МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему Віддалений моніторинг і управління програмними системами
Remote monitoring and management of software systems

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія.
(шифр і назва напряму підготовки, спеціальності)
Тіщенко Владислав Ігорович
(прізвище, ім'я, по-батькові)

Керівник к.ф.- м.н., доц. Крапівний Ю. М.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент к.т.н., доц. Волощук Л.А.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2022 р.
Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___
протокол № __ від «_»__ ____ 2022 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ Надія КАЗАКОВА
(підпис) (ім'я, прізвище)

Одеса – 2022

АНОТАЦІЯ

Дипломна робота присвячена проектуванню та розробці програмного-апаратного проекту для забезпечення віддаленого моніторингу і управління програмними системами.

Метою даної роботи є створення системи віддаленого моніторингу і управління з використанням програмно-апаратних засобів на базі мікроконтролерів ESP32 з підтримкою бездротової мережі Wi-Fi. Розробка програмного забезпечення для мікроконтролерів з підтримкою протоколів обміну MQTT, створення веб-інтерфейсу для моніторингу та управління клієнтами MQTT сервера (брокера).

Застосування розробки можливо в найширших сферах автоматизації, наприклад, в системах «розумного будинку», автоматизація тепличних господарств або промислових об'єктів. Розробка може використовуватися для реалізації «IoT» мережі «інтернету речей».

ABSTRACT

Thesis is devoted to the design and development of software and hardware project to provide remote monitoring and management of software systems.

The aim of this work is to create a system of remote monitoring and control using software and hardware based on ESP32 microcontrollers with support for Wi-Fi wireless network. Development of software for microcontrollers with support for MQTT exchange protocols, creation of a web interface for monitoring and managing MQTT server clients (brokers).

The application of development is possible in the widest areas of automation, for example, in the systems of "smart home", automation of greenhouses or industrial facilities. The development can be used to implement the IoT of the Internet of Things.

ЗМІСТ

Стор.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

IoT – Internet of things (інтернет речі)

ЖКГ – Житлово-комунальне господарство

Умовні позначення

Кбайт – кілобайт (1024 байт)

Мбіт – мегабіт (1000000 біт)

МГц – мегагерц (1000000 герц)

Терміни

QoS (англ. quality of service «якість обслуговування») — технологія надання різним класам трафіку різних пріоритетів в обслуговуванні, також цим терміном у галузі комп'ютерних мереж називають ймовірність того, що мережа зв'язку відповідає заданій угоді про трафік, або ж, у ряді випадків, неформальне позначення ймовірності проходження пакета між двома точками мережі.

Topic – набір символів з кодуванням UTF 8. Довжина теми не може перевищувати 32 767 символів.

ВСТУП

Актуальність проекту полягає в зростаючій необхідності віддаленого управління з різними технологічними об'єктами з зв'язку розвитком структури комп'ютерних мереж та з розвитком автоматизованих виробництв.

Наприклад: автоматизація процесу зняття показань із лічильників (економить час клієнта і гроші комунальних служб) і автоматичне відключення в аварійних ситуаціях (напр. протікання води, витік газу), менеджер комунальної служби може в реальному часі контролювати трафік (води, газу, світла, опалення) у клієнтів, автоматично отримувати звіти з даними лічильників по електронній пошті. Також є приклади з комплексної автоматизації теплиць в аграрній галузі.

У даний час зростає необхідність віддаленого управління з різними технологічними об'єктами з зв'язку розвитком структури комп'ютерних мереж та з розвитком автоматизованого виробництв і т.д. Широке поширення отримує поняття Internet of things (IoT) або інтернету речей, яке охоплює чи не всі, що будь-яким чином пов'язано з Інтернетом. У більш вузькому сенсі поняття може означати все безліч пристроїв, які використовують мережу Інтернет для передачі інформації і взаємодії один з одним. При цьому до IoT відносять будь-які типи пристроїв, від найпростіших сенсорів і датчиків до смартфонів і ноутбуків. Об'єднання таких пристроїв в автоматизовані системи дозволяє не тільки збирати величезну кількість інформації, але і виробляти обробку з метою вироблення конкретних дій, спрямованих на вирішення поставленого завдання.

З боку бізнесу поширення IoT пропонує можливості підвищення продуктивності, зниження витрат, економію часу і коштів, в багато дозволяючи по-новому поглянути на надані послуги і вироблені продукти, точніше визначити потреби споживачів і виявити нові можливості подальшого розвитку.

Широке поширення IoT отримав і в соціальній сфері. Сюди можна віднести використання IoT в медицині, логістиці, енергетиці, метеорології, в задачах екологічного моніторингу, спостереження за навколишнім середовищем, в сфері інтелектуалізації людського оточення.

Активно розвиваються актуальні напрямки реалізації «IoT»:

1. «Розумне місто» - міська інфраструктура і супутні муніципальні послуги, такі як освіта, охорона здоров'я, громадська безпека, ЖКГ, стануть більш пов'язаними і ефективними;
2. «Розумний будинок» - система буде розпізнавати конкретні ситуації, що відбуваються в будинку, і реагувати на них відповідним чином, що забезпечить мешканцям безпеку, комфорт і ресурсозбереження;
3. «Розумна енергетика» - буде забезпечена надійна і якісна передача електричної енергії від джерела до приймача в потрібний час і в необхідній кількості;
4. «Розумний транспорт» - переміщення пасажирів з однієї точки простору в іншу стане зручніше, швидше і безпечніше;
5. «Розумна медицина» - лікарі і пацієнти зможуть отримати віддалений доступ до дорогого медичного обладнання або до електронної історії хвороби в будь-якому місці, буде реалізована система віддаленого моніторингу здоров'я, автоматизована видача лікарських препаратів хворим і багато іншого.

Метою дипломної роботи є створення комплексної системи віддаленого моніторингу та управління, використовуючи програмно-апаратні засоби, що демонструють застосування трирівневої архітектури на прикладі роботи ,,міні-теплиці”, як одного з прикладів предметної області аграрної галузі.

Отже, в ході виконання дипломної роботи потрібно вирішити наступні завдання:

1) побудувати трирівневу архітектуру, яка складається з трьох об'єктів:
   - видавець;
   - брокер;
   - підписник.
2) створити веб-інтерфейс для взаємодії з контролером (видавцем).
3) скласти апаратну частину проекту (видавець).
4) на стороні MQTT сервера (брокера) необхідно прописати topic, які призначені для ідентифікації сутностей.
5) організувати протокол обміну повідомленнями відповідною реакцією на стороні клієнта.

1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ

1.1 Система розумної мікротеплиці Biopod

Біоакваріум Biopod Microhabitat по суті є теплицею [1] на рисунку 1.1, в якій за допомогою мобільного додатка можна підтримувати будь-який мікроклімат, необхідний для вирощування кімнатних рослин, а також для утримання акваріумних рибок і тераріумних тварин.

Мобільний додаток теплиці Biopod дозволить задати оптимальні параметри і підтримуватиме їх в автономному режимі, а вбудована HD камера допоможе з мобільного телефону стежити за поведінкою тварин і за процесом вирощування рослин.

Рисунок 1.1 — Мікротеплиця Biopod

Теплиця, в якій можна вирощувати будь-які рослини, починаючи з базиліку та закінчуючи екзотичними субтропічними квітами, підключена до хмарного сховища Biopod Cloud, що дозволяє ділитися своїми біологічними досягненнями з іншими любителями флори та фауни.

1.2 Система мікротеплиці InnoGro

Домашня ферма InnoGro [2] - американський стартап, який заснований на гідропоніки (безпідставного вирощування рослин) та аквапоніки – поєднання аквакультури (розведення риби). Домашній акваріум, у якому вирощуються овочеві культури забезпечить власника свіжими продуктами на цілий рік, який зображений на рисунку 1.2.

Рослинна ферма InnoGro встановлена на акваріум ємністю 45 літрів, а керується за допомогою спеціальної програми-компаньйона.

Щоб користувачеві програма сформувала необхідні налаштування для догляду, потрібно користувачеві вибрати бажані рослини та рибу з запропонованої бази даних. Ферма InnoGro працює практично автономно, завдяки інтегрованим датчикам та автоматизованому обладнанню. Завдяки бездротовоному зв'язку Wi-Fi, дані надсилаються на смартфон. Для забезпечення необхідного освітлення у конструкції передбачена люмінесцентна лампа. Користувачеві ферми не доведеться чистити акваріум, тому що відходи від риб використовуються рослинами як добрива, при цьому відбувається очищення води від забруднень.

Рисунок 1.2 — Мікротеплиця InnoGro

1.3 Екосистема Grove

Американська розробка штучної екосистеми Grove [3] представлена на краудфандинговому майданчику Kickstarter, яка зображена на рисунку 1.3, в якій співіснують риби та рослини, дозволяючи цілий рік вирощувати продукти багаті на поживні речовини, завдяки високотехнологічному сільськогосподарському методу, такий як аквапоніка. Принцип цього методу полягає в тому, що відходи життєдіяльності риб для рослин представляють їжу. При використанні цього методу не потреби удобрювати рослини та чистити акваріум. Мобільний додаток Grove OS дозволить управляти та контролювати невеликий город та акваріум, крім того в ньому міститься вся необхідна довідкова інформація. В екосистемі Grove використовується датчики для визначення рівня води, температури та вологості повітря, які створюють ідеальні умови для рослин та риб.

Рисунок 1.3 — Система Grove

За умови придбання системи, клієнт отримує все необхідне, для того щоб негайно приступити до роботи. Завдяки додатку Grove OS через всього місяць-півтора можна збирати свіжі органічні продукти, при цьому Grove дозволяє досягти більшої врожайності, ніж у звичайному саду.

1.4 Переваги та недоліки системи

Отже, провівши дослідження та аналіз існуючих систем та порівнявши характеристики системи між собою, які зображені в таблиці 1.1, можна виділити такі переваги своєї система над аналогічними системами, такими як:

1) можливість вимірювання температури;
2) можливість вимірювання рівню вологості повітря;
3) можливість вимірювання рівню освітлення.

Система також має недоліки порівняно з аналогами, такими як:

1) в системі не передбачено вимірювання рівню кисню та вуглецю;
2) система не має додаткових модулів.

Але не дивлячись на недоліки, в системі застосовується порівняно з аналогічними системами трирівнева архітектура та відносно новий протокол MQTT, які дають вагомі переваги над іншими системами такими як: низька потреба, низьке споживання трафіку, відсутність затримок у передачі даних та робота в умовах нестабільного зв'язку на лінії передачі даних.

Таблиця 1.1 — Порівняльна характеристика існуючих систем

Система / Можливості	Biopod	InnoGro	Grove
Вимірювання температури	-	+	+
Рівень вологості повітря	+	-	-
Рівень освітлення	+	+	+
Рівень кисню та вуглецю	+	-	-
Додаткові модулі	-	-	акваріум
Ціна	500$	380$	2700$

2 ПОСТАНОВКА ЗАВДАННЯ

Мета дипломної роботи є створення системи віддаленого моніторингу і управління з використанням програмно-апаратних засобів на базі мікроконтролерів ESP32 з підтримкою бездротової мережі Wi-Fi. Розробка програмного забезпечення для мікроконтролерів з підтримкою протоколів обміну MQTT, створення веб-інтерфейсу для моніторингу та управління клієнтами MQTT сервера (брокера).

Для досягнення поставленої мети потрібно виконати наступні завдання:

1) побудувати трирівневу архітектуру, яка складається з трьох об'єктів:
   - видавець;
   - брокер;
   - підписник.
2) створити веб-інтерфейс для взаємодії з контролером (видавцем).
3) скласти апаратну частину проекту;
4) на стороні MQTT сервера (брокера) необхідно прописати topic, які призначені для ідентифікації сутностей;
5) організувати протокол обміну повідомленнями відповідною реакцією на стороні клієнта.

3 ОПИС ВИКОРИСТОВУВАНИХ КОМПОНЕНТІВ ТА ПРОТОКОЛІВ

3.1 Мікроконтролер ESP32

3.1.1 Технічні характеристики ESP32

Для створення пристроїв Інтернет речей за основу було обрано апаратну платформу, а саме: мікроконтролер ESP32, який зображений на рисунку 3.1, тому що ESP32 є серією недорогих мікроконтролерів з інтегрованими модулями Wi-fi та Bluetooth порівняно з мікроконтролер ESP8266 він має значний приріст у продуктивності - його обчислювальна потужність зросла вчетверо. У ESP32 є два ядра, кожне з яких працює на частоті 160 МГц (ESP8266 має 1 ядро, що працює на частоті 80 МГц). Контролер містить 520 Кбайт оперативної пам'яті та 448 Кбайт flash-пам'яті. Підтримує не тільки Wi-Fi (802.11n з максимальною швидкістю 150 Мбіт в секунду, але і Bluetooth 4.2 BR/EDR та Low Energy) [4].

Рисунку 3.1 — Мікроконтролер ESP32

Основною перевагою ESP32 над ESP8266 є велика кількість каналів виводів та вони багатофункціональні. Для зручної роботи з мікроконтролером ESP32 було випущено модуль WROOM-32 (рис.3.2).

