, веб-ресурсів, REST API | Обмін повідомленнями в IoT, M2M комунікації |
| Енергоефективність | Менш ефективний для пристроїв з обмеженим живленням | Оптимізований для низького енергоспоживання |
| Надійність доставки (QoS) | Забезпечується на рівні TCP | Має три рівні QoS |
| Масштабованість для IoT | Може бути складною для великої кількості одночасних з’єднань | Добре масштабується для великої кількості підключених пристроїв через брокера |
| Використання пропускної здатності | Більше через розмір заголовків | Менше, оптимізований для мереж з низькою пропускною здатністю |

Продовження таблиці 2.1

| Характеристика | HTTP | MQTT |
|---|---|---|
| Стан сесії | Без стану | Може підтримувати стан сесії |
| Зв’язок | Переважно односпрямований | Двонаправлений, асинхронний |

Однією з головних переваг HTTP є його універсальність і сумісність з будь-якими браузерами, мобільними застосунками та серверними технологіями. Завдяки простоті реалізації HTTP дозволяє швидко створювати веб-інтерфейси та REST-сервіси «REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів» [24], що забезпечують управління та моніторинг пристроїв.

MQTT був створений спеціально для систем з обмеженою пропускною здатністю, високою затримкою та обмеженим енергоспоживанням. Його основною особливістю є асинхронна передача даних через брокер, що дозволяє клієнтам надсилати повідомлення лише тоді, коли це потрібно, і не утримувати постійне з’єднання з усіма іншими учасниками.

MQTT використовує модель, де брокер, виступає посередником між пристроями. Така архітектура дозволяє спростити масштабування системи.

## 2.6 Вибір MQTT-брокера HiveMQ: обґрунтування

HiveMQ – це комерційна та хмарна реалізація MQTT-брокера, створена німецькою компанією GmbH. Сервіс надає надійний, безпечний і масштабований канал обміну даними між пристроями, орієнтований на використання у промислових та корпоративних IoT-рішеннях. Окрім платної комерційної версії, HiveMQ також пропонує безкоштовний публічний брокер, що дозволяє реалізовувати прототипи, навчальні та особисті проєкти без необхідності розгортання власного MQTT-сервера.

Брокер повністю підтримує специфікацію MQTT 3.1.1 і 5.0, забезпечує високу продуктивність, підтримує TLS-шифрування «Протокол TLS шифрує інтернет-трафік усіх видів, тим самим роблячи безпечними спілкування та продаж в інтернеті» [25], механізми авторизації користувачів, інтеграцію з базами даних, панель адміністрування і веб-сокети для взаємодії з браузером.

Одним з вирішальних факторів для студентських та прототипних проєктів є доступність безкоштовного брокера без потреби розгортання окремого сервера. HiveMQ надає відкриту адресу, яка працює на стандартному порту і не вимагає автентифікації. Це значно спрощує початкове налаштування та дає змогу зосередитися на логіці проєкту, а не на конфігурації серверної частини.

Ще однією важливою перевагою є можливість підключення з веб-додатків безпосередньо через WebSocket. HiveMQ підтримує підключення по порту 8000, що дозволяє клієнтам, безпосередньо обмінюватися повідомленнями з брокером. Така сумісність спрощує реалізацію динамічного веб-інтерфейсу без необхідності використовувати додаткові проксі-сервери або конвертери протоколів.

## 2.7 Вибір фронтенд-бекенд стеку

React надає декларативний підхід, дозволяє описувати вигляд інтерфейсу через JSX-розмітку та автоматично оновлювати його при зміні стану, що є ідеальним для роботи з динамічними даними, такими як температура та вологість, забезпечуючи миттєве оновлення без повного перезавантаження сторінки. Компонентна архітектура React дає змогу структурувати інтерфейс на самостійні, перевикористовувані частини, що значно спрощує розробку, підтримку та модифікацію коду.

Використання фреймворку Next.js розширює можливості React, зокрема завдяки підтримці гібридного рендерингу, що дозволяє, пререндерити на сервері сторінку налаштувань чи початковий дашборд для швидшого першого завантаження, а потім оновлювати дані на клієнті для плавної взаємодії.

Yarn забезпечує паралельне завантаження пакетів, що збільшує швидкість їх встановлення порівняно з npm.

# РОЗДІЛ 3
# ПРАКТИЧНА РЕАЛІЗАЦІЯ

## 3.1 Загальна архітектура рішення

Розроблена система побудована за модульним принципом і включає три основних пристрої, що взаємодіють з централізованим веб-інтерфейсом через MQTT-брокер HiveMQ. Ця архітектура забезпечує незалежну роботу кожного вузла та гнучке керування через веб-додаток.

Апаратна частина системи складається з вузла моніторингу температури та вологості, реалізованого на базі NodeMCU з підключеним датчиком DHT11. Другим компонентом є вузол керування RGB-підсвіткою, також реалізований на базі NodeMCU, що керує RGB-світлодіодом KY-016. Третій вузол відповідає за відображення інформації і складається з Arduino Uno, до якого підключено LCD-дисплей 1602A через I2C, інтернет-зв’язок для цього вузла та отримання даних з веб-інтерфейсу через MQTT забезпечує окремий модуль ESP8266, який передає інформацію на Arduino Uno.

Веб-інтерфейс, розроблений на Next.js, надає користувачеві можливість відстежувати показники температури та вологості, керувати кольором RGB-підсвітки та виводити текстові повідомлення на LCD-дисплей. Взаємодія між веб-додатком та мікроконтролерними вузлами відбувається асинхронно через MQTT-брокер.

Схема потоків даних для моніторингу температури та вологості наступна NodeMCU з DHT11 зчитує дані, форматує їх у JSON та публікує в MQTT-топік dht11/data. Бекенд веб-додатку підписаний на цей топік, отримує дані та передає їх на фронтенд для відображення. Для керування RGB-підсвіткою користувач обирає колір на веб-інтерфейсі фронтенд передає RGB-значення на бекенд, який потім публікує їх у JSON-форматі в MQTT-топік rgb_controller/color, на який підписаний другий NodeMCU, що встановлює відповідний колір світлодіода. Відображення інформації на LCD відбувається так: користувач вводить повідомлення або обирає дані на веб-інтерфейсі, фронтенд передає текст на

бекенд, який публікує його в MQTT-топік lcd/messaging, модуль ESP8266, що забезпечує інтернет-зв’язок для Arduino Uno, отримує це повідомлення та передає по послідовному порту на Arduino Uno, який виводить текст на LCD-дисплей.

### 3.2 Прошивки компонентів

Програмне забезпечення для всіх мікроконтролерів розроблено в середовищі Arduino IDE мовою C++, з використанням відповідних бібліотек для кожного завдання.

3.2.1 Система моніторингу температури та вологості на базі NodeMCU

Перший вузол на базі NodeMCU відповідає за збір та передачу даних про мікроклімат. До цифрового піну NodeMCU підключено датчик температури та вологості DHT11, за схемою на рисунку 3.1.