Рисунку 3.2 — Модуль WROOM-32

3.2 Програмні засоби розробки

3.2.1 Платформа та мова програмування ARDUINO

Мова програмування Arduino основана на C/C++ з фреймворком Wiring, вона має деякі відмінності в частині написання коду, який компілюється і збирається за допомогою бібліотеки avr-libc [5]. Ця бібліотека полегшує написання працюючої програми та дозволяє використовувати будь-які її функції та об'єкти.

Програми, які пишуть програмісти на мові Arduino, називаються начерки або скетчі і зберігаються у файлах із розширенням *.ino. Ці файли перед компіляцією обробляються препроцесор Arduino. Також існує можливість створювати та підключати до проекту стандартні файли C++.

Програміст повинен написати дві обов'язкові для Arduino функції setup() та loop(). Перша викликається одноразово при старті, друга виконується у нескінченному циклі.

Arduino - це відкрита програмована апаратна платформа для роботи з різними фізичними об'єктами і є простою платою з мікроконтролером, а також спеціальне середовище розробки для написання програмного забезпечення мікроконтролера.

Платформа Arduino має ряд переваг:

1) низька вартість;
2) крос-платформність;
3) просте та зрозуміле середовище програмування;
4) програмне забезпечення з можливістю розширення та відкритим вихідним текстом.

3.2.2 Фреймворк Angular

Angular представляє фреймворк від компанії Google для створення клієнтських програм. Насамперед він націлений розробку SPA-рішень (Single Page Application), тобто односторінкових додатків. У цьому плані Angular є спадкоємцем іншого фреймворку AngularJS. У той же час Angular це не нова версія AngularJS, а новий фреймворк [6].

Angular надає таку функціональність як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація і так далі.

Однією з ключових особливостей Angular є те, що він використовує мову програмування TypeScript, яка в свою чергу є основною мовою Angular.

TypeScript - мова програмування, представлена Microsoft в 2012 році і позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript. TypeScript є сумісним з JavaScript і компілюється в останній. Фактично, після компіляції програму на TypeScript можна виконувати у будь-якому сучасному браузері або використовувати разом із серверною платформою Node.js. Код на TypeScript виглядає майже так само, як і код на JS, і якщо у вас є досвід frontend-розробки, вивчити TypeScript досить просто. Особливо враховуючи, що ви можете писати JS-код прямо у TS-скриптах [7]. TypeScript має такі переваги [8]:

1. Сувора типізація;
2. TypeScript реалізує багато концепцій ООП, такі як успадкування, поліморфізм, інкапсуляція та модифікатори доступу, а також є класи, інтерфейси та абстрактні класи;
3. Потенціал мови дозволяє швидше та простіше писати складні комплексні рішення, які легше розвивати та тестувати надалі, ніж на стандартному JavaScript.

3.2.3 СУБД MariaDB

MariaDB – відгалуження реляційної СУБД MySQL. MariaDB повністю сумісна з програмами, що використовують MySQL, а перехід на цю СУБД виправданий тим, що MySQL вже не так активно розвивається [9].

MariaDB була вибрана, як за СУБД, а ніж MySQL, тому що вона має такі переваги, такі як [10]:

1. Продуктивність MariaDB у сховищі набагато краща завдяки стисненню даних в порівнянні з MySQL;
2. MariaDB більш прозора на відміну від MySQL;
3. MariaDB підтримує тимчасовий табличний простір і двійкове шифрування на відміну від MySQL;
4. Двійковий журнал можна стиснути в MariaDB на відміну від MySQL;
5. У MariaDB виправляються помилки частіше на відміну від MySQL;
6. Кілька потоків працюють паралельно, забезпечуючи кращу продуктивність бази даних у MariaDB, а в MySQL часто випускаються нові функції;
7. Зіставлення більше підтримується в MariaDB, а ніж в MySQL;
8. У MariaDB для тестування функцій доступні різні конфігурації, а ніж в MySQL;
9. У MariaDB з набором тестів для тестування краще, ніж в MySQL;
10. Складні запити можна вирішувати швидше у MariaDB, а ніж в MySQL;
11. Механізм зберігання пам’яті краще в MariaDB, а ніж в MySQL;
12. Перевірка привілеїв в MariaDB виконується швидше, а ніж в MySQL.

3.3 Трирівнева архітектура

Трирівнева архітектура представляє собою модель «Видання-підписка», яка відома як pub/sub є спосіб відділення клієнта, що передає повідомлення від іншого клієнта, що отримує повідомлення [6]. На відміну від традиційної моделі клієнт-сервер, клієнти не обізнані про будь-які фізичні ідентифікатори, на зразок IP-адреси або порту. MQTT – це архітектура pub/sub, але не черга повідомлень. Черги повідомлень за своєю природою зберігають повідомлення, а MQTT – ні. У MQTT, якщо ніхто не підписується (або не слухає) на тему, вона просто ігнорується та губиться.

Клієнт, який передає повідомлення, називається видавцем. Клієнт, який отримує повідомлення, називається передплатником. У центрі знаходиться брокер MQTT, який відповідає за з'єднання клієнтів і фільтрацію даних. Такі фільтри забезпечують:

1) фільтрацію за темами – за задумом, клієнти підписуються на теми та певні гілки тим, і не отримують даних більше, ніж хочуть. Кожне опубліковане повідомлення має містити тему і брокер несе відповідальність за повторну передачу цього повідомлення передплатникам або ігнорування його;

2) фільтрацію за вмістом – брокери мають можливість перевіряти та фільтрувати опубліковані дані. Таким чином, будь-які дані, які не зашифровані, можуть керуватися брокером перед тим, як зберегти їх або передати іншим клієнтам;

3) фільтрацію на кшталт – клієнт, прослуховуючи потік даних, куди він підписаний, може також використовувати власні фільтри.

Вхідні дані можуть аналізуватися і залежно від цього потік даних обробляється далі або ігнорується.

MQTT успішно відокремлює видавців від споживачів. Оскільки брокер є керівним органом між видавцями та споживачами, немає необхідності безпосередньо ідентифікувати видавця та споживача на основі фізичних даних (таких як IP-адреса). Це дуже корисно при розгортанні IoT, оскільки фізичний ідентифікатор може бути невідомим загальним. MQTT та інші моделі pub/sub також є тимчасово незалежними. Це означає, що повідомлення, опубліковане одним клієнтом, передплатником може прочитати та відповісти на нього будь-коли. Абонент може знаходитися в місці з дуже низьким енергоспоживанням / обмеженою смугою пропускання та відповісти на повідомлення за кілька хвилин або годин. Через відсутність фізичних та тимчасових відношень моделі pub/sub дуже добре підходять для підвищення продуктивності.

MQTT не залежить від формату даних. Корисне навантаження може містити будь-який тип даних, тому і видавці, і передплатники повинні розуміти та погоджувати формат даних. Можна надсилати текстові повідомлення, дані зображення, аудіодані, зашифровані дані, двійкові дані, об'єкти JSON або будь-яку іншу структуру в корисному навантаженні. Однак текстові та двійкові дані JSON є найбільш поширеними типами даних корисного навантаження.

Модель та топологія «Видання-підписка», яка зображена на рисунку 3.3, складається з трьох рівнів, таких як:

1. Рівень хмари. Хмарні технології – це розподілена обчислювальна система, яка надає користувачам можливість вирішувати свої проблеми, зберігати свої дані та їх обробляти, використовуючи через інтернет обчислювальні ресурси та хмарні сховища центрів обробки даних, що розташовані на великих підприємствах;
2. Рівень "Fog". Цей рівень є другим рівнем, який знаходиться між хмарою та останнім пристроєм ("edge device") та охоплює мережу, тобто на цьому рівні використовуються “fog” вузли. “Fog” вузли

можуть бути такі пристрої, такі як шлюз (“gateway”), комутатор (“switch”), засоби доступу (“access point”) тощо;

3. Рівень кінцевих користувачів. Перший рівень - найближчий рівень до кінцевого користувача ("end user"). На цьому рівні використовуються різні типи розумних пристроїв, наприклад, такі як: мобільні телефони, розумні камери відео спостереження, розумні системи сигналізації, розумні пральні машини тощо. Основна функція розумних пристроїв – це зборі даних про події, що відбуваються у фізичному середовищі, а також передачі накопичених даних у “fog” вузли, а далі передача на верхній рівень для їх зберігання та обробки.

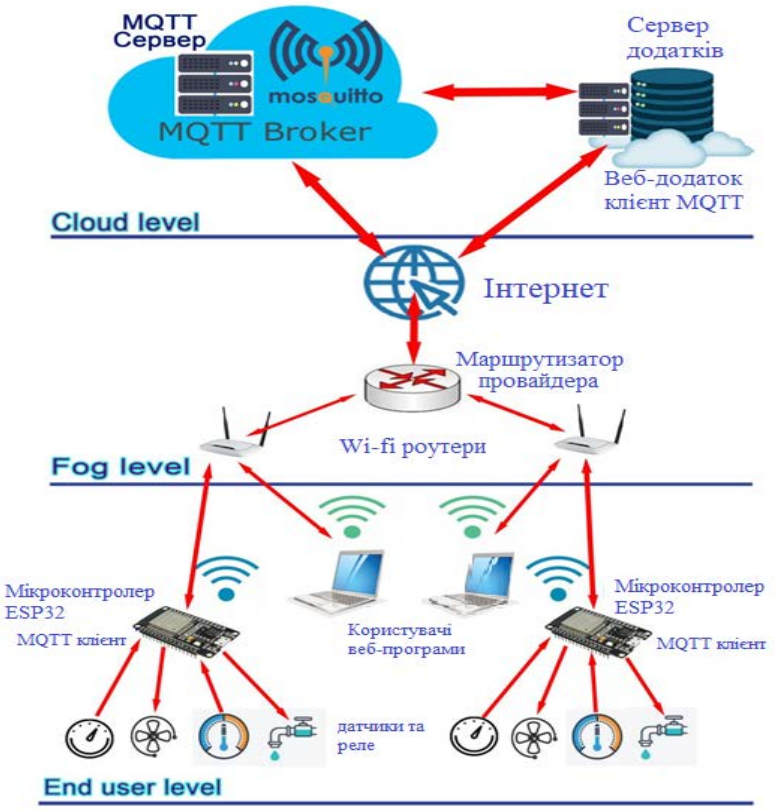

Рисунок 3.3 — Архітектурний опис принципу роботи трирівневої архітектури

3.4 Протокол MQTT

MQQT – це мережевий протокол, який є легким, відкритим та компактним для передачі даних на віддалені локації, де є потреба в невеликому розмірі пакету протоколу та є обмеження щодо пропускної спроможності каналу. Протокол є надійним, оскільки оснований на протоколе TCP/IP. Протокол MQQT працює на прикладному рівні поверх протоколу TCP/IP, який в свою чергу працює на мережевому рівні.

Основна сфера его застосування є доставка невеликих повідомлень, наприклад, показників сенсорів.

MQQT протокол має такі переваги [7]:

1. MQTT є асинхронним протоколом. Наприклад, вузол А повинен зв'язуватися з вузлом В. Асиметричний протокол між А і В вимагає, щоб протокол використовувала лише одна сторона (А), проте вся інформація, необхідна для повторного складання пакетів, повинна міститися в заголовку фрагментації, надісланому А. В асиметричних системах є один ведучий та один ведений;
2. Компактні повідомлення;
3. Робота в умовах нестабільного зв'язку на лінії передачі даних;
4. Підтримка кількох рівнів якості обслуговування (QoS);
5. Легка інтеграція нових пристроїв.

Принцип роботи MQQT протоколу [7].

Обмін повідомленнями в протоколі MQTT здійснюється між клієнтом (client), який може бути видавцем або підписником (publisher/subscriber) повідомлень.

Видавець відправляє дані на брокер MQTT, вказуючи в повідомленні певну тему, топік (topic). Підписники можуть отримувати різні дані від багатьох видавців залежно від підписки на відповідні топіки.

Схема простої взаємодії між підписником, видавцем та брокером (рис.3.4).

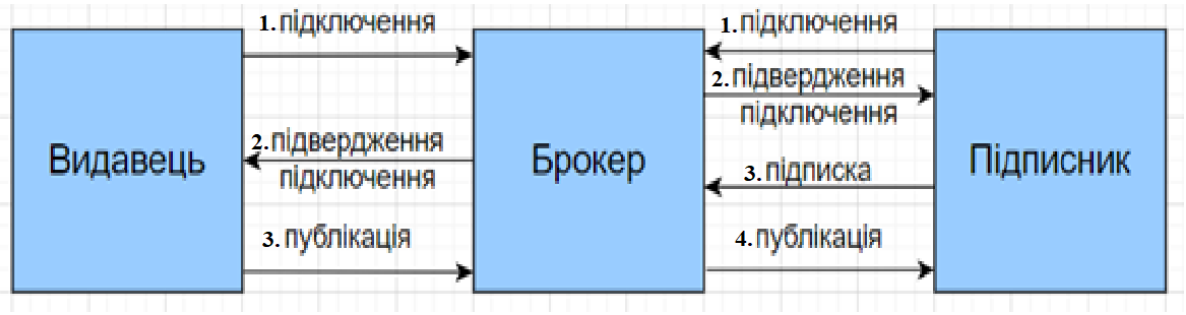

Рисунок 3.4 — Схема роботи протоколу MQTT

Пристрої MQTT використовують певні типи повідомлень для взаємодії з брокером, наведені нижче основні:

1) Connect – встановити з'єднання з брокером;
2) Disconnect – розірвати з’єднання з брокером;
3) Publish – опублікувати дані в топік на брокері;
4) Subscribe – підписатися на топік на брокері;
5) Unsubscribe - відписатися від топіка.

3.4.1 Структура повідомлень протоколу MQTT

MQTT повідомлення складається з кількох частин:

1. Фіксований заголовок (присутнє по всіх повідомленнях);
2. Змінний заголовок (є лише у певних повідомленнях);
3. Дані, «навантаження» (є лише у певних повідомленнях).

3.4.2 Фіксований заголовок протоколу MQTT

Фіксований заголовок складається з кількох частин зображений на рисунку 3.5:

1. Message Type – це тип повідомлення, наприклад: CONNECT, SUBSCRIBE, PUBLISH та інші;
2. Flags specific to each MQTT packet – ці 4 біти відведені під допоміжні прапори, наявність та стан яких залежить від типу повідомлення;
3. Remaining Length – це довжина поточного повідомлення (змінний заголовок + дані), може займати від 1 до 4 байти.

Рисунок 3.5 — Фіксований заголовок

Чотири старші біти першого байта фіксованого заголовка відведені під спеціальні прапори зображені на рисунку 3.6.

Рисунок 3.6 — Прапори

1. DUP – прапорець дубліката встановлюється, коли клієнт або MQTT брокер здійснює повторне відправлення пакета (використовується у типах PUBLISH, SUBSCRIBE, UNSUBSCRIBE, PUBREL). Якщо встановлено прапорець, змінний заголовок повинен містити Message ID (ідентифікатор повідомлення);

2. QoS – якість обслуговування (0,1,2);
3. RETAIN – при публікації даних із встановленим прапором retain, брокер збереже його. При наступній підписці на цей топік брокер негайно надішле повідомлення із цим прапором. Використовується лише у повідомленнях із типом PUBLISH.

3.4.3 Змінний заголовок протоколу MQTT

Змінний заголовок, який зображений на рисунку 8 міститься в деяких заголовках.

У ньому містяться такі дані:

1) Packet identifier – ідентифікатор пакета, присутній у всіх типах повідомлень, крім: CONNECT, CONNACK, PUBLISH (з QoS <1), PINGREQ, PINGRESP, DISCONNECT;
2) Protocol name – назва протоколу (тільки у повідомленнях типу CONNECT);
3) Protocol version – версія протоколу (тільки у повідомленнях типу CONNECT);
4) Connect flags – прапори, що вказують на поведінку клієнта при підключенні;

Bit	7	6	5	4	3	2	1	0
Byte 8	User name	Password	Will Retain	Will QoS		Will Flag	Clean Session	Reserved

Рисунок 3.7 — Змінний заголовок

Змінний заголовок складається з таких частин:

1) User name – за наявності цього прапора у «навантаженні» має бути вказано ім'я користувача (використовується для автентифікації клієнта);
2) Password – за наявності цього прапора в «навантаженні» має бути вказаний пароль (використовується для автентифікації клієнта);
3) Will Retain - при установці в 1, брокер зберігає у себе Will Message.
4) Will QoS – якість обслуговування для Will Message, при встановленому прапорі Will Flag, Will QoS та Will retain є обов'язковими;
5) Will Flag - при встановленому прапорі, після того, як клієнт відключиться від брокера без відправки команди DISCONNECT (у випадках непередбачуваного обриву зв'язку тощо), брокер сповістить про це всіх підключених клієнтів через так званий Will Message;
6) Clean Session – очистити сесію. При встановленому "0" брокер збереже сесію, всі підписки клієнта, а також передасть йому всі повідомлення з QoS1 і QoS2, які були отримані брокером під час відключення клієнта, при наступному підключенні. Відповідно при встановленій «1», при повторному підключенні клієнту необхідно буде заново підписуватися на топіки.

3.4.4 Дані, навантаження

Зміст та формат даних, що надсилаються в MQTT повідомленнях, визначаються у додатку. Розмір даних можна обчислити шляхом віднімання з Remaining Length довжини змінного заголовка.

3.5 Датчик температури та вологості

DHT11 - це цифровий датчик вологості та температури представлений на рисунку 9, що складається з термістора та ємнісного датчика вологості, який дозволяє калібрувати цифровий сигнал на виході. Також датчик містить у собі АЦП для перетворення аналогових значень вологості та температури. Датчик DHT11 не має високу швидкодію і точність, зате простий, недорогий і відмінно підходять для навчання і контролю вологості в приміщенні [8].

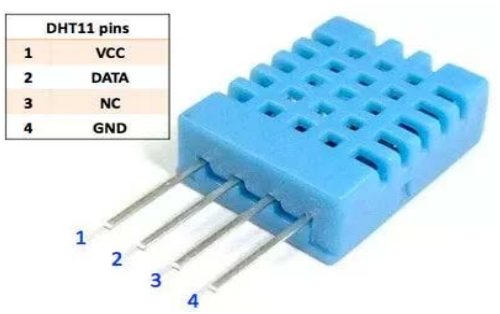

Рисунок 3.8 — Датчик температури та вологості DHT11

Датчик має 4 виводи стандарту 2,54 мм (рисунок 3.8):

1) VCC (живлення 3-5 В);
2) DATA (виведення даних);
3) не використовується;
4) GND (земля).

Технічні характеристики датчика DHT11:

1) живлення: DC 3,5 - 5,5 В;
2) визначення вологості 20-80% з точністю 5%;
3) визначення температури 0–50 °C із точністю 2 %;
4) частота опитування не більше 1 Гц (не більше 1 разу на 1 сек.).

3.6 Типи використовуваних реле

Одно канальний модуль реле JQC-3FF-S-Z, який зображений на рисунку 3.9 призначений для управління потужним струмом, як постійним, так і змінним. Крім самого реле модуль містить ще й оптоелектронну розв'язку з транзистором, які захищають виходи Arduino від стрибків напруги на котушці. Реле спрацьовує під час подачі на вхід низького рівня сигналу [9].

Рисунок 3.9 — 1-канальний модуль реле JQC-3FF-S-Z

Опис висновків модуля:

1) VCC – підключається до 5V для живлення модуля (+);
2) GND – підключається до "землі" (-);
3) IN – підключається, наприклад, до плати Arduino або мікроконтролера для управління замиканням та розмиканням виходів реле;
4) COM – для підключення контакту керованого ланцюга;
5) NO – нормально розімкнений вихід (Normally Open), замкнеться з виходом COM при подачі високого рівня на вхід IN;
6) NC – нормально замкнутий вихід (Normally Closed), розімкнеться з виходом COM при подачі високого рівня на вхід IN.

На модулі також є два світлодіоди:

1) червоний – індикація подачі живлення на модуль;
2) зелений – індикація замикання реле.

Модуль електромеханічного реле на 2 канали SRD-05-VDC-SL-C зображений на рисунку 11 дозволяє комутувати ланцюги як змінного, так і постійного струму до 10А. Але рекомендується комутувати ланцюги зі струмом до 7А. Модуль побудований на базі реле SRD-05-VDC-SL-C, чим і забезпечується його комутаційна здатність. Реле модуля здатне комутувати вихідні ланцюги з напругою до 250В змінного струму (AC) або до 30В постійного струму (DC) [10].

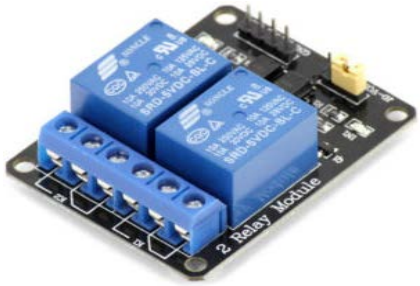

Рисунок 3.10 — 2-канальний модуль реле SRD-05-VDC-SL-C

Струм, що протікає в комутованих (вихідних) ланцюгах, при максимальній напрузі, не повинен перевищувати 10А. Вхідна (керівна) напруга 5В. Модулем можна керувати, підключивши його до Arduino, рівень логічного «1» висновків якого дорівнює 5В.

Вхідна напруга живлення 5В постійного струму подається на входи «Vcc» та «GND» модуля.

Додатково, на платі модуля є роз'єм із трьома висновками:

- "JD-VCC";
- "VCC";
- "GND";

3.7 LCD-екран та кнопки

Графічний дисплей 128x64 зображений на рисунку 3.11. На відміну від текстових дисплеїв дозволяє виводити також і картинки. Може бути підключений як за паралельним, так і за послідовним інтерфейсом [11].

Характеристики:

- напруга логічних рівнів 5В/3В
- робочий діапазон температур: –20 до +70 град.
- розміри плати: 93x70 мм
- розміри рамки екрану: 73x51 мм

Рисунок 3.11 — Графічний дисплей 128x64

Модуль тактової кнопки з підтягуючим резистором та контактним роз'ємом. Більш зручний, ніж просто кнопка для макетування, зважаючи на більший розмір кнопки і ковпачка, але при цьому зберігає місце на макетній платі. Модуль кнопки з ковпачком зображений на рисунку 3.12.

Характеристики:

- Діапазон робочих температур: -20ºС…+60ºС
- Опір контактів: 50 мОм
- Максимальна напруга: 100В
- Ліміт натискань: 10 000

Рисунок 3.12 — Модуль кнопки з ковпачком

Кнопка двоконтактна на 4 виведення, нормальний стан розімкнена, яка зображена на рисунку 3.13.

Характеристики:

- діапазон температур: -20 ºС... +60 ºС
- опір контактів: 30 мОм
- максимальна напруга: 250В (протягом 1 хвилини)
- ліміт натискань: 10 000
- ширина х глибина х висота: 6мм х 6мм х 5мм (зазначена загальна висота кнопки).

Рисунок 3.13 — Двоконтактна кнопка

3.8 Ефективність обраних програмних засобів системи

У цьому розділі були розглянуті основні протоколи, що використовуються в області IoT. На основі їх аналізу та порівняння було обрано протокол MQTT, оскільки він легкий та ефективний з точки зору пропускної спроможності та може працювати в умовах нестабільного зв'язку. Також було обрано програмні та апаратні засоби для реалізації майбутньої системи.

4 ОБГРУНТУВАННЯ ПРОЕКТНОГО РІШЕННЯ

Об'єктом дослідження є розробка програмно-апаратних вимог для систем «розумної теплиці», «розумного будинку», та реалізація обраного рішення, гібридної системи з трирівневою архітектурою.

Реалізація систем розумний будинок, розумна теплиця накладає на розробників цих систем ряд обмежень та вимог, пов'язаних не тільки з необхідністю забезпечення максимального ступеня надійності, швидкодії та безпеки, але й висувають ряд вимог до підходів, що використовуються, архітектурі, інструментарію:

1) можливість розвитку інженерних систем (масштабування та модернізацію);
2) значна кількість датчиків для збору оперативної інформації про стан об'єкта;
3) програмні та апаратні елементи системи не повинні бути прив'язані до одного виробника;
4) застосування типових пристроїв - контролерів, датчиків, систем відображення інформації та ін.

У зв'язку з цими вимогами всі інженерні системи в розумному будинку, розумній теплиці повинні бути інтегровані на базі єдиної інформаційної системи – автоматизованої системи управління об'єктом.

5 РОЗРОБКА ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ ВІДДАЛЕНОГО УПРАВЛІННЯ ТА МОНІТОРИНГУ ТЕПЛИЦІ

5.1 Розробка електричних принципових схем модулів для керування та моніторингу теплиць

Для розробки апаратної частин були розроблені електричні принципові схеми модулів для керування та моніторингу теплиці, які зображені на схемі 1 для першого клієнта та схемі 2 для другого клієнта.

Схема 1 складається з:

- Мікроконтролера ESP32 (1);
- Датчика температури та вологості DHT11 (2);
- 2-канальний модуль реле SRD-05-VDC-SL-C (3);
- Батарея живлення на 5 В (4);
- Кнопки S1, S2;
- Резисторів R1, R2, R3, R4, R5 на 220 Ом. Підключення цих резисторів необхідне забезпечення стабільної роботи пристроїв.

Схема 2 відрізняється від схеми 1, тим що на ній додатково підключено LSD-екран (5).