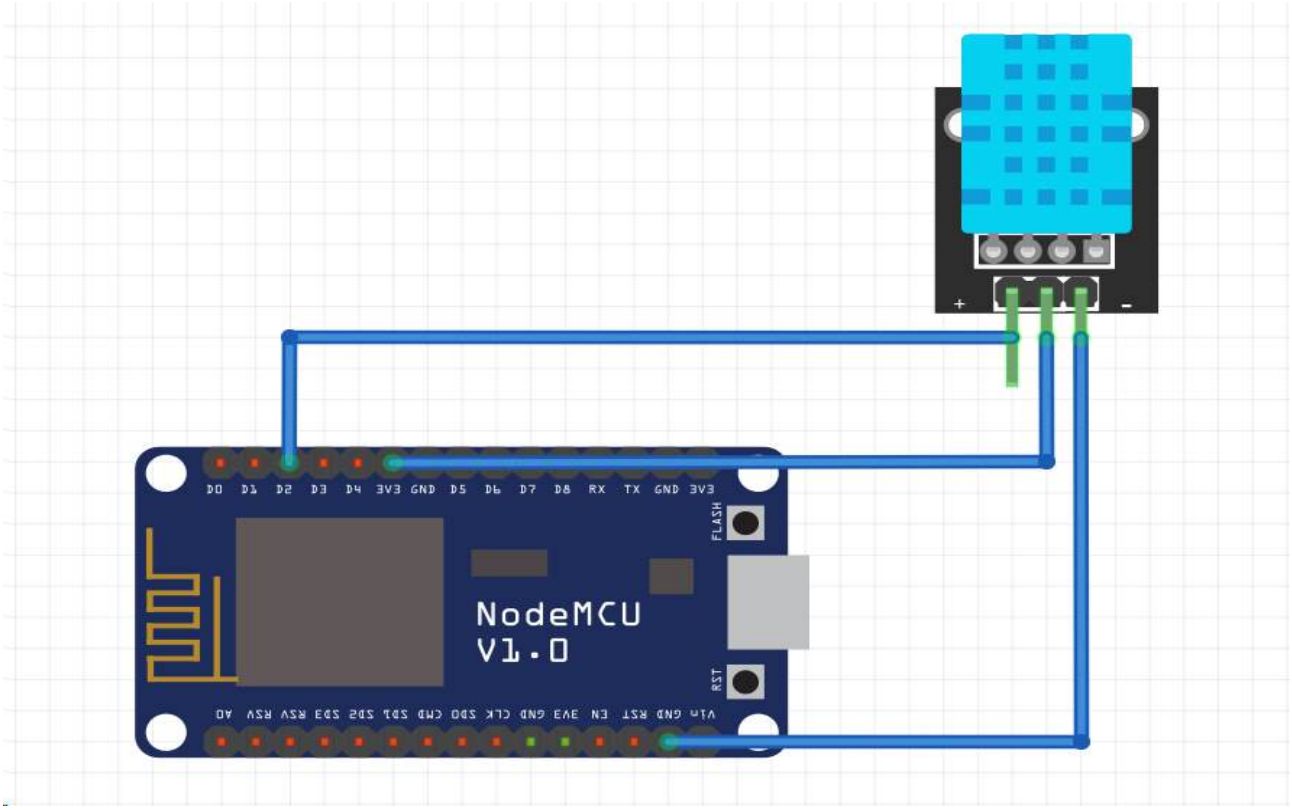


Рисунок 3.1 – Схема підключення DHT-11 до NodeMCU

Прошивка цього модуля забезпечує підключення до мережі Wi-Fi за допомогою бібліотеки ESP8266WiFi. Після успішного з’єднання з Wi-Fi, встановлюється зв’язок з MQTT-брокером HiveMQ, використовуючи бібліотеку PubSubClient. У головному циклі програми мікроконтролер з визначеним інтервалом зчитує показники температури та вологості з датчика DHT11. Отримані значення записуються у JSON-формат та публікуються у MQTT-топік dht11/data з прапорцем retain

3.2.2 Система керування RGB-підсвіткою на базі NodeMCU

Другий вузол, також на базі NodeMCU, призначений для керування RGB-світлодіодом KY-016. На рисунку 3.2, можемо побачити, що три контакти світлодіода підключені до GPIO NodeMCU, що підтримують функцію широтно-імпульсної модуляції (ШІМ).

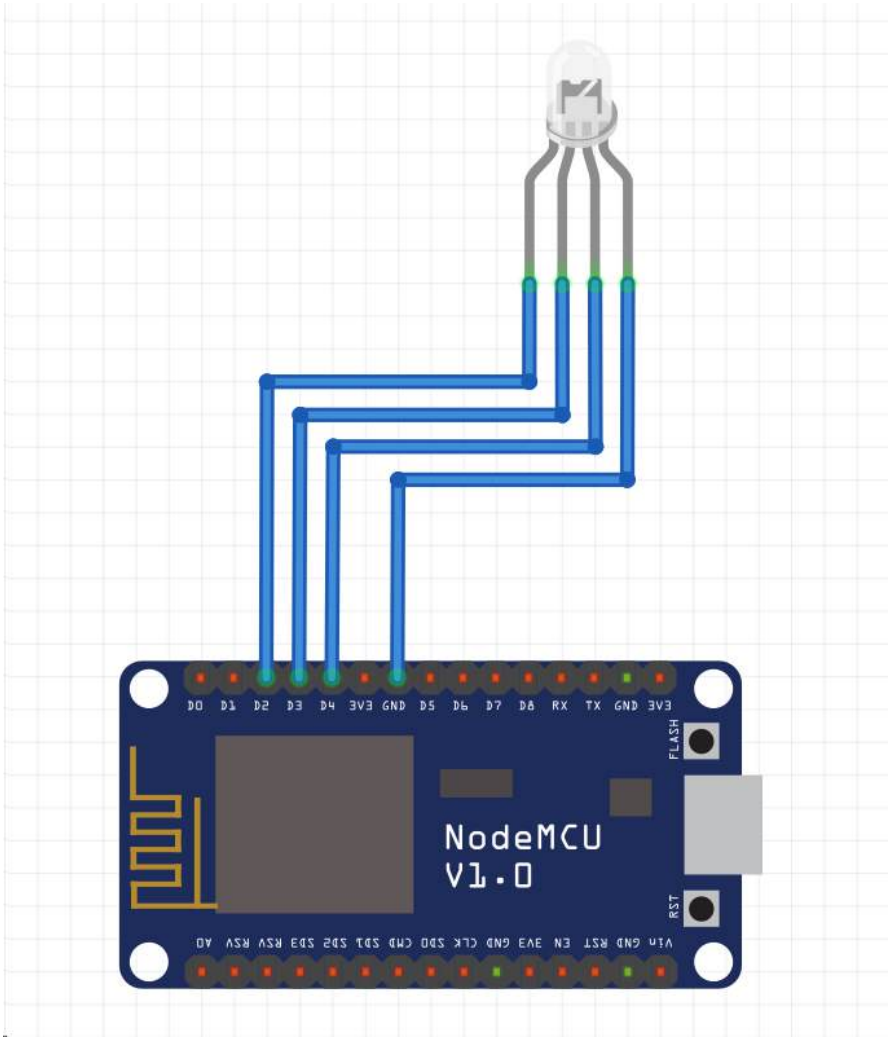


Рисунок 3.2 – Схема підключення KY-016 до NodeMCU

Програмна логіка аналогічна попередньому вузлу в частині підключення до Wi-Fi та MQTT-брокера. Цей NodeMCU підписується на MQTT-топік rgb_controller/color. При надходженні повідомлення у цей топік, функція обробляє отриманий JSON-рядок, який містить значення інтенсивності для червоного, зеленого та синього кольорів. Після парсингу JSON, мікроконтролер за допомогою функції analogWrite() встановлює відповідні значення ШІМ для кожного каналу, змінюючи колір та яскравість світіння RGB-світлодіода.

### 3.2.3 Система відображення інформації на LCD1602A

Третій вузол відповідає за виведення текстової інформації на рідкокристалічний дисплей LCD1602A. Цей вузол складається з Arduino Uno, який безпосередньо керує дисплеєм, та модуля ESP8266, схему підключення, можна побачити на рисунку 3.3, що забезпечує інтернет-зв’язок та виконує роль MQTT-клієнта.

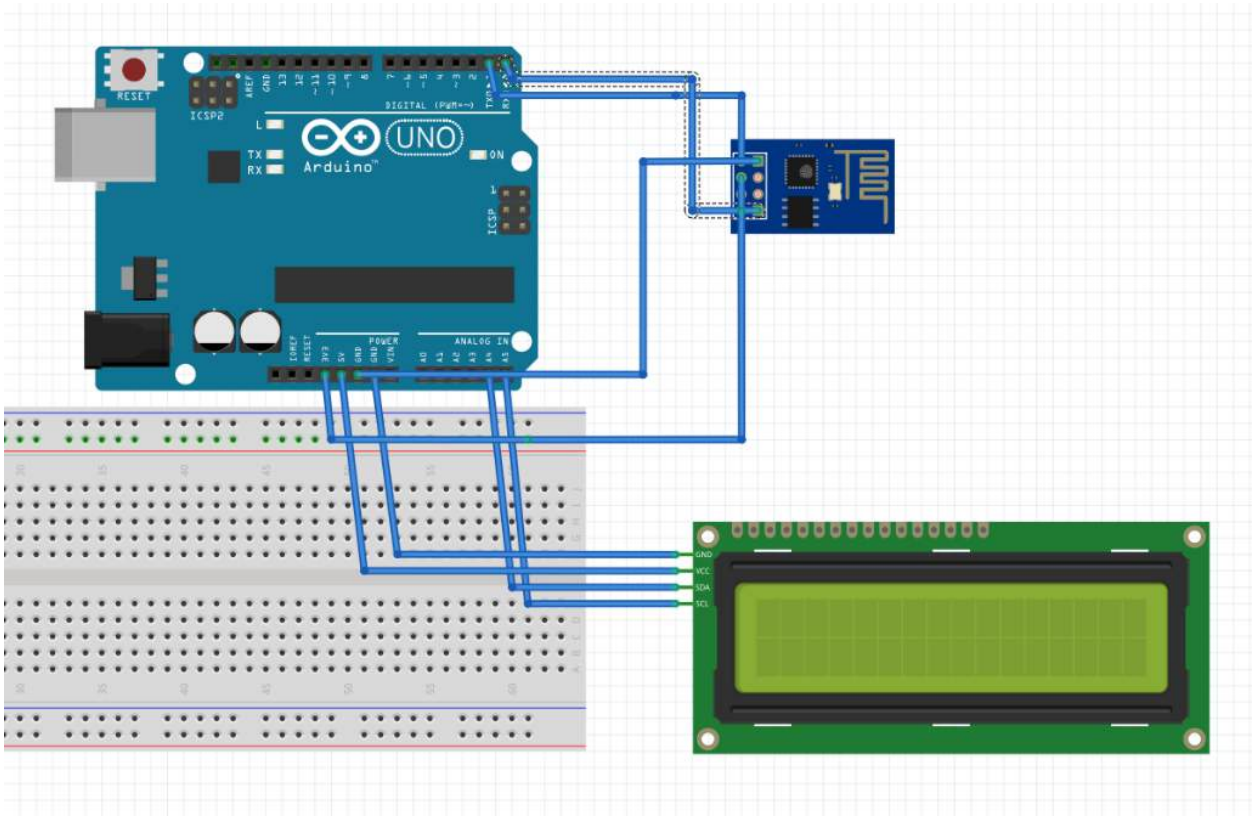


Рисунок 3.3 – Схема підключення ESP8266 і LCD1602A до Arduino Uno

Модуль ESP8266 підключається до Wi-Fi та MQTT-брокера, підписуючись на топік lcd/messaging. При отриманні повідомлення, ESP8266 передає цей JSON-рядок на Arduino Uno через послідовний інтерфейс. Arduino Uno, до якого через шину I2C підключений LCD1602A, використовуються бібліотеки Wire.h для I2C та LiquidCrystal_I2C.h для роботи з дисплеєм, приймає дані від ESP8266. Отриманий JSON-рядок парситься за допомогою бібліотеки ArduinoJson.h, і вилучені текстові значення виводяться на екран LCD-дисплея.

### 3.3 Розробка веб-інтерфейсу на React/Next.js

Веб-інтерфейс слугує центральною точкою для моніторингу та керування всіма трьома мікроконтролерами. Він розроблений на базі Next.js, що дозволяє поєднати фронтенд на React та бекенд-логіку у вигляді API Routes.

Бекенд-частина взаємодіє з MQTT-брокером HiveMQ за допомогою спеціального модуля connectMQTT. Для кожного з трьох пристроїв створено відповідні API. API для DHT11 (/api/dht) підключається до MQTT, підписується на топік dht11/data, зберігає останні отримані значення температури та вологості і повертає їх за GET-запитом. API для RGB-світлодіода (/api/setColor) обробляє GET-запити для отримання поточного кольору та POST-запити для встановлення нового, отримані RGB-значення публікуються в MQTT-топік rgb_controller/color. API для LCD1602A дозволяє отримувати поточні повідомлення та відправляти нові, які потім публікуються в MQTT-топік lcd/messaging.

Фронтенд-частина реалізована за допомогою React-компонентів, візуальну частину можна побачити на рисунку 3.4. Компонент Temperature відповідає за відображення даних температури та вологості, періодично оновлюючи їх шляхом запитів. Аналогічно, компонент ColorPicker надає користувачеві інтерактивні елементи, такі як слайдери та пресети кольорів, для вибору бажаного відтінку RGB-підсвітки, який потім передається на відповідний мікроконтролер.