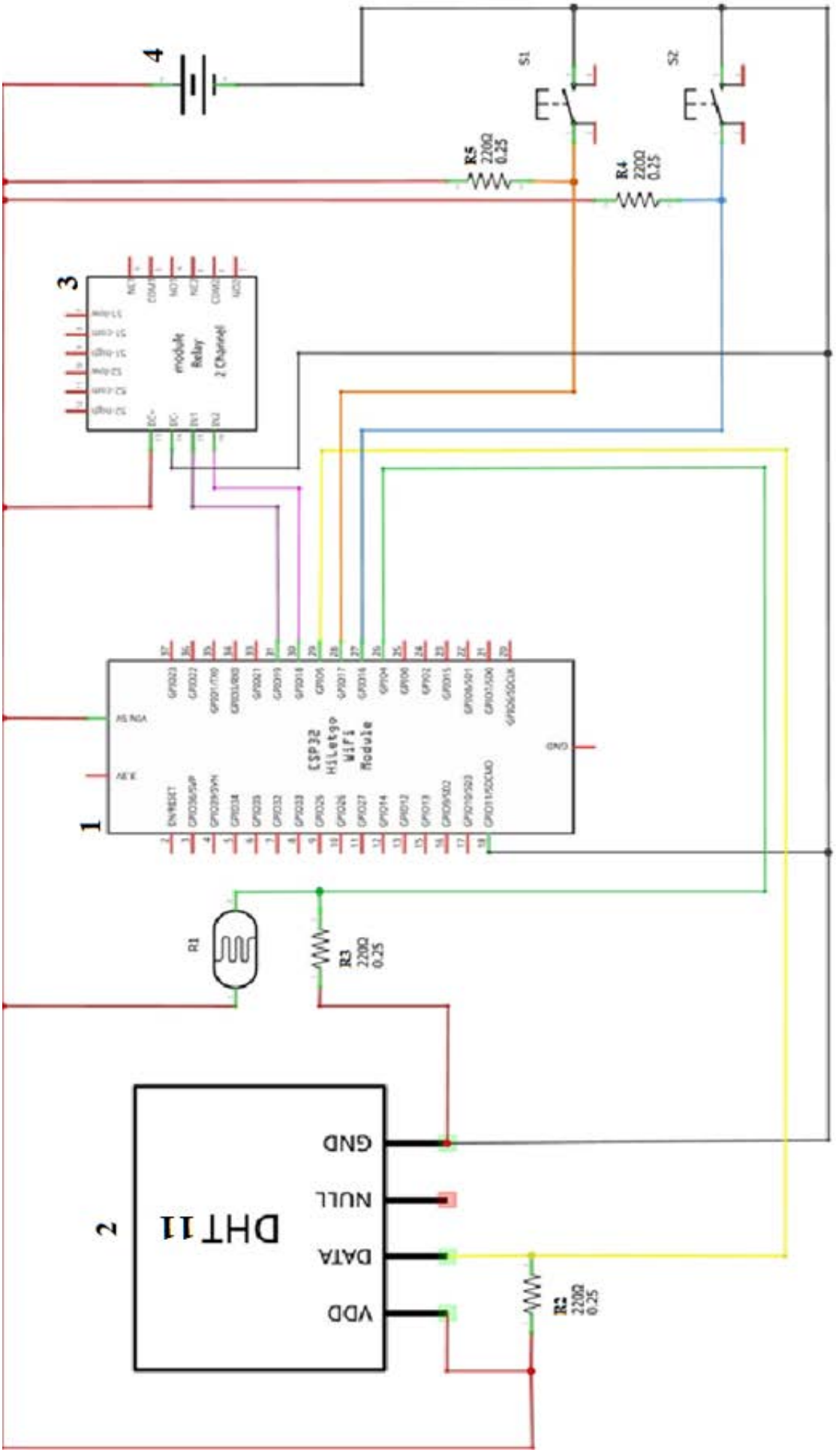

Схема 1. Електрична принципова схема модуля для керування та моніторингу теплиці для 1-го клієнта

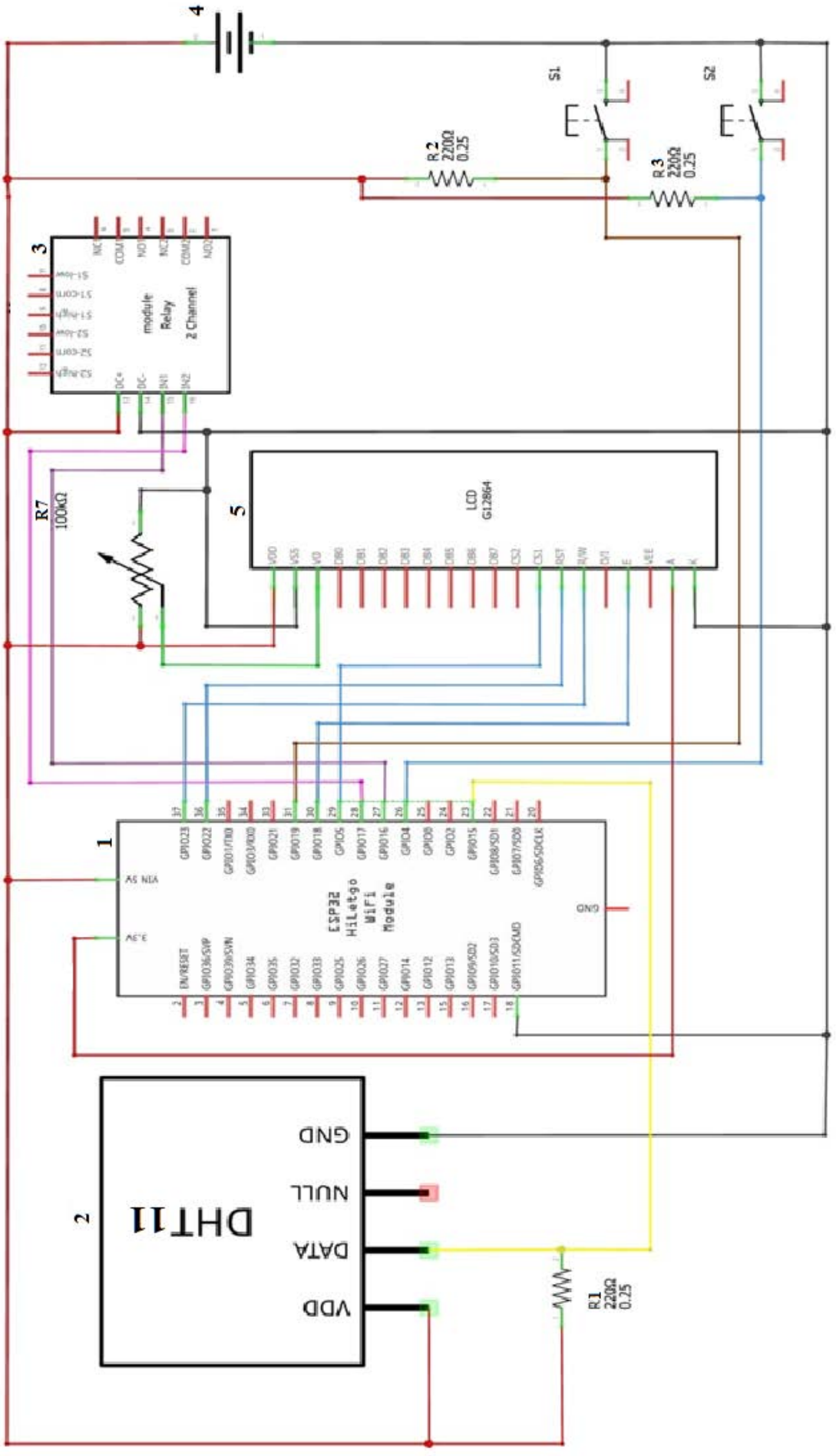

Схема 2. Електрична принципова схема модуля для керування та моніторингу теплиці для 2-го клієнта

5.2 Розробка архітектури

Після того як усі вимоги та умови до системи сформовані, приступаємо до створення архітектури.

На нижньому апаратному рівні вибираємо варіант топології провідного з'єднання "зірка": кожен датчик, перемикач, реле управління з'єднуються окремим дротом з контролером ESP32.

Контролери ESP32 зв'язуються (підключаються) до роутерів локальної мережі Wi-Fi для доступу в інтернет. Цей варіант підключення дає гнучкість та легку масштабованість системі, також він економічно вигідний через можливість використання домашніх (побутових роутерів).

З існуючих видів архітектури таких систем, децентралізована, централізована та гібридна, виберемо гібридну (сумісну).

У запрограмованому проекті існує два сервери перший - це MQTT брокер необхідний для обміну повідомленнями між клієнтами (контролерами), а другий – це сервер веб-додатків з додатком користувачів для здійснення віддаленого керування та моніторингу показань датчиків та стану пристроїв (реле).

При розробці системи віддаленого керування були виділені такі основні компоненти: контролери ESP32, підключені до них датчики та пристрої (реле), роутери WiFi, маршрутизатори, MQTT брокер (сервер), сервер з веб-додатком і БД.

Розглядаємо ці компоненти та описуємо алгоритми їх роботи:

1. Датчики температури та вологості, блоки реле управління підключаються до портів контролера ESP32, і передають чи одержують дані.

2. Контролери ESP32 підключені через Wi-Fi мережу до інтернету і підключаються до MQTT брокеру і обмінюються даними (у вигляді

топіків, що прописуються) переданими чи отриманими з датчиків і реле підключених до їх портів.

3. Сервер з додатком є клієнтом MQTT брокера обмінюється даними (у вигляді топіків, що прописуються) MQTT брокером, і зберігає необхідні дані в базі даних для побудови графіків для статистики.

5.2.1 Доставка повідомлень

Доставка повідомлень в системі здійснюється за допомогою топіків. Топік – адреса повідомлення. Топік складається з таких частин: корінь топіка та устройства. Однорівневий топік зображений на рисунку 5.1 посилається від підписника видавцю або навпаки і після цього прийде у відповідь топік зі значенням 0 або 1, залежно від того, в якому стані було реле вимкненому або включеному. 0 означає, що реле включилося, а 1 навпаки вимкнулося.

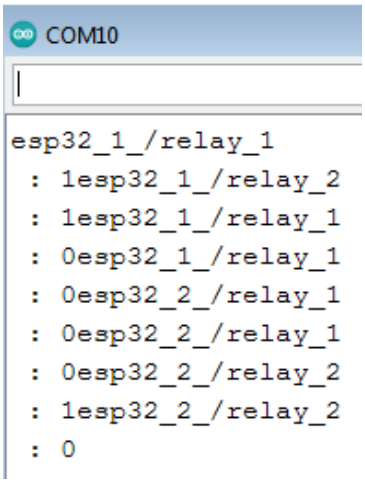

Рисунок 5.1 — Топіки на відключення або включення реле

Доставка повідомлень в системі здійснюється також багаторівневим топіком завдяки якому можливо отримати дані від датчика температуру та вологість, а також освітлення. На запит даних від видавця топіком в

результаті отримуємо дані згідно з надісланим топіком зображеного на рисунку 5.2.

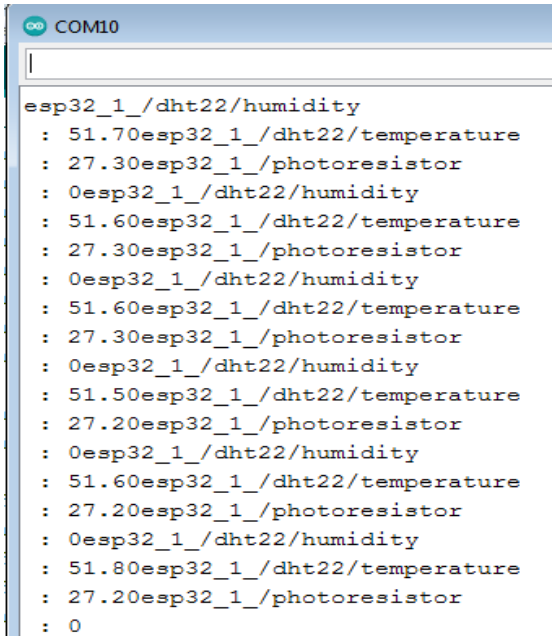

Рисунок 5.2 — Топік на моніторинг температури та вологості

Програмний код скетчів для 1-го клієнта представлено у додатку А, а для 2-го у додатку Б.

5.3 Інформаційне моделювання предметної області

Перший етап проектування полягає у створенні схеми БД, яка включає визначення сутностей та існуючих між ними зв'язків та атрибутів. Результатом даного проектування є схема БД, представлена у вигляді ER-діаграми (Entity-Relationship) розумної теплиці, на якій відображаються основні сутності ПрО, їх атрибути та зв'язки.

У цій роботі поставлено завдання побудови ER-діаграми для розумної теплиці. З аналізу ПрО, для функціонування інформаційної системи

необхідні такі сутності: користувач, пристрій, датчик, параметри датчика, підключений користувач.

Сутності та його властивості з описом обмежень, необхідні вирішення поставлених завдань, наведені у таблиці 5.3.1.

Таблиця 5.3.1 - Опис сутностей та їх властивостей

Властивість	Описання	Обмеження
Users		
Id	Ідентифікатор користувача	Первинний ключ, не порожнє
Name	Ім’я користувача	до 8 символів, не порожнє
Login	Пароль користувача	до 100 символів, не порожнє
Role	Роль в системі	до 8 символів, не порожнє
Password	Пароль користувача	до 8 символів, не порожнє
Devices		
Id	Ідентифікатор пристрою	Первинний ключ,не порожнє
Name	Назва пристрою (ESP 32)	до 100 символів, не порожнє
UserId	Ідентифікатор користувача	Зовнішній ключ для зв'язку з таблицею Users, не порожнє
DeviceParameters		
Id	Ідентифікатор параметра	Первинний ключ, не порожнє, унікальне значення
Type	Тип Наприклад: (Relay, Button, DHT22_temp, DHT22_hum).	до 100 символів, не порожнє
Topic	Повідомлення клієнта	до 100 символів, не порожнє
Name	Назва параметра	до 100 символів, не порожнє
SensorLogsId	Ідентифікатор датчика	Зовнішній ключ для зв'язку з таблицею SensorLogs.

Продовження таблиці 5.3.1

ConnectedUsers		
UserId	Ідентифікатор клієнту	не порожнє
Id	Ідентифікатор підключеного (до SignalR) користувача	Первинний ключ, не порожнє, унікальне значення
GroupeName	Назва групи SignalR	до 100 символів, не порожнє
ConnectionId	Ідентифікатор з’єднання виданий SignalR	не порожнє
SensorLogs		
Id	Ідентифікатор датчика	Первинний ключ, не порожнє, унікальне значення
DeviceId	Ідентифікатор пристрою	Зовнішній ключ для зв'язку з таблицею Devices, не порожнє
Value	Значення	не порожнє
Time	Час створення або оновлення	не порожнє
Type	Тип запису	до 100 символів, не порожнє

Наступним кроком є формалізація зв’язків між сутностями.

У даній ПрО можна виділити такі зв’язки:

1. Зв’язок між сутностями Users та ConnectedUsers формалізується за правилом "один-до-одного". Для формалізації даного зв'язку ідентифікатор UsersId був доданий до таблиці ConnectedUsers як зовнішній ключ.
2. Зв’язок між сутностями Users та Devices формалізується за правилом "один до багатьох". Для формалізації даного зв'язку ідентифікатор UsersId був доданий до таблиці Devices як зовнішній ключ.

3. Зв’язок між сутностями Devices та SensorLogs формалізується за правилом "один до багатьох". Для формалізації даного зв'язку ідентифікатор DevicesId був доданий до таблиці SensorLogs як зовнішній ключ.
4. Зв’язок між сутностями SensorLogs та DeviceParameters формалізується за правилом "один-до-одного". Для формалізації даного зв'язку ідентифікатор SensorLogsId був доданий до таблиці DeviceParameters як зовнішній ключ.

Після формалізації отримана ER-діаграма бази даних, представлена на рисунку 5.3.1.

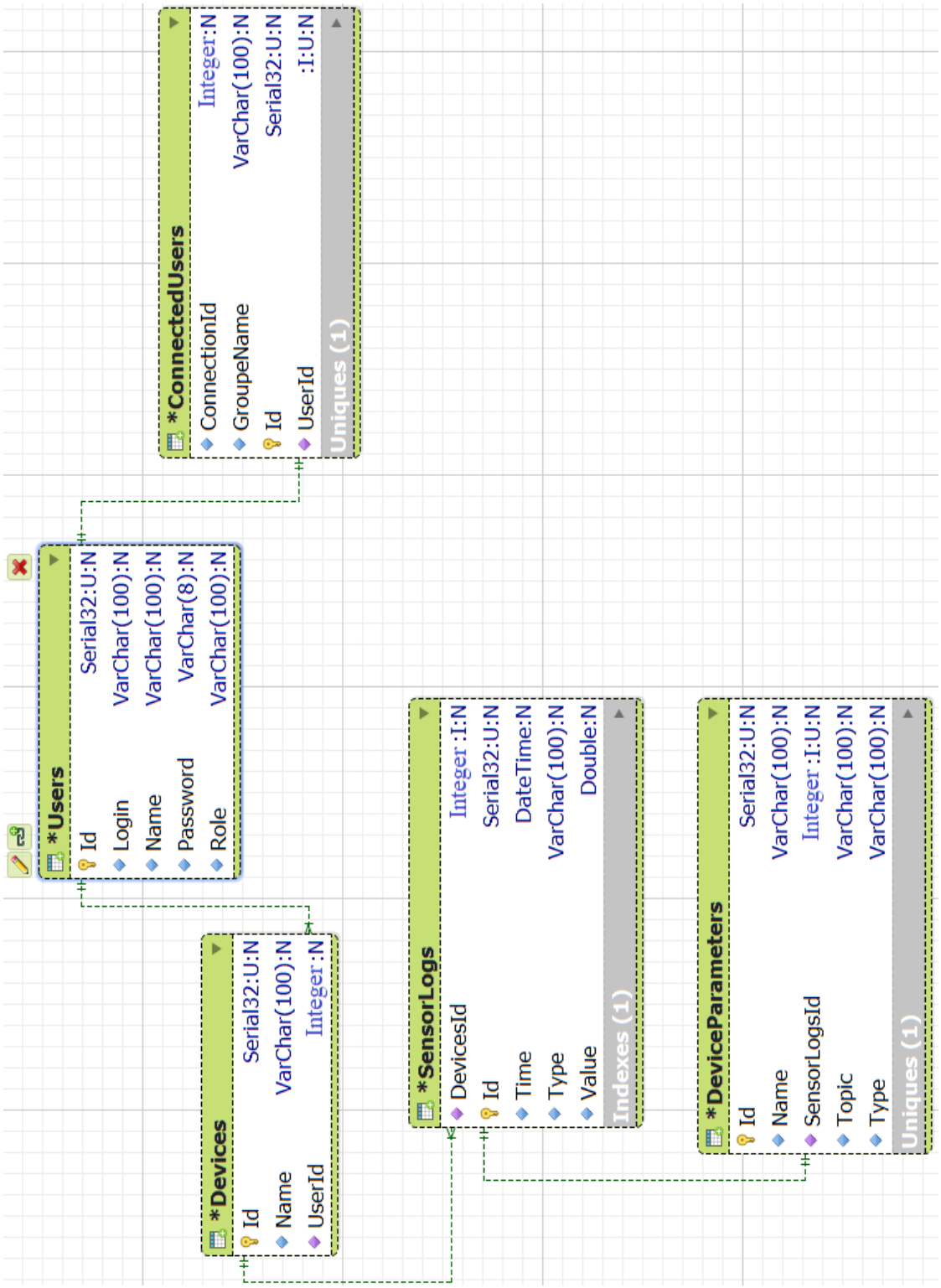

Рисунок 5.3.1 – ER-діаграма бази даних розумної теплиці

5.4 Запитання до бази даних для рішення поставлених завдань

Технологія Entity Framework пропонує відмовитись від написання чистих SQL запитів, використовуючи при цьому технологію LINQ to Objects. Це потужний інструмент, що дозволяє формувати прості та складні запити за допомогою коду. При цьому робота йде з об'єктами, а не стовпцями та рядками, що є величезною перевагою перед чистим SQL. Всі LINQ вирази в кінцевому підсумку перетворюються на запити, але це робиться прозоро для розробника. Розглянемо приклади таких LINQ виразів.

Для ідентифікації користувача використовується такий запит, який представлений у лістингу 5.1:

```
var user = _db.Users
    .AsNoTracking()
    .FirstOrDefault(x => x.Name == User.Identity.Name);
```

Лістинг 5.1 — Запит для ідентифікації користувача

Аналогічний SQL запит виглядав би так для ідентифікації користувача, який представлений у лістингу 5.2:

```
SELECT `u`.`UserId`, `u`.`Login`, `u`.`Name`, `u`.`Password`,
     `u`.`Role`
 FROM `Users` AS `u`
 WHERE `u`.`Name` = @__User_Identity_Name_0
```

Лістинг 5.2 — Аналогічний SQL запит для ідентифікації користувача

Для отримання параметру за ідентифікатором використовується такий запит, який представлений у лістингу 5.3:

```
var parameter = _db.DeviceParameters
    .FirstOrDefault(x => x.DeviceParameterId == parameterId);
```

Лістинг 5.3 — Запит для отримання параметру за ідентифікатором

Аналогічний SQL запит виглядав би так для отримання параметру за ідентифікатором, який представлений у лістингу 5.4:

```
SELECT `d`.`DeviceParameterId`, `d`.`DeviceId`, `d`.`Name`,
`d`.`Topic`, `d`.`Type`
      FROM `DeviceParameters` AS `d`
      WHERE `d`.`DeviceParameterId` = @__parameterId_0
      LIMIT 1
```

Лістинг 5.4 — Аналогічний SQL запит для отримання параметру за ідентифікатором

Для отримання запису за ідентифікатором використовується такий запит, який представлений у лістингу 5.5:

```
var sensorLog = _db.SensorLogs
    .FirstOrDefault(x => x.DeviceParameterId ==
parameter.DeviceParameterId);
```

Лістинг 5.5 — Запит за ідентифікатором

Аналогічний SQL запит виглядав би так для отримання запису за ідентифікатором, який представлений у лістингу 5.6:

```
SELECT `s`.`Id`, `s`.`DeviceParameterId`, `s`.`Time`,
`s`.`Type`, `s`.`Value`
      FROM `SensorLogs` AS `s`
      WHERE `s`.`DeviceParameterId` =
@__parameter_DeviceParameterId_0
      LIMIT 1
```

Лістинг 5.6 — Аналогічний SQL запит для отримання запису за ідентифікатором

Для отримання пристрою за ідентифікатором використовується такий запит, який представлений у лістингу 5.7:

```
var device = _db.Devices
    .FirstOrDefault(x => x.DeviceId == parameter.DeviceId);
```

Лістинг 5.7 — Запит для отримання пристрою за ідентифікатором

Аналогічний SQL запит виглядав би так для отримання пристрою за ідентифікатором, який представлений у лістингу 5.8:

```
SELECT `d`.`DeviceId`, `d`.`Name`, `d`.`UserId`
      FROM `Devices` AS `d`
      WHERE `d`.`DeviceId` = @__parameter_DeviceId_0
      LIMIT 1
```

Лістинг 5.8 — Аналогічний SQL запит для отримання пристрою за ідентифікатором

Для отримання користувача за ім’ям використовується такий запит, який представлений у лістингу 5.9:

```
var user = _db.Users
    .FirstOrDefault(x => x.Name == User.Identity.Name);
```

Лістинг 5.9 — Запит для отримання користувача за ім’ям

Аналогічний SQL запит виглядав би так для отримання користувача за ім’ям, який представлений у лістингу 5.10:

```
SELECT `u`.`UserId`, `u`.`Login`, `u`.`Name`, `u`.`Password`,
`u`.`Role`
      FROM `Users` AS `u`
      WHERE `u`.`Name` = @__User_Identity_Name_0
      LIMIT 1
```

Лістинг 5.10 — Аналогічний SQL запит для отримання користувача за ім’ям

Для отримання підключень користувача за ідентифікатором з’єднання використовується такий запит, який представлений у лістингу 5.11:

```
var connectedUsers = _db.ConnectedUsers.Where(x =>
x.ConnectionId == Context.ConnectionId).ToArray();
```

Лістинг 5.11 — Запит для отримання підключень користувача за ідентифікатором

Аналогічний SQL запит виглядав би так для отримання підключень користувача за ідентифікатором з’єднання, який представлений у лістингу 5.12:

```
SELECT `c`.`ConnectedUserId`, `c`.`ConnectionId`,
`c`.`GroupeName`, `c`.`UserId`
      FROM `ConnectedUsers` AS `c`
      WHERE `c`.`ConnectionId` = @__Context_ConnectionId_0
      LIMIT 1
```

Лістинг 5.12 — Аналогічний SQL запит для отримання підключень користувача за ідентифікатором з’єднання

Для додавання запису про підключеного до користувача використовується такий запит, який представлений у лістингу 5.13:

```
db.ConnectedUsers.Add(new ConnectedUser { UserId = userId,
ConnectionId = Context.ConnectionId, GroupeName = groupName });
_db.SaveChanges();
```

Лістинг 5.13 — Запит про підключеного до користувача

Аналогічний SQL запит виглядав би так для додавання запису про підключеного до користувача, який представлений у лістингу 5.14:

```
INSERT INTO `ConnectedUsers` (`ConnectionId`, `GroupeName`,
`UserId`)
      VALUES (@p0, @p1, @p2);
```

```
      SELECT `ConnectedUserId`

     FROM `ConnectedUsers`
      WHERE ROW_COUNT() = 1 AND `ConnectedUserId` =
LAST_INSERT_ID();
```

Лістинг 5.13 — Аналогічний SQL запит для додавання запису про підключеного до користувача

5.5 Реалізація відображення даних за допомогою фреймворку Angular

Розглянемо зміни (увімкнення або вимикання кнопки) залежно від даних, що одержуються. Реалізація цього відображається у лістингу 5.14:

```
<mat-card *ngFor="let device of devices" class="device">
    <h1 style="text-align: center;">
        {{device.name}}
    </h1>
    <hr style="width: 100%;" size="2" color="#3f51b5" />
    <mat-card *ngFor="let parameter of device.parameters"
        class="parameter"
        (click)="changeRelayState(parameter.deviceParameterId)"
        [ngClass]="{'RelayStateOn':
parameter.sensorLogs[0].value == '0', 'RelayStateOff':
parameter.sensorLogs[0].value == '1' }">
      <div class="name-block">
        {{parameter.name}} <br> </div>
      <div class="state_block">
        <span [ngStyle]="{'background-
color':parameter.sensorLogs[0].value == '0' ? 'green' : null
}">Вкл</span>     /
        <span [ngStyle]="{'background-
color':parameter.sensorLogs[0].value == '1' ? 'red' : null
}">Викл</span>
      </div>
    </mat-card>
</mat-card>
```

Лістинг 5.14 — Реалізація увімкнення або вимикання кнопки

Відображаємо пристрої на основі отриманих даних.

*ngFor="let device of devices" - структурна директива , яка відображає шаблон для кожного елемента колекції.

*ngFor="let parameter of device.parameters"

Відображаємо кількість параметрів (виводи на ESP 32) у вигляді інформативних кнопок, які відображають теперішній статус реле.