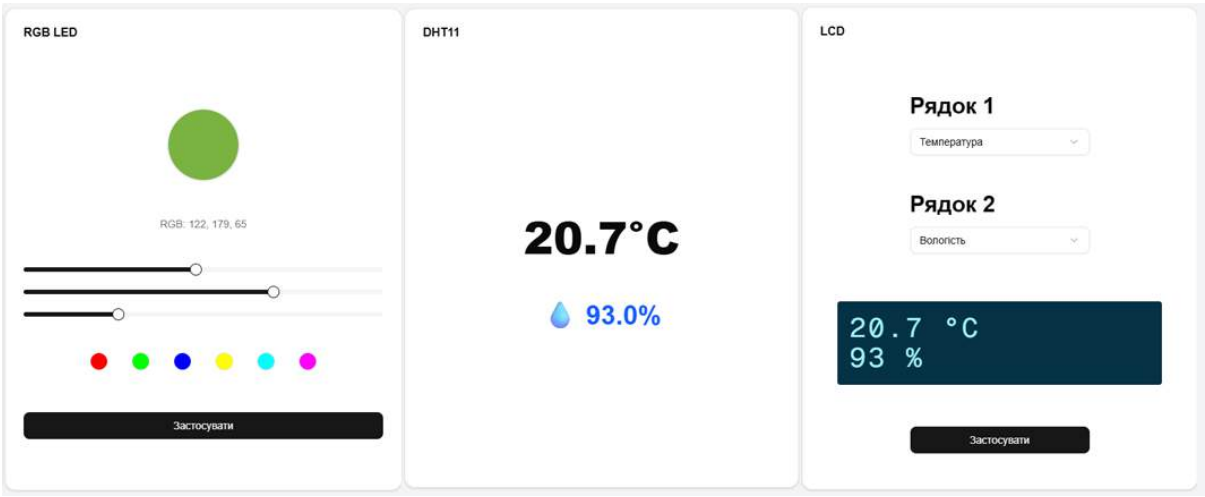


Рисунок 3.4 – Інтерфейс веб-додатку

Компонент ColorPicker дозволяє користувачеві обирати колір RGB-підсвітки за допомогою слайдерів або з набору пресетів, обраний колір відправляється на бекенд через /api/setColor. Компонент LCD надає можливість вибору даних або введення власного текстового повідомлення для відображення на LCD-екрані, ці дані передаються на /api/lcd. Фронтенд не підключається до MQTT безпосередньо, а взаємодіє з ним через описані API-роути бекенду.

### 3.4 Інтеграція та тестування системи

Етап інтеграції передбачав об’єднання всіх розроблених апаратних та програмних компонентів в єдину, злагоджено функціонуючу систему. Цей процес вимагав ретельного підходу та послідовного тестування для забезпечення коректної взаємодії між окремими вузлами та веб-інтерфейсом.

Першим кроком було поетапне тестування окремих модулів. Кожна група, включаючи NodeMCU з датчиком DHT11, NodeMCU з RGB-світлодіодом, а також Arduino Uno з модулем ESP8266 та LCD-екраном, проходила індивідуальну перевірку. На цьому етапі контролювалася стабільність підключення до мережі, коректність встановлення з’єднання з MQTT-брокером HiveMQ, правильність публікації даних, наприклад, з вузла з DHT11, та реакцію на отримані MQTT-повідомлення для вузлів керування RGB-світлодіодом та

відображення на LCD. Для відладки програмного коду мікроконтролерів активно використовувався серійний монітор, вбудований в Arduino IDE, що дозволяло в реальному часі відстежувати стан виконання програми та значення змінних. Для моніторингу та надсилання тестових MQTT-повідомлень застосовувалися зовнішні MQTT-клієнти, такі як MQTT Explorer, що давало змогу перевірити правильність формування топіків та вмісту повідомлень.

Паралельно проводилося тестування бекенд API-роутів веб-додатку. За допомогою інструментів типу Postman перевірялася логіка обробки GET та POST, а саме /api/dht, /api/setColor та /api/lcd. Також контролювалася коректність валідації вхідних даних та правильність взаємодії бекенду з MQTT-брокером, включаючи підписку на топіки та публікацію повідомлень. Фронтенд-компоненти тестувалися на коректність відображення отриманих даних, адекватну роботу інтерактивних елементів, таких як слайдери, поля вводу та кнопки, а також на правильність формування і відправки запитів на бекенд.

Після успішного завершення індивідуального тестування компонентів розпочався етап тестування всієї системи. Метою цього етапу було переконатися у правильній взаємодії всіх частин системи в реальних умовах експлуатації. Перевірялися різні сценарії використання, зміна кольору RGB-світлодіода через веб-інтерфейс та візуальна оцінка відповідності встановленого кольору на фізичному пристрої, динамічне оновлення показників температури та вологості на веб-інтерфейсі при зміні умов навколишнього середовища біля датчика DHT11, наприклад, шляхом його нагрівання або зволоження, відправка текстових повідомлень або вибір даних сенсорів для відображення на LCD-екрані через веб-інтерфейс та перевірка коректності їх виведення на фізичному дисплеї.

Також, перевірявся час відгуку, стабільність роботи та обробка помилок. Час відгуку вимірювався як затримка між дією користувача в веб-інтерфейсі, наприклад, відправкою команди на зміну кольору, та фактичною реакцією відповідного вузла. Також оцінювався час, необхідний для оновлення даних з датчиків на веб-сторінці. Стабільність роботи перевірялася тестуванням системи

впродовж тривалого періоду для виявлення можливих проблем, таких як розриви Wi-Fi або MQTT з'єднань, некоректне накопичення даних або зависання одного з мікроконтролерів. Обробка помилок та нештатних ситуацій передбачала перевірку реакції системи на можливі проблеми, такі як введення некоректних даних користувачем, тимчасова недоступність MQTT-брокера або відключення одного з мікроконтролерних вузлів від мережі, бекенд та фронтенд повинні були обробляти такі ситуації, надаючи користувачеві зрозумілі повідомлення про помилки.

У процесі інтеграції та тестування було виявлено та усунуто декілька дрібних недоліків, зокрема пов'язаних з форматуванням JSON-повідомлень для LCD та оптимізацією частоти оновлення даних з датчика DHT11 для зменшення навантаження на MQTT-брокер. Загалом, розроблена система продемонструвала стабільну та коректну роботу, відповідаючи поставленим функціональним вимогам.

### 3.5 Розгортання веб-додатку з використанням Docker

Для забезпечення простоти розгортання та портативності веб-додатку Next.js було використано технологію контейнеризації Docker. Процес розгортання включав створення файлу Dockerfile в кореневій директорії проєкту, який містить набір інструкцій для побудови образу, вибір базового образу Node.js, копіювання файлів проєкту, встановлення залежностей, компіляцію проєкту, визначення порту та команду для запуску. Побудова образу здійснювалася командою docker build -t arduino-iot-interface. Запуск контейнера з побудованого образу виконувався командою docker run -p 3000:3000 arduino-iot-interface, що робило веб-інтерфейс доступним у браузері.

# ВИСНОВКИ

Здійснено реалізацію прошивок для мікроконтролерів ESP8266 та NodeMCU з інтегрованою підтримкою протоколу MQTT, що забезпечує публікацію даних з датчика температури та вологості DHT11 та прийом команд для керування RGB-світлодіодом KY-016. Розробка велася у середовищі Arduino IDE з використанням мови програмування C++ та відповідних бібліотек, зокрема ESP8266WiFi для мережевих з'єднань і взаємодії з MQTT-брокером.

Спроєктовано та реалізовано веб-інтерфейс на базі технологій React та Next.js, який забезпечує підключення до MQTT-брокера HiveMQ. Даний інтерфейс дозволяє користувачеві в реальному часі керувати кольором RGB-світлодіода KY-016 та відображати актуальні показники температури і вологості, отримані з датчика DHT11. Взаємодія фронтенду з MQTT-брокером реалізована через спеціально розроблені API-роути на бекенді.

Виконано дослідження характеристик протоколів передачі даних HTTP та MQTT, проведено їх порівняльний аналіз щодо ефективності та особливостей використання в контексті систем Інтернету речей. Встановлено, що MQTT, завдяки своїй моделі публікації-підписки та оптимізації для пристроїв з обмеженими ресурсами, є більш доцільним для розробленої системи, ніж традиційний HTTP.

Візуалізовано отримані дані температури та вологості як у розробленому веб-інтерфейсі, так і на символьному дисплеї LCD1602A. Для виведення інформації на LCD-дисплей використано мікроконтролер Arduino Uno, що отримує дані через модуль ESP8266, підписаний на відповідний MQTT-топік.

Запропоновано шляхи подальшого вдосконалення розробленого проєкту. Серед них – впровадження захищеного TLS-з'єднання для обміну даними по протоколу MQTT, що підвищить безпеку системи; можливість масштабування системи шляхом підключення декількох однотипних сенсорів для моніторингу параметрів у різних точках, а також покращення адаптивності веб-інтерфейсу для коректного відображення та зручного використання на мобільних пристроях.

Таким чином, у рамках кваліфікаційної роботи було створено повноцінний прототип системи Інтернету речей з веб-інтерфейсом, продемонстровано практичне застосування мікроконтролерних технологій, протоколу MQTT та сучасних веб-фреймворків для вирішення задач віддаленого моніторингу та керування. Результати роботи підтверджують актуальність обраної теми та мають як практичну, так і наукову цінність.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розкриваємо поняття «Інтернет речей». InfoTel - Системний інтегратор телекомунікаційних рішень. URL: https://surli.cc/dwdndr (дата звернення: 10.04.2025).

2. Інтернет речі (IoT) в бізнесі: ключові тренди та переваги Vamark. Vamark. URL: https://surl.li/kbewru (дата звернення: 10.04.2025).

3. Business Intelligence, BI. IT-Enterprise your one-stop platform for digital transformation | www.it.ua. URL: https://surl.li/vktvnz (date of access: 13.05.2025).

4. Що таке ERP?. fw_error_www. URL: https://surl.li/bvwpef (дата звернення: 15.04.2025).

5. Що таке CRM-система та як вона пѕрацює? | Creatio. AI-Native Platform to Automate CRM and Workflows with No-Code Creatio.ai. URL: https://surl.li/juofls (дата звернення: 15.04.2025).

6. TLS протокол що це таке та як він захищає ваші дані в Інтернеті. Хостинг в Україні від Cityhost найкращий хостинг сайту. URL: https://surl.li/elqhgk (дата звернення: 01.05.2025).

7. CoAP протокол IT Master електроніка та програмування. Головна IT Master електроніка та програмування. URL: https://surli.cc/vmilfo (дата звернення: 01.05.2025).

8. Учасники проєктів Вікімедіа. AVR Вікіпедія. Вікіпедія. URL: https://surl.lu/ygbhpm (дата звернення: 05.05.2025).

9. Що таке оперативна пам’ять та її види?. TAUSOFT. URL: https://surl.lu/wijata (дата звернення: 05.05.2025).

10. EEPROM: все, що вам потрібно знати про цю пам’ять. Hardware libre. URL: https://surl.lu/efrdvw (дата звернення: 05.05.2025).

11. Arduino урок 4 PWM IT Master електроніка та програмування. Головна IT Master електроніка та програмування. URL: https://surl.gd/hkqsaa (дата звернення: 05.05.2025).

12. Плата Arduino UNO: продаж, ціна в Україні. Набори та компоненти для

самостійної збірки електроніки від «ArduinoKit навчальний набори». URL: https://surl.lu/yrcdik (дата звернення: 09.05.2025).

13. Учасники проєктів Вікімедіа. UART – Вікіпедія. Вікіпедія. URL: https://surl.li/scbbhy (дата звернення: 09.05.2025).

14. SPI інтерфейс IT Master електроніка та програмування. Головна IT Master електроніка та програмування. URL: https://surl.lu/sltumh (дата звернення: 09.05.2025).

15. Учасники проєктів Вікімедіа. I2C Вікіпедія. Вікіпедія. URL: https://surl.lu/oimhvr (дата звернення: 09.05.2025).

16. ESP32 Обробка Файлів за допомогою SPIFFS в ESP-IDF з кодом - javascript.org.ua JS Communities. javascript.org.ua JS Communities. URL: https://surl.li/bypujq (дата звернення: 09.05.2025).

17. Що таке оновлення через OTA?. Сучасні тенденції. URL: https://surl.li/tuepvp (дата звернення: 10.05.2025).

18. Контролер NodeMCU V3 WI-FI ESP8266, CH340 + 32MB flash ver. URL: https://surl.li/vzqfpr (дата звернення: 10.05.2025).

19. DHT11 Temperature Humidity Module. Makerfabs,Turnkey PCB Assembly Solution Provider. URL: https://surl.li/nkqrkl (дата звернення: 10.05.2025).

20. KY-016 Модуль RGB світлодіода Запчастини та комплектуючі для 3D принтера та верстатів. URL: https://surl.li/gfwvyh (дата звернення: 10.05.2025).

21. Шим у моніторах: що це таке, як виявити і чи існує можливість вирішити цю проблему URL: https://surl.li/imsfhp (дата звернення: 12.05.2025).

22. Мікроконтролери STM32 вступ IT Master електроніка та програмування. Головна IT Master електроніка та програмування. URL: https://surli.cc/omvonu (дата звернення: 13.05.2025).

23. LCD1602A 5v символьний дисплей синій фон URL: https://surl.li/mmmikk (дата звернення: 13.05.2025).

24. Учасники проєктів Вікімедіа. REST Вікіпедія. Вікіпедія. URL: https://surli.cc/ekwkel (дата звернення: 14.05.2025).

25. Що таке SSL/TLS протокол та остання версія TLS 1.3 Блог VPS.ua.

URL: https://surl.li/lonefb (дата звернення: 14.05.2025).