[ngStyle]=" – Встановлює одну або кілька властивостей стилю, зазначених у вигляді пар ключ-значення, розділених двокрапкою.

За допомогою чого робимо колір тла тексту (ВКЛ/ВИКЛ) червоним або зеленим, що б підкреслити статус реле.

Отримуємо колекцію пристроїв, надсилаючи GET запит, які представлений на лістингу 5.15:

```
relays.component.ts
this.http.get<Device[]>('/api/Device/GetDevicesOnlyRelay').subsc
ribe(devices => {
  this.devices = devices;
  console.log(devices);
});
```

Лістинг 5.15 — Отримання колекції пристроїв

Змінюємо статус реле на протилежний, надсилаючи GET запит з ідентифікатором параметра, який представлений на лістингу 5.16:

```
changeRelayState(parameterId: number){
  this.isQuery = true;
  console.log('parameterId = ', parameterId)
  this.http.get('/api/Device/ChangeRelayState?parameterId=' +
parameterId).subscribe(() => {
    this.isQuery = false;
  }, error => this.isQuery = false);
}
```

Лістинг 5.16 — Змінення статусу реле

5.6 Алгоритм управління теплицею

Алгоритм управління теплицею представлений у вигляді UseCase diagram на рисунку 5.3 та у вигляді блок-схеми на рисунку 5.4, який складається з:

1. Авторизація користувача та адміністратора в особистому кабінеті у веб-додатку;
2. Після авторизації користувач, а також адміністратор має можливість переглянути поточні показник лічильника та управляти пристроями приводи реле у реальному часі;
3. Всі дані, які переглядає користувач та адміністратор, передаються у реальному часі з самих датчиків температури та вологості у вигляді топіків на контролер користувача, а далі передаються на MQTT сервер та на веб-сервер для зображення на веб-сайті.

Рисунок 5.3 – UseCase diagram – підключення, отримання даних та керування теплицею

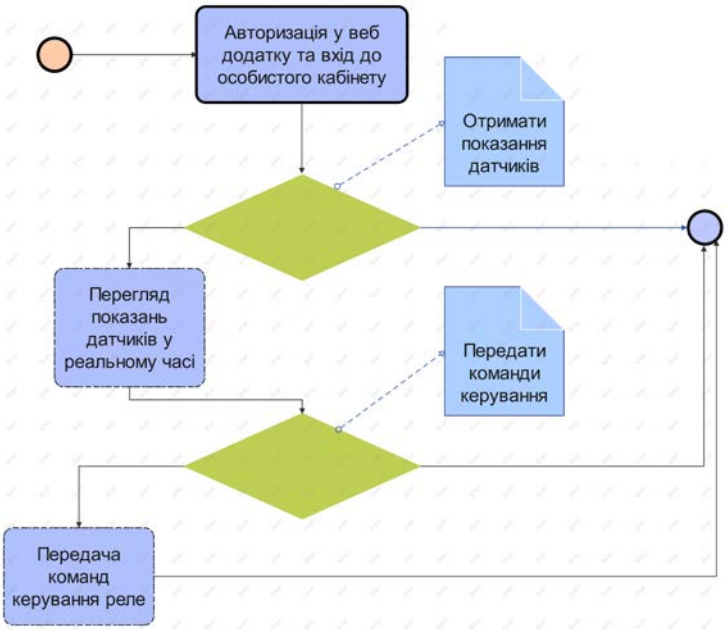

Рисунок 5.4 – Алгоритм по управлінню теплицею представлений у вигляді блок схеми

5.7 Опис взаємодії клієнту із системою управління та моніторингу

На рисунку 5.5 зображено сторінку для авторизації клієнта, для того щоб клієнт зміг за авторизуватися він повинен ввести логін та пароль.

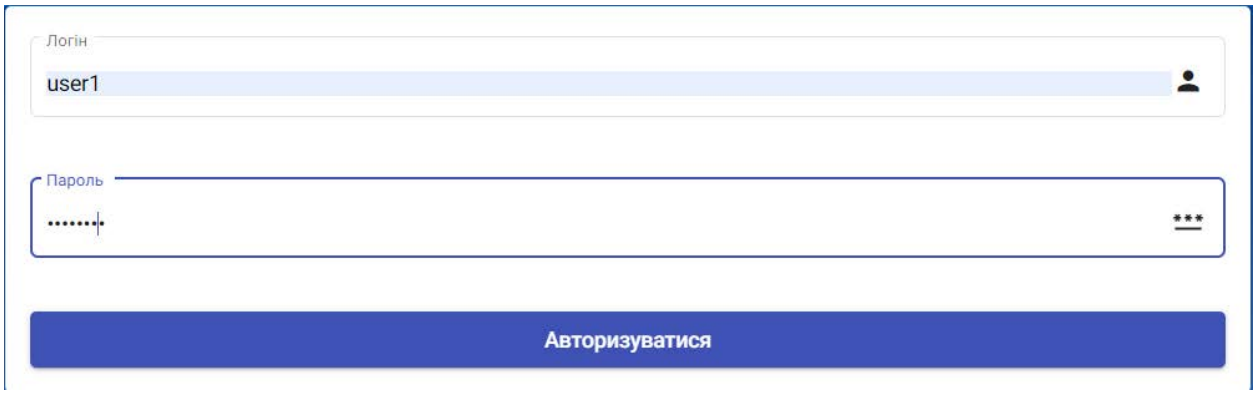

Рисунок 5.5 – Авторизація клієнта

Після авторизації клієнт має змогу керувати двома реле (включати та виключати їх) завдяки передачі на контролер топіків на першій сторінці, яка зображена на рисунку 5.6.

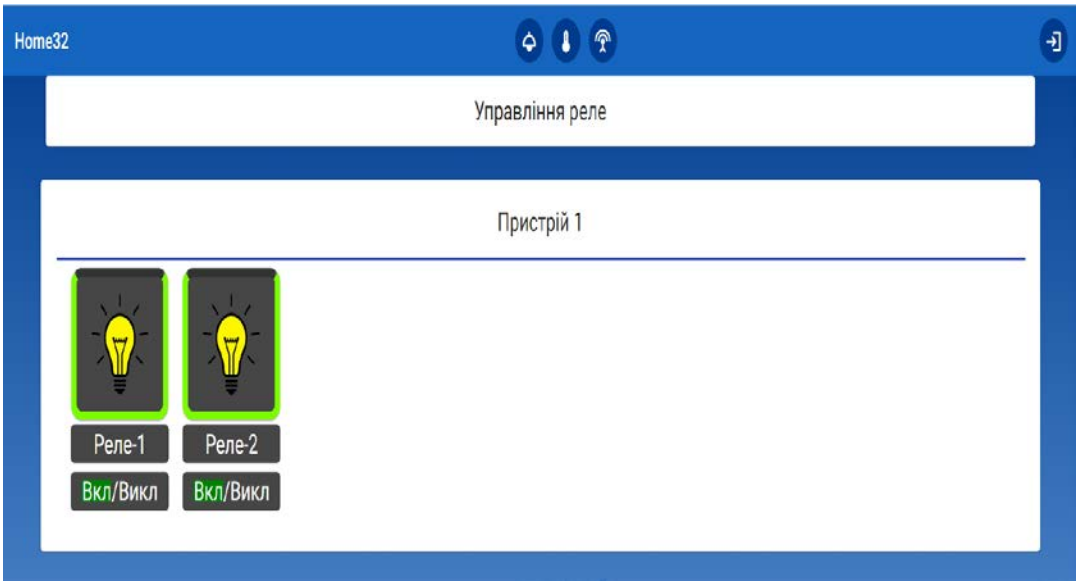

Рисунок 5.6 – Керування реле клієнтом

На другій сторінці показані графіки температури, які зображені на рисунку 5.7. На цій же сторінці зображено графіки вологості, на рисунку 5.8. Кожних 30 хвилин оновлюються графіки температури та вологості.

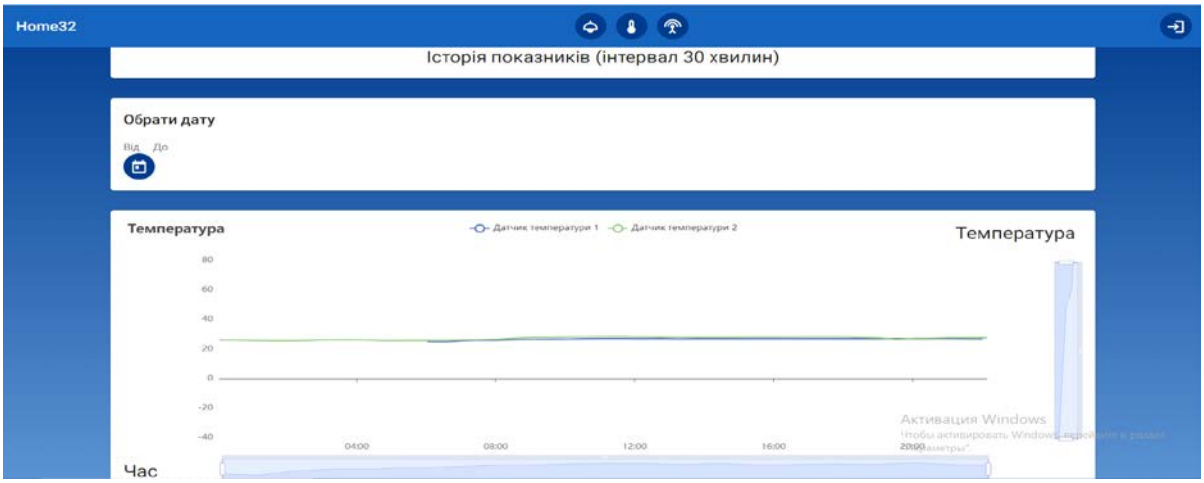

Рисунок 5.7 – Графіки температур

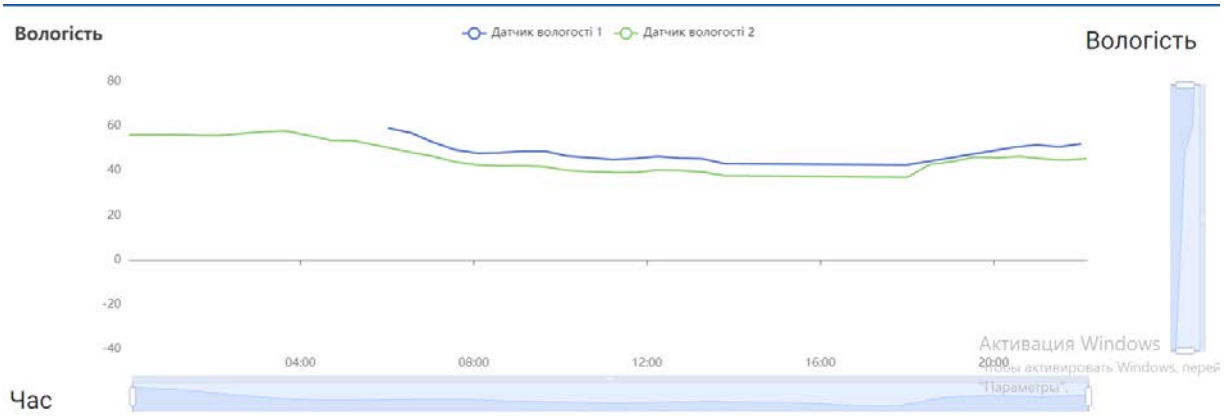

Рисунок 5.8 – Графіки вологості

Також на цій сторінці можливо подивитися минулі дані температури та вологості за конкретний період часу. Щоб вказати конкретний період часу, треба натиснути на кнопку, на якій зображено календар. Відкриється календар, який зображений на рисунку 5.9, в якому можливо вказати період часу. Щоб відобразилися попередні дані, треба натиснути кнопку «Застосувати», якщо треба повернутися до поточних даних, треба натиснути кнопку «Скасувати».

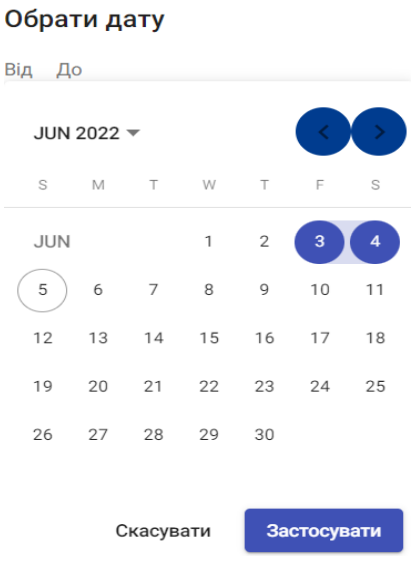

Рисунок 5.9 – Календар

На третій сторінці зображено графіки температури та вологості на рисунку 5.10, які відображаються у реальному часі.