# ДОДАТКИ

# Додаток А

## Прошивка ESP8266 (ArduinoUno+ESP8266+LCD1602A)

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h> // For MQTT communication
#include <ArduinoJson.h>

// WiFi credentials
const char* ssid = "RedmiNote8Pro";
const char* password = "11112222";

// MQTT settings
const char* mqtt_server = "broker.hivemq.com"; // Public MQTT broker (change to
your broker)
const int mqtt_port = 1883;
const char* mqtt_client_id = "ESP8266Client-"; // Will be appended with chip ID
const char* mqtt_topic = "lcd/messaging"; // Topic to subscribe to

WiFiClient espClient;
PubSubClient client(espClient);
long lastReconnectAttempt = 0;

void setup_wifi() {
 delay(10);
 WiFi.begin(ssid, password);

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

}

// Callback when MQTT message is received
void callback(char* topic, byte* payload, unsigned int length) {

 // Convert payload to string and forward to Arduino
 String message;
 for (int i = 0; i < length; i++) {
  message += (char)payload[i];
 }

 Serial.println(message);
}

// Reconnect to MQTT broker
boolean reconnect() {
 String clientId = mqtt_client_id + String(ESP.getChipId());

 if (client.connect(clientId.c_str())) {
  // Subscribe to topic
  client.subscribe(mqtt_topic);
 }
 return client.connected();
}

void setup() {
 Serial.begin(9600);
 setup_wifi();
 client.setServer(mqtt_server, mqtt_port);
 client.setCallback(callback);
```

```
 lastReconnectAttempt = 0;
}

void loop() {
 if (!client.connected()) {
  long now = millis();
  if (now - lastReconnectAttempt > 500) {
   lastReconnectAttempt = now;
   // Attempt to reconnect
   if (reconnect()) {
    lastReconnectAttempt = 0;
   }
  }
 } else {
  // Client connected - let the library handle incoming messages
  client.loop();
 }
}
```

# Додаток Б

## Прошивка Arduino Uno (ArduinoUno+ESP8266+LCD1602A)

```
#include <ArduinoJson.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

SoftwareSerial espSerial(1, 0);
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {

 espSerial.begin(9600);
 lcd.init();
 lcd.backlight();
}
void loop() {

 if (espSerial.available()) {
  String payload = espSerial.readStringUntil('\n');

  // Parse JSON
  StaticJsonDocument<128> doc;
  DeserializationError error = deserializeJson(doc, payload);

  if (error) {
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("ERROR");
   delay(3000);
   lcd.clear();
   return;
  }

  // Convert to String
  String msg1 = String(doc["msg1"].as<const char*>());
  String msg2 = String(doc["msg2"].as<const char*>());

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(msg1);
  lcd.setCursor(0, 1);
  lcd.print(msg2);

 }
 delay(1000);
}
```

# Додаток В

## Бекенд частина (ArduinoUno+ESP8266+LCD1602A)

```
import { connectMQTT } from '@/lib/mqttClient'

export interface Messages {
  msg1: string
  msg2: string;
}

let currentMessages: Messages = { msg1: "", msg2: "" };

// For app router version (API Routes)
// Import in Next.js app router format
export const GET = async () => {
  return new Response(JSON.stringify(currentMessages), {
    headers: { 'Content-Type': 'application/json' }
  });
};

export const POST = async (request: Request) => {
  const mqttClient = connectMQTT();
  const { msg1, msg2 }: Messages = await request.json();

  if (
    msg1.length <= 16 &&
    msg2.length <= 16
  ) {
    currentMessages = { msg1, msg2 };

    mqttClient?.publish('lcd/messaging', JSON.stringify(currentMessages), {
      qos: 1,
      retain: true
    });

    return new Response(JSON.stringify({ status: 'ok' }), {
      headers: { 'Content-Type': 'application/json' }
    });
  } else {
    return new Response(JSON.stringify({ error: 'Invalid color values' }), {
      status: 400,
      headers: { 'Content-Type': 'application/json' }
    });
  }
};
```

# Додаток Г

## Фронтенд частина (ArduinoUno+ESP8266+LCD1602A)

```
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
'@/components/ui/card'
import { CarouselItem } from '@/components/ui/carousel'
import { Input } from '@/components/ui/input'
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
'@/components/ui/select'
import { useEffect, useState } from 'react'
import { Messages } from '../api/lcd/route'
import { Button } from '@/components/ui/button'

interface SensorData {
  temperature: number
  humidity: number
}

interface RGBColor {
  r: number
  g: number
  b: number
}

const LCD = () => {
  const [row1Type, setRow1Type] = useState('temperature')
  const [row2Type, setRow2Type] = useState('humidity')
  const [msg1, setMsg1] = useState('')
  const [msg2, setMsg2] = useState('')
  const [sensorData, setSensorData] = useState<SensorData | null>(null)
  const [color, setColor] = useState<RGBColor | null>(null)
  const [LCD, setLCD] = useState<Messages | null>(null)

  useEffect(() => {
    const fetchData = async () => {
      try {
        const sensorRes = await fetch('/api/dht')
        const colorRes = await fetch('/api/setColor')
        const lcdRes = await fetch('/api/lcd')

        const sensorJson = await sensorRes.json()
        const colorJson = await colorRes.json()
        const lcdJson = await lcdRes.json()

        // console.log('Sensor data:', sensorJson)
        // console.log('Color data:', colorJson)
        // console.log('LCD data:', lcdJson)

        setLCD(lcdJson)
        setSensorData(sensorJson)
        setColor(colorJson)
        setMsg1(lcdJson.msg1)
        setMsg2(lcdJson.msg2)
      } catch (error) {
        console.error('Error fetching sensor/color data:', error)
      }
    }

    fetchData()
  }, [])

  const getDisplayValue = (message: string | undefined) => {
```

```
    return message?.slice(0, 16).padEnd(16, ' ')
  }

  const handleSelectChange = (row: number, type: string) => {
    if (row === 1) {
      setRow1Type(type)
      if (type === 'temperature') {
        setMsg1(`${sensorData?.temperature} °C`)
      } else if (type === 'humidity') {
        setMsg1(`${sensorData?.humidity} %`)
      } else if (type === 'color') {
        setMsg1(`RGB(${color?.r}, ${color?.g}, ${color?.b})`)
      } else {
        setMsg1('')
      }
    } else if (row === 2) {
      setRow2Type(type)
      if (type === 'temperature') {
        setMsg2(`${sensorData?.temperature} °C`)
      } else if (type === 'humidity') {
        setMsg2(`${sensorData?.humidity} %`)
      } else if (type === 'color') {
        setMsg2(`RGB(${color?.r}, ${color?.g}, ${color?.b})`)
      } else {
        setMsg2('')
      }
    }
  }

  const handleSendData = async () => {
    await fetch('/api/lcd', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ msg1, msg2 }),
    })
  }


  return (
    <CarouselItem key={"temperature"} className="content-center">
      <div className="p-1">
        <Card>
          <CardHeader>
            <CardTitle>LCD</CardTitle>
            <CardDescription>Інформація на lcd екрані</CardDescription>
          </CardHeader>
          <CardContent className="flex flex-col gap-12 items-center justify-
center aspect-square p-6">
            <div className="space-y-3 w-1/2">
              <h2 className='text-3xl font-bold'>Рядок 1</h2>
              <Select onValueChange={(e) => handleSelectChange(1,
e.toString())}>
                <SelectTrigger className='w-full'>
                  <SelectValue placeholder="Select display type" />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="temperature">Температура</SelectItem>
                  <SelectItem value="humidity">Вологість</SelectItem>
                  <SelectItem value="color">Колір</SelectItem>
                  <SelectItem value="custom">Власне повідомлення</SelectItem>
                </SelectContent>
              </Select>
              {row1Type === 'custom' && (
                <Input
```