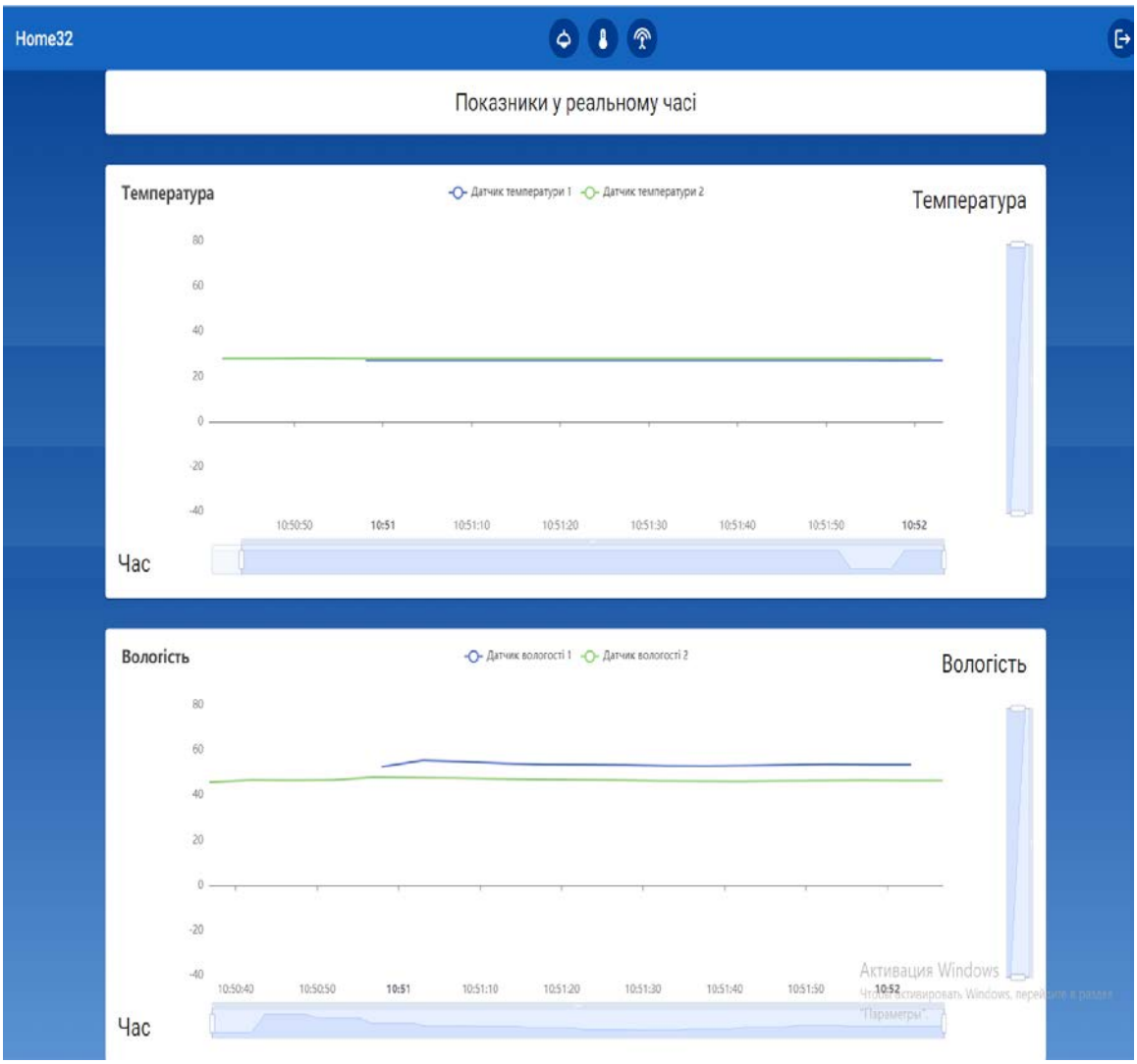

Рисунок 5.10. Графіки температури та вологості у реальному часі

5.8 Перспективи і можливості подальшого розвитку системи

Інтернет речей захопить практично кожен сегмент у сфері промисловості, бізнесу, охорони здоров'я та споживчих товарів.

Інтернет речей пов'язаний зі сферою послуг, галузями промисловості та торгівлі у 2020 р. вплинув на чотири відсотки світового ВВП. Світовий ВВП за 2016 р. склав 75,64 трлн доларів США, а 2020 р. він зріс до 81,5 трлн доларів. Таким чином, в інтернет речах було залучено від 5,68 трлн доларів.

Вплив інтернету речей проявляється у вигляді:

1) нових джерел доходу (отримання електроенергії екологічно чистим методом);
2) скорочення витрат (догляд за пацієнтами вдома);
3) скорочення терміну виведення товару ринку (автоматизація виробництва);
4) удосконалення структури ланцюжка поставок (облік матеріальних активів);
5) скорочення виробничих втрат (крадіжка, псування товарів із коротким терміном придатності);
6) підвищення продуктивності (машинне навчання та аналіз даних);
7) витіснення (розумний термостат Nest витісняє з ринку звичайні термостати).

У майбутньому в системі мінітеплиці можна буде додатково реалізувати систему аналізу даних на основі даних, які були отримані від різних датчиків, наприклад, від датчика освітленості та датчика температури та вологості. Це дозволить робити прогноз з посіву продукції та оптимізації витрат вирощування продукції, використовуючи аналітику або машинне навчання.

У цьому розділі була розглянута розробка архітектури, алгоритм управління теплицею, розробка електричних принципових схем модулів для

керування та моніторингу теплиць, розробка програмного забезпечення модуля та перспективи і можливості подальшого розвитку системи.

Розроблена система з трирівневою архітектурою дозволяє віддалено керувати теплицею та моніторити процеси, які відбуваються в теплиці (температура та вологість). Система продемонструвала себе надійно працюючою навіть при нестабільному інтернет з'єднанні. Також при цьому швидкодія системи дозволяє в реальному часі керувати системою та отримувати дані. Вибрана архітектура системи зберігає всі дані (статистка) у хмарі, що дає користувачеві почуття незалежності від розташування та надійне зберігання даних.

ВИСНОВКИ

Представлену дипломну роботу спрямовано на проектування та реалізацію віддаленого моніторингу та управління програмними системами.

Трирівнева архітектура дає можливість дистанційно моніторити та керувати програмними системами, а саме в даній конкретній предметній області, такій як тепличне господарювання. Трирівневу архітектуру можна застосовувати не тільки в цій предметній області,але в інших, а саме:

1. Промисловий інтернет речей;
2. Роздрібна торгівля, фінанси та маркетинг;
3. Медицина;
4. Транспортування та логістика;
5. Розумне місто.

Завдяки використанню MQTT протоколу для передачі даних в трирівневій архітектурі в системі покращилась надійність завдяки своїй простоті, а також пропускна здібність.

У подальшому розвитку системи є такі напрямки:

1. Зйомка земель за допомогою безпілотних літальних апаратів;
2. Розумні системи поливу та добрива для підвищення врожайності;
3. Оптимізація ланцюжка постачання фермерської продукції ринку з урахуванням матеріальних активів;
4. Планово-попереджувальний ремонт обладнання дистанційних теплиць під контролем виробника;
5. Автоматизація теплиці.

За результатами даної роботи опубліковано тези на всеукраїнській конференції [15].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розумна мікротеплиця Biopod [Електронний ресурс] Режим доступу: http://obzor.bymobile.ru/2015/10/25/biopod/
2. InnoGro – домашня ферма на основі аквапоніки [Електронний ресурс] Режим доступу: https://mastergrad.com/blogs/post/10601/
3. Екосистема Grove дозволить вирощувати продукти за допомогою акваріуму [Електронний ресурс] Режим доступу: https://mastergrad.com/blogs/post/9695/
4. Петін В. Нові можливості. Нові можливості Arduino, ESP, Raspberry Pi в проектах IoT. БХВ-Петерберг, 2022. – 27 с.
5. Що таке Arduino ? [Електронний ресурс] Режим доступу: https://doc.arduino.ua/ru/about/
6. Введення в Angular. Що таке Angular. Перший проект [Електронний ресурс] Режим доступу: https://metanit.com/web/angular2/1.1.php
7. Введення у TypeScript [Електронний ресурс] Режим доступу: https://www.jscamp.app/ru/docs/typescript00/
8. Думки вголос про TypeScript [Електронний ресурс] Режим доступу: https://habr.com/ru/post/272055/
9. Перрі Лі. Архітектура інтернету речей. ДМК Пресс, 2019. – 285 с.
10. Що таке MQTT і для чого він потрібний у IIoT ? Опис протоколу MQTT [Електронний ресурс] Режим доступу: https://ipc2u.ru/articles/prostye-resheniya/chto-takoe-mqtt/
11. Датчик вологості та температури DHT11 [Електронний ресурс] Режим доступу:https://3d-diy.ru/wiki/arduino-datchiki/datchik-vlazhnosti-i-temperatury-dht11/
12. 1-канальний модуль реле JQC-3FF-S-Z [Електронний ресурс] Режим доступу:https://iarduino.ru/shop/Expansion-payments/rele-elektromehanicheskoe-do-250v-10-a-2--kanala-5v.htm

13. Реле електромеханічне до 250V 10 А. 2-каналу 5V [Електронний ресурс] Режим доступу: https://synthetic.ua/ru/product/1-kanalnyi-modul-rele-jqc-3ff-s-z/P02833TU
14. LCD 12864 графічний дисплей 128x64 [Електронний ресурс] Режим доступу:https://arduino.ua/ru/prod349-lcd-12864-graficheskii-displei-128x64-sinii
15. Тіщенко В.І., Віддалений моніторинг і управління програмними системами // Тезиси доповідей XVIII всеукраїнська конференція студентів і молодих науковців «Інформатика, інформаційні системи та технології», Одеса, 23 квітня 2021 р. – Одеса 2021. – с.191.

ДОДАТОК А

Програмна реалізація управління контролером esp32 першим клієнтом

ПРИВ’ЯЗКА ДО АПАРАТНИХ ПОРТІВ КОНТРОЛЕРА

У цьому фрагменті коду блок реле керується 16 та 17 портом контролера. Датчик температури та важності підключений до 15 порту. Кнопки управління №1 та №2 підключені до 19 та 4 порту. LSD 128X64-дисплей підключаєм 18, 23, 5, 22 порту згідно сигналам clock, data, CS, reset.

```
#define relay_1 16
#define relay_2 17
#define DHTPIN 15
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
#define button_1 19
#define button_2 4

U8G2_ST7920_128X64_F_SW_SPI u8g2(U8G2_R0, /* clock=*/ 18, /*
data=*/ 23, /* CS=*/ 5, /* reset=*/ 22); // ESP32
```

ПАРАМЕТРИ ПІДКЛЮЧЕННЯ ТА РЕЖИМИ РОБОТИ КОНТРОЛЕРА

У цьому фрагменті коду призначаємо інтервал таймера. Параметри підключення до Wi-fi мережі (ім'я мережі та пароль). Параметри підключення до брокеру: IP-адрес, логін та пароль, вказання кореневого topic клієнта та порту.

```
// Таймер
const unsigned long eventTime_1 = 5000; // інтервал у ms
unsigned long previousTime_1 = 0;

// WiFi
const char* ssid = "Madagascar"; // Введіть ім'я WiFi
const char* password = "220977qQ";  // Введіть пароль WiFi

// MQTT Broker
const char* mqtt_broker = "164.90.231.188";
const char* topic = "esp32_1_/const char* mqtt_username = "";
```

```
const char* mqtt_password = "";
const int mqtt_port = 1883;
```

У цьому фрагменті коду встановлюються значення стартових параметрів роботи та стану портів.

```
// Стан реле
char* relay_1_state = "0";
char* relay_2_state = "0";
char* relay_1_state_remote = "0";
char* relay_2_state_remote = "0";
bool photoresistor_state = false;

WiFiClient espClient;
PubSubClient client(espClient);
//Thread sensorReader = Thread();
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  dht.begin();

  pinMode(relay_1, OUTPUT);
  pinMode(relay_2, OUTPUT);
  pinMode(button_1, INPUT);
  pinMode(button_2, INPUT);
  pinMode(photoresistor, INPUT);
  pinMode(14, OUTPUT);
  pinMode(3, INPUT);
```

У цьому фрагменті коду встановлюються значення інтервалу послідовної передачі даних.

```
  // Встановіть програмну послідовну передачу даних на 115200;
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Reconnecting...");
    delay(5000);
    Serial.println(WiFi.status());
    WiFi.begin(ssid, password);
    Serial.println(WiFi.status());
  }
```

```
  // За замовчуванням порт 3232
  // ArduinoOTA.setPort(3232);

 // Ім’я хосту за замовчуванням - esp3232-[MAC]
  ArduinoOTA.setHostname("ESP32_1_");

  // За замовчуванням немає аутентифікації
  // ArduinoOTA.setPassword("admin");

  // Пароль також можна встановити зі значенням md5
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  //
ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");

  ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

    // NOTE: if updating SPIFFS this would be the place to
unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total /
100)));
  })
```

У цьому фрагменті коду виводяться на друк у COM порт значення, а саме основні параметри з'єднання та помилки. (Зроблено для спрощення процесу налагодження та тестування).

```
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin
Failed");
```

```
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect
Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive
Failed");
    else if (error == OTA_END_ERROR) Serial.println("End
Failed");
  });

  ArduinoOTA.begin();
  Serial.println("Ready");

Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  client.setServer(mqtt_broker, mqtt_port);
  client.setCallback(callback);
  while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt
broker\n", client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username,
mqtt_password)) {
      Serial.println("Public emqx mqtt broker connected");
    }
    else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
  /*
    sensorReader.enabled = true; // За умовчанням увімкнено
значення true
    sensorReader.setInterval(10000); // Встановлює бажаний
інтервал на 10 мс
    sensorReader.ThreadID = 1;
    sensorReader.onRun(sensorReaderCallback); //
callback_function is the name of the function
  */
```