```
                  value={msg1}
                  onChange={(e) => setMsg1(e.target.value)}
                  maxLength={16}
                  placeholder="Custom Row 1"
                  className='w-full'
                />
              )}
            </div>

            <div className="space-y-3 w-1/2">
              <h2 className='text-3xl font-bold'>Рядок 2</h2>
              <Select onValueChange={(e) => handleSelectChange(2,
e.toString())}>
                <SelectTrigger className='w-full'>
                  <SelectValue placeholder="Select display type" />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="temperature">Температура</SelectItem>
                  <SelectItem value="humidity">Вологість</SelectItem>
                  <SelectItem value="color">Колір</SelectItem>
                  <SelectItem value="custom">Власне повідомлення</SelectItem>
                </SelectContent>
              </Select>
              {row2Type === 'custom' && (
                <Input
                  value={msg2}
                  onChange={(e) => setMsg2(e.target.value)}
                  maxLength={16}
                  placeholder="Custom Row 2"
                />
              )}
            </div>

            <div className="mt-4 bg-cyan-950 text-cyan-200 font-mono rounded p-4
tracking-widest shadow-inner text-4xl min-w-1/2 whitespace-pre">
              <div>{getDisplayValue(msg1)}</div>
              <div>{getDisplayValue(msg2)}</div>
            </div>

            <Button className="w-1/2 mt-2" onClick={handleSendData}>
              Застосувати
            </Button>
          </CardContent>
        </Card>
      </div>
    </CarouselItem>
  )
}

export default LCD
```

# Додаток Д

## Прошивка NodeMCU (NodeMCU+DHT-11)

```
#include "DHT.h"
#include <ESP8266WiFi.h>
#include <PubSubClient.h> // For MQTT communication

#define DPIN 4    // Pin to connect DHT sensor (GPIO number) D2
#define DTYPE DHT11  // Define DHT 11 or DHT22 sensor type

// WiFi credentials
const char* ssid = "RedmiNote8Pro";
const char* password = "11112222";

// MQTT settings
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;
String clientId = "ESP8266Client-";
const char* mqtt_topic = "dht11/data"; // Updated topic

DHT dht(DPIN, DTYPE);
WiFiClient espClient;
PubSubClient client(espClient);

unsigned long lastMsg = 0;
const long interval = 10000;

void setup_wifi() {
 delay(10);
 WiFi.begin(ssid, password);

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

 Serial.println("WiFi connected");
}

void reconnect() {
 while (!client.connected()) {
  Serial.print("Attempting MQTT connection...");
  clientId += String(random(0xffff), HEX);

  if (client.connect(clientId.c_str())) {
   Serial.println("connected");
  } else {
   Serial.print("failed, rc=");
   Serial.print(client.state());
   Serial.println(" try again in 5 seconds");
   delay(5000);
  }
 }
}

void setup() {
 Serial.begin(9600);
 randomSeed(micros());
 setup_wifi();
 client.setServer(mqtt_server, mqtt_port);
 dht.begin();
```

```
 Serial.println("DHT11 MQTT client started");
}

void loop() {
 if (!client.connected()) {
  reconnect();
 }
 client.loop();

 unsigned long now = millis();
 if (now - lastMsg > interval) {
  lastMsg = now;

  float tempC = dht.readTemperature(false); // Celsius
  float humidity = dht.readHumidity();

  if (isnan(tempC) || isnan(humidity)) {
   Serial.println("Failed to read from DHT sensor!");
   return;
  }

  Serial.print("Temp: ");
  Serial.print(tempC);
  Serial.print(" C, Hum: ");
  Serial.print(humidity);
  Serial.println(" %");

  // Format as JSON
  char jsonPayload[64];
  snprintf(jsonPayload, sizeof(jsonPayload),
       "{\"temperature\":%.2f,\"humidity\":%.2f}", tempC, humidity);

  client.publish(mqtt_topic, jsonPayload, true);
  Serial.print("JSON published: ");
  Serial.println(jsonPayload);
 }
}
```

# Додаток Е

## Бекенд частина (NodeMCU+DHT-11)

```
// app/api/sensor/route.ts (App Router) or pages/api/sensor.ts (Pages Router)
import { NextResponse } from 'next/server'
import { connectMQTT } from '@/lib/mqttClient'

interface SensorData {
 temperature: number
 humidity: number
}

let latestSensorData: SensorData = { temperature: 0, humidity: 0 }
let client = connectMQTT() // Ensure MQTT client is listening
client?.subscribe('dht11/data')

client?.on('message', (topic, message) => {
 if (topic === 'dht11/data') {
  try {
   const data = JSON.parse(message.toString())
   if (
    typeof data.temperature === 'number' &&
    typeof data.humidity === 'number'
   ) {
    latestSensorData = data
    console.log('Sensor data received:', data)
   }
  } catch (err) {
   console.error('Invalid sensor data received:', err)
  }
 }
})

client?.on('error', (err) => {
 console.error('MQTT connection error:', err)
 client = null
})

export async function GET() {
 return NextResponse.json(latestSensorData)
}
```

# Додаток Є

## Фронтенд частина (NodeMCU+DHT-11)

```
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
"@/components/ui/card"
import { useEffect, useState } from "react"
import { CarouselItem } from "@/components/ui/carousel"

const Temperature: React.FC = () => {
  const [temperature, setTemperature] = useState<number | null>(null)
  const [humidity, setHumidity] = useState<number | null>(null)

  useEffect(() => {
    const fetchSensorData = async () => {
      const res = await fetch('/api/dht')
      const data = await res.json()
      setTemperature(data.temperature)
      setHumidity(data.humidity)
    }

    fetchSensorData()
    const interval = setInterval(fetchSensorData, 3000)
    return () => clearInterval(interval)
  }, [])

  return (
    <CarouselItem key={"temperature"} className="content-center">
      <div className="p-1">
        <Card>
          <CardHeader>
            <CardTitle>DHT11</CardTitle>
            <CardDescription>Дані про температуру та вологість</CardDescription>
          </CardHeader>
          <CardContent className="flex flex-col gap-12 items-center justify-
center aspect-square p-6">
            <p className=" font-extrabold text-6xl">
              {temperature !== null ? `${temperature.toFixed(1)}°C` : '...'}
            </p>
            <p className=" mt-2 font-bold text-blue-600 text-4xl">
              💧 {humidity !== null ? `${humidity.toFixed(1)}%` : '...'}
            </p>
          </CardContent>
        </Card>
      </div>
    </CarouselItem>
  )
}