У цьому фрагменті коду відбувається публікація та підписка клієнта на відповідні топіки.

```
  // опублікувати та підписатися
```

```
  client.publish("esp32_2_/1Test", "ESP32 1");
  client.subscribe("esp32_1_/1Test");
  client.subscribe("esp32_1_/relay_1");
  client.subscribe("esp32_1_/relay_2");
  client.subscribe("esp32_2_/relay_1");
  client.subscribe("esp32_2_/relay_2");
  client.publish("esp32_1_/relay_1", "0");
  client.publish("esp32_1_/relay_2", "0");
}
/*
  void sensorReaderCallback() {
  char humidity[10];
  sprintf (humidity, "%f", dht.readHumidity());
  client.publish("esp32_1_/dht22/Humidity", humidity);
  char temperature[10];
  sprintf (temperature, "%f", dht.readTemperature());

client.publish("esp32_1_/dht22/Temperature", temperature);
  char gpio14[10];
  sprintf (gpio14, "%d", digitalRead(14));
  client.publish("esp32_1_/gpio14", gpio14);

  }
*/
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.println(topic);
  Serial.print(" : ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  /*
    if (payload[0] == '1') {
    digitalWrite(relay_1, 1);
    }
    if (payload[0] == '0') {
    digitalWrite(relay_1, 0);
    }
  */
  if (String(topic) == "esp32_1_/relay_1") {
    if (payload[0] == '0') {
      digitalWrite(relay_1, false);
      relay_1_state = "0";
    }
    else if (payload[0] == '1') {
      digitalWrite(relay_1, true);
      relay_1_state = "1";
```

```
    }
  }
  else if (String(topic) == "esp32_1_/relay_2") {
    if (payload[0] == '0') {
      digitalWrite(relay_2, false);
      relay_2_state = "0";
    }
    else if (payload[0] == '1') {
      digitalWrite(relay_2, true);
      relay_2_state = "1";
    }
  }
  else if (String(topic) == "esp32_2_/relay_1") {
    if (payload[0] == '0') {
      relay_1_state_remote = "0";
    }
    else if (payload[0] == '1') {

relay_1_state_remote = "1";
    }
  }
  else if (String(topic) == "esp32_2_/relay_2") {
    if (payload[0] == '0') {
      relay_2_state_remote = "0";
    }
    else if (payload[0] == '1') {
      relay_2_state_remote = "1";
    }
  }
}
void loop() {
  ArduinoOTA.handle();
  client.loop();
  //sensorReader.run();
  if (digitalRead(button_1) == true) {
    delay(200);
    if (relay_1_state_remote == "0") {
      client.publish("esp32_2_/relay_1", "1");
    }
    else if (relay_1_state_remote == "1") {
      client.publish("esp32_2_/relay_1", "0");
    }
    client.publish("esp32_1_/button_1", "pressed");
  }
  if (digitalRead(button_2) == true) {
    delay(200);
```

```
    if (relay_2_state_remote == "0") {
      client.publish("esp32_2_/relay_2", "1");
    }
    else if (relay_2_state_remote == "1") {
      client.publish("esp32_2_/relay_2", "0");
    }
    client.publish("esp32_1_/button_2", "pressed");
  }
```

У цьому фрагменті коду оновлюється таймер після дії, що відбулася, для наступного.

```
  /* Оновлення часто */
  unsigned long currentTime = millis();
  //Serial.println(currentTime - previousTime_1);
  /* Це подія 1 */
  if (currentTime - previousTime_1 >= eventTime_1) {
    float hum = dht.readHumidity();
    float temp = dht.readTemperature();
    client.publish("esp32_1_/dht22/humidity",
String(hum).c_str());

      client.publish("esp32_1_/dht22/temperature",
String(temp).c_str());
    if(digitalRead(photoresistor)){
      client.publish("esp32_1_/photoresistor", "0");
      photoresistor_state = false;
    }
    else{
      client.publish("esp32_1_/photoresistor", "1");
      photoresistor_state = true;
    }
    /* Оновіть час для наступної події */
    previousTime_1 = currentTime;
  }
}
```

ДОДАТОК Б

Програмна реалізація управління контролером esp32 другим клієнтом

ПРИВ’ЯЗКА ДО АПАРАТНИХ ПОРТІВ КОНТРОЛЕРА

У цьому фрагменті коду блок реле керується 16 та 17 портом контролера. Датчик температури та важності підключений до 15 порту. Кнопки управління №1 та №2 підключені до 19 та 4 порту. LSD 128X64-дисплей підключаєм 18, 23, 5, 22 порту згідно сигналам clock, data, CS, reset.

```
#define relay_1 16
#define relay_2 17
#define DHTPIN 15
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
#define button_1 19
#define button_2 4

U8G2_ST7920_128X64_F_SW_SPI u8g2(U8G2_R0, /* clock=*/ 18, /*
data=*/ 23, /* CS=*/ 5, /* reset=*/ 22); // ESP32
```

ПАРАМЕТРИ ПІДКЛЮЧЕННЯ ТА РЕЖИМИ РОБОТИ КОНТРОЛЕРА

У цьому фрагменті коду призначаємо інтервал таймера. Параметри підключення до Wi-fi мережі (ім'я мережі та пароль). Параметри підключення до брокеру: IP-адрес, логін та пароль, вказання кореневого topic клієнта та порту.

```
// Таймер
const unsigned long eventTime_1 = 5000; // інтервал у ms
unsigned long previousTime_1 = 0;
// WiFi
const char *ssid = "Madagascar"; // Введіть ім'я Wi-Fi
const char *password = "220977qQ";  // Введіть пароль Wi-Fi
// MQTT Broker
const char *mqtt_broker = "164.90.231.188";
const char *topic = "esp32_2_/";
const char *mqtt_username = ""const char *mqtt_password = "";
const int mqtt_port = 1883;
```

У цьому фрагменті коду встановлюються значення стартових параметрів роботи та стану портів.

```
// Стан реле
char *relay_1_state = "0";
char *relay_2_state = "0";
char *relay_1_state_remote = "0";
char *relay_2_state_remote = "0";

// DHT22 State
String humidity = "00.00%";
String temperature = "00.00°C";

// Фоторезисторний датчик
bool day = true;

WiFiClient espClient;
PubSubClient client(espClient);
//Thread sensorReader = Thread();
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  dht.begin();
  u8g2.begin();
  pinMode(relay_1, OUTPUT);
  pinMode(relay_2, OUTPUT);
  pinMode(button_1, INPUT);
  pinMode(button_2, INPUT);
  pinMode(14, OUTPUT);
  pinMode(3, INPUT);
```

У цьому фрагменті коду встановлюються значення інтервалу послідовної передачі даних.

```
  // Встановіть програмну послідовну передачу даних на 115200;
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
```

```
  Serial.println("Connected to Wifi");

  // За замовчуванням порт 3232
  // ArduinoOTA.setPort(3232);

// Ім’я хосту за замовчуванням – esp3232-[MAC]
  ArduinoOTA.setHostname("ESP32_2_");
  // За замовчуванням немає аутентифікації
  // ArduinoOTA.setPassword("admin");
  // Пароль також можна встановити зі значенням md5
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  //
ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");

  ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

    // NOTE: if updating SPIFFS this would be the place to
unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total /
100)));
  })
```

У цьому фрагменті коду виводяться на друк в COM порт значення, а саме основних параметрів з'єднання і помилки. (Зроблено для спрощення процесу налагодження та тестування).

```
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin
Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect
Failed");
```

```
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive
Failed");
    else if (error == OTA_END_ERROR) Serial.println("End
Failed");
  });

  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

client.setServer(mqtt_broker, mqtt_port);
  client.setCallback(callback);
  while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public mqtt
broker\n", client_id.c_str());
    if (client.connect(client_id.c_str(), mqtt_username,
mqtt_password)) {
      Serial.println("Public emqx mqtt broker connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
  /*
    sensorReader.enabled = true; // За умовчанням увімкнено
значення true
    sensorReader.setInterval(10000); // Встановлює бажаний
інтервал на 10 мс
    sensorReader.ThreadID = 1;
    sensorReader.onRun(sensorReaderCallback); //
callback_function is the name of the function
  */
```

У цьому фрагменті коду відбувається публікація та підписка клієнта на відповідні топіки.

```
  // опублікувати та підписатися
  client.publish("esp32_2_/1Test", "ESP32 1");
  client.subscribe("esp32_2_/1Test");
  client.subscribe("esp32_2_/relay_1");
  client.subscribe("esp32_2_/relay_2");
  client.subscribe("esp32_1_/relay_1");
```

```
  client.subscribe("esp32_1_/relay_2");
  client.subscribe("esp32_1_/dht22/humidity");
  client.subscribe("esp32_1_/dht22/temperature");
  client.subscribe("esp32_1_/photoresistor");
  client.publish("esp32_2_/relay_1", "0");
  client.publish("esp32_2_/relay_2", "0");
}
void sensorReader() {
  float hum = dht.readHumidity();
  float temp = dht.readTemperature();
  client.publish("esp32_2_/dht22/humidity",
String(hum).c_str());
  client.publish("esp32_2_/dht22/temperature",
String(temp).c_str());
  u8g2.clearBuffer();


u8g2.setFont(u8g2_font_unifont_t_symbols);
  u8g2.drawUTF8(73, 10, temperature.c_str());
  u8g2.drawUTF8(0, 10, humidity.c_str());
  u8g2.drawUTF8(73, 64, (String("") + temp + "°C").c_str());
  u8g2.drawUTF8(0, 64, (String("") + hum + "%").c_str());
  if (day) {
    u8g2.drawUTF8(0, 30, (String("Day")).c_str());
  }
  else {
    u8g2.drawUTF8(0, 30, (String("Night")).c_str());
  }
  u8g2.sendBuffer();
}
void callback(char *topic, byte *payload, unsigned int length) {
  Serial.println(topic);
  Serial.print(" : ");
  for (int i = 0; i < length; i++) {
    Serial.print((char) payload[i]);
  }
  /*
    if (payload[0] == '1') {
    digitalWrite(relay_1, 1);
    }
    if (payload[0] == '0') {
    digitalWrite(relay_1, 0);
    }
  */
  if (String(topic) == "esp32_2_/relay_1") {
    if (payload[0] == '0') {
```

```
      digitalWrite(relay_1, false);
      relay_1_state = "0";
    }
    else if (payload[0] == '1') {
      digitalWrite(relay_1, true);
      relay_1_state = "1";
    }
  }
  else if (String(topic) == "esp32_2_/relay_2") {
    if (payload[0] == '0') {
      digitalWrite(relay_2, false);
      relay_2_state = "0";
    }
    else if (payload[0] == '1') {
      digitalWrite(relay_2, true);
      relay_2_state = "1";
   }
  else if (String(topic) == "esp32_1_/relay_1") {
    if (payload[0] == '0') {
      relay_1_state_remote = "0";
    }
    else if (payload[0] == '1') {
      relay_1_state_remote = "1";
    }
  }
  else if (String(topic) == "esp32_1_/relay_2") {
    if (payload[0] == '0') {
      relay_2_state_remote = "0";
    }
    else if (payload[0] == '1') {
      relay_2_state_remote = "1";
    }
  }
  else if (String(topic) == "esp32_1_/dht22/humidity") {
    humidity = (String("") + (char)payload[0] + (char)payload[1]
+ (char)payload[2] + (char)payload[3] + (char)payload[4] +
"%").c_str();
  }
  else if (String(topic) == "esp32_1_/dht22/temperature") {
    temperature = (String("") + (char)payload[0] +
(char)payload[1] + (char)payload[2] + (char)payload[3] +
(char)payload[4] + "°C").c_str();
  }
  else if (String(topic) == "esp32_1_/photoresistor") {
    if ((char)payload[0] == '0') {
      day = false;
```

```
    }
    else if ((char)payload[0] == '1') {
      day = true;
    }
  }
}
void loop() {
  ArduinoOTA.handle();
  client.loop();
  //sensorReader.run();
  if (digitalRead(button_1) == true) {
    delay(200);
    if (relay_1_state_remote == "0") {
      client.publish("esp32_1_/relay_1", "1");
    }
    else if (relay_1_state_remote == "1") {

client.publish("esp32_1_/relay_1", "0");
    }
    client.publish("esp32_2_/button_1", "pressed");
  }
  if (digitalRead(button_2) == true) {
    delay(200);
    if (relay_2_state_remote == "0") {
      client.publish("esp32_1_/relay_2", "1");
    }
    else if (relay_2_state_remote == "1") {
      client.publish("esp32_1_/relay_2", "0");
    }
    client.publish("esp32_2_/button_2", "pressed");
  }
```

У цьому фрагменті коду оновлюється таймер після дії, що відбулася, для наступного.

```
  /*Оновлення часто */
  unsigned long currentTime = millis();
  //Serial.println(currentTime - previousTime_1);
  /*Це подія 1 */
  if ( currentTime - previousTime_1 >= eventTime_1 ) {
    sensorReader();
    /*Оновіть час для наступної події */
    previousTime_1 = currentTime;
```