export default Temperature
```

# Додаток Ж

## Прошивка NodeMCU (NodeMCU+KY-016)

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h> // For MQTT communication
#include <ArduinoJson.h>

// WiFi credentials
const char* ssid = "RedmiNote8Pro";
const char* password = "11112222";

// MQTT settings
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;
String mqtt_client_id = "ESP8266Client-";
const char* mqtt_topic = "rgb_controller/color";

int redPin = 4;
int greenPin = 0;
int bluePin = 2;

WiFiClient espClient;
PubSubClient client(espClient);

long lastReconnectAttempt = 0;

void setup_wifi() {
 delay(10);
 WiFi.begin(ssid, password);

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

 Serial.println("WiFi connected");
}

// Callback when MQTT message is received
void callback(char* topic, byte* payload, unsigned int length) {

 // Convert payload to string and forward to Arduino
 String message;
 for (int i = 0; i < length; i++) {
  message += (char)payload[i];
 }

 Serial.println(message);

 StaticJsonDocument<128> doc;
  DeserializationError error = deserializeJson(doc, message);

  if (!error) {
   int r = doc["r"];
   int g = doc["g"];
   int b = doc["b"];

   // Apply the color
   analogWrite(redPin, r);
   analogWrite(greenPin, g);
   analogWrite(bluePin, b);
```

```
  }
}

// Reconnect to MQTT broker
boolean reconnect() {
 String clientId = mqtt_client_id + String(ESP.getChipId());

 if (client.connect(clientId.c_str())) {
  // Subscribe to topic
  client.subscribe(mqtt_topic);
 }
 return client.connected();
}

void setup() {
 Serial.begin(9600);
 randomSeed(micros());
 setup_wifi();
 client.setServer(mqtt_server, mqtt_port);
 client.setCallback(callback);
 lastReconnectAttempt = 0;

 pinMode(redPin, OUTPUT);
 pinMode(greenPin, OUTPUT);
 pinMode(bluePin, OUTPUT);

 // Default color - red
 analogWrite(redPin, 255);
 analogWrite(greenPin, 0);
 analogWrite(bluePin, 0);
}

void loop() {
 if (!client.connected()) {
  long now = millis();
  if (now - lastReconnectAttempt > 500) {
   lastReconnectAttempt = now;
   // Attempt to reconnect
   if (reconnect()) {
    lastReconnectAttempt = 0;
   }
  }
 } else {
  // Client connected - let the library handle incoming messages
  client.loop();
 }

}
```

# Додаток М

## Бекенд частина (NodeMCU+KY-016)

```
import { connectMQTT } from '@/lib/mqttClient'

interface Color {
 r: number;
 g: number;
 b: number;
}

let currentColor: Color = { r: 255, g: 0, b: 0 }; // Default color

// For app router version (API Routes)
// Import in Next.js app router format
export const GET = async () => {
 return new Response(JSON.stringify(currentColor), {
  headers: { 'Content-Type': 'application/json' }
 });
};

export const POST = async (request: Request) => {
 const mqttClient = connectMQTT();
 const { r, g, b }: Color = await request.json();

 if (
  typeof r === 'number' &&
  typeof g === 'number' &&
  typeof b === 'number' &&
  r >= 0 && r <= 255 &&
  g >= 0 && g <= 255 &&
  b >= 0 && b <= 255
 ) {
  currentColor = { r, g, b };

  mqttClient?.publish('rgb_controller/color', JSON.stringify(currentColor), {
   qos: 1,
   retain: true
  });

  return new Response(JSON.stringify({ status: 'ok' }), {
   headers: { 'Content-Type': 'application/json' }
  });
 } else {
  return new Response(JSON.stringify({ error: 'Invalid color values' }), {
   status: 400,
   headers: { 'Content-Type': 'application/json' }
  });
 }
};
```

# Додаток Н

## Фронтенд частина (NodeMCU+DHT-11)

```
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
"@/components/ui/card"
import { Button } from "@/components/ui/button"
import { Slider } from "@/components/ui/slider"
import { useEffect, useState } from "react"
import { CarouselItem } from "@/components/ui/carousel"

const ColorPicker: React.FC = () => {
  const [color, setColor] = useState<string>('#ff0000')
  const [r, setR] = useState(255)
  const [g, setG] = useState(0)
  const [b, setB] = useState(0)
  const [mounted, setMounted] = useState(false)

  useEffect(() => {
    const fetchColor = async () => {
      const response = await fetch('/api/setColor')
      const updatedColor = await response.json()

      setR(updatedColor.r)
      setG(updatedColor.g)
      setB(updatedColor.b)
      setColor(`rgb(${updatedColor.r}, ${updatedColor.g}, ${updatedColor.b})`)
    }

    fetchColor()
    setMounted(true)
  }, [])

  const handleSliderChange = (value: number[], colorType: 'r' | 'g' | 'b') => {
    const newValue = value[0]
    if (colorType === 'r') setR(newValue)
    if (colorType === 'g') setG(newValue)
    if (colorType === 'b') setB(newValue)

    setColor(`rgb(${colorType === 'r' ? newValue : r}, ${colorType === 'g' ?
newValue : g}, ${colorType === 'b' ? newValue : b})`)
  }

  const handlePresetClick = (r: number, g: number, b: number) => {
    setR(r)
    setG(g)
    setB(b)
    setColor(`rgb(${r}, ${g}, ${b})`)
  }

  const handleSendColor = async () => {
    await fetch('/api/setColor', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ r, g, b }),
    })
  }

  if (!mounted) return null

  return (
    <CarouselItem key={"rgb"} className="content-center">
      <div className="p-1">
```

```
        <Card>
          <CardHeader>
            <CardTitle>RGB LED</CardTitle>
            <CardDescription>Керуй підсвідкою rgb світлодіода.</CardDescription>
          </CardHeader>
          <CardContent className="flex flex-col gap-12 items-center justify-
center aspect-square p-6">
            <div
              className="w-24 h-24 rounded-full border shadow"
              style={{ backgroundColor: color }}
            />
            <p className="text-sm text-muted-foreground">RGB: {r}, {g}, {b}</p>

            <div className="w-full space-y-6" onTouchStartCapture={(e) =>
e.stopPropagation()}>
              <Slider
                value={[r]}
                max={255}
                step={1}
                onValueChange={(val) => handleSliderChange(val, 'r')}
              />

              <Slider
                value={[g]}
                max={255}
                step={1}
                onValueChange={(val) => handleSliderChange(val, 'g')}
              />
              <Slider
                value={[b]}
                max={255}
                step={1}
                onValueChange={(val) => handleSliderChange(val, 'b')}
              />
            </div>


            <div className="grid grid-cols-6 gap-8">
              {[
                [255, 0, 0],
                [0, 255, 0],
                [0, 0, 255],
                [255, 255, 0],
                [0, 255, 255],
                [255, 0, 255],
              ].map(([r, g, b], i) => (
                <button
                  key={i}
                  className="w-6 h-6 rounded-full border"
                  style={{ backgroundColor: `rgb(${r}, ${g}, ${b})` }}
                  onClick={() => handlePresetClick(r, g, b)}
                />
              ))}
            </div>

            <Button className="w-full mt-2" onClick={handleSendColor}>
              Застосувати
            </Button>
          </CardContent>
        </Card>
      </div>
    </CarouselItem>
  )
}
```

```
export default ColorPicker
```