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))

Кваліфікаційна робота
на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

тема: Проектування і розробка IoT системи
Design and development of the IoT system

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія .
(шифр і назва напряму підготовки, спеціальності)

Єлесін Олег Олександрович
(прізвище, ім'я, по-батькові)

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

Рецензент ст. викл. Максимов О.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№ 12 від « 09 » 06 2023 р.

Захищено на засіданні ЕК № ___
протокол № __ від «___» ___ 2023 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (ім'я, прізвище)

Голова ЕК
_____ Алла КОБОЗЄВА
(підпис) (ім'я, прізвище)

Одеса – 2023

АНОТАЦІЯ

В дипломній роботі розглянуті та проаналізовані сучасні системи розумного будинку. Проведено огляд сучасних протоколів та мікросхем для реалізації IoT мереж. Виконано проектування плану будинку та функціоналу IoT системи.

Обране обладнання для реалізації IoT системи розумного будинку. Побудовані принципові схеми пристроїв розумного будинку. Розроблені функції мікроконтролерів пристроїв. Розроблена програма, яка керує пристроями розумного будинку та виконує команди з мобільного додатку. Створена система рекомендацій, яка пропонує користувачу сценарії використання пристроїв на основі даних з них. Розроблена програма розпізнавання осіб, яка застосовується до камер відеоспостереження. Створений мобільний додаток, через який користувач отримує інформацію та керує системою. Проведене тестування IoT системи.

За результатами дипломної роботи опубліковані тези доповідей двадцятої всеукраїнської конференції студентів та молодих науковців [1].

ABSTRACT

The thesis reviews and analyzes modern smart home systems. A review of modern protocols and chips for the implementation of IoT networks was conducted. The design of the house plan and the functionality of the IoT system was carried out.

The equipment for the implementation of the IoT system of a smart home is selected. Schematic diagrams of smart home devices are built. The functions of microcontrollers of devices are developed. A program was developed that controls smart home devices and executes commands from a mobile application. A recommendation system was created that offers the user scenarios for using devices based on data from them. Developed a face recognition program that is applied to video surveillance cameras. A mobile application was created through which the user receives information and controls the system. The IoT system was tested.

Based on the results of the thesis, the abstracts of the twentieth all-Ukrainian conference of students and young scientists were published [1].

ЗМІСТ

Стор.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ 6

ВСТУП 7

1 ОГЛЯД І АНАЛІЗ ІОТ СИСТЕМ РОЗУМНОГО БУДИНКУ 8

1.1 Поняття і визначення технології інтернет речей 8

1.2 Огляд сучасних IoT систем «Розумний будинок» 12

1.3. Огляд сучасних протоколів технологій IoT мереж 14

1.4 Огляд сучасних мікроконтролерів для побудови системи «Розумний будинок» 17

1.5 Постановка завдання 21

2 ПРОЕКТУВАННЯ ІОТ СИСТЕМИ ОБ'ЄКТА «РОЗУМНИЙ БУДИНОК» 22

2.1 План будинку і розміщення IoT елементів 22

2.2 Проектування функціонального набору IoT системи об'єкта «Розумний будинок» 23

2.3 Проектування функцій IoT системи 29

3 РОЗРОБКА ІОТ СИСТЕМИ ОБ'ЄКТА «РОЗУМНИЙ БУДИНОК» 38

3.1 Вибір обладнання IoT мережі 38

3.2 Принципові схеми підключення пристроїв 46

3.3 Розробка функцій безпечного підключення мікроконтролерів 50

3.4 Розробка керуючої програми 55

3.5 Створення підсистеми аналізу використання IoT-системи 61

3.6 Розробка підсистеми розпізнавання осіб 65

3.7 Створення мобільного додатку клієнта IoT-системи 68

3.8. Розробка макету та тестування IoT системи 71

ВИСНОВКИ 77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 78
ДОДАТОК А 80
ДОДАТОК Б 90
ДОДАТОК В 98

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

IoT (Internet of Things) – Інтернет речей.

DC (Direct current) – постійний струм.

AC (Alternating current) – змінний струм.

ТЕРМІНИ

I2C – послідовна асиметрична шина зв'язку між інтегральними схемами всередині електронних приладів.

ВСТУП

Сьогодні, у сучасному світі, технології розвиваються з неймовірною швидкістю, а люди все більше шукають ефективні та зручні рішення для свого життя. В тому числі, становляться актуальними та потребують подальшого розвитку технології та системи побудови об’єктів «Розумний будинок»

Мета даної дипломної роботи – проектування та розробка IoT-системи для об’єкту «Розумний будинок» з підвищеними вимогами до безпеки, гнучкості та інтелектуальними можливостями.

Для досягнення поставленої мети потрібно вирішити такі завдання:

1) Провести огляд сучасних рішень для побудови мереж та систем «Розумний будинок».
2) Аналіз сучасного обладнання та протоколів мережі IoT для системи «Розумний будинок».
3) Формулювання постановки завдання.
4) Виконати проектування фізичної та логічної топології IoT-мережі об’єкту «Розумний будинок».
5) Спроектувати функціонального набору IoT-системи.
6) Виконати проектування алгоритмів функцій пристроїв IoT-мережі.
7) Обрати необхідне обладнання та розробка принципових схем пристроїв мережі IoT.
8) Розробити функції захищеного підключення пристроями системи та їх функціювання.
9) Розробити програму, яка керує пристроями та сценаріями IoT-системи.
10) Розробити підсистему рекомендацій використання IoT-системи «Розумний будинок» на основі досвіду експлуатації.
11) Розробити підсистему розпізнавання осіб з використанням існуючих бібліотек.
12) Розробити мобільний додаток клієнта IoT-системи.
13) Виконати макетування та тестування пристроїв IoT системи.

1 ОГЛЯД І АНАЛІЗ IOT СИСТЕМ РОЗУМНОГО БУДИНКУ

1.1 Поняття і визначення технології інтернет речей

Інтернет речей [2] - це технологія, яка дозволяє об'єднувати різноманітні пристрої та датчики, що мають доступ в Інтернет та можуть обмінюватися даними між собою без потреби в прямому участі людини. Такі пристрої можуть включати в себе різні речі, від побутових пристроїв, таких як холодильники та пральні машини, до важкої промисловості, такої як обладнання для виробництва та транспортування продуктів. Схематичне зображення сфер застосування IoT наведено на рис.1.1.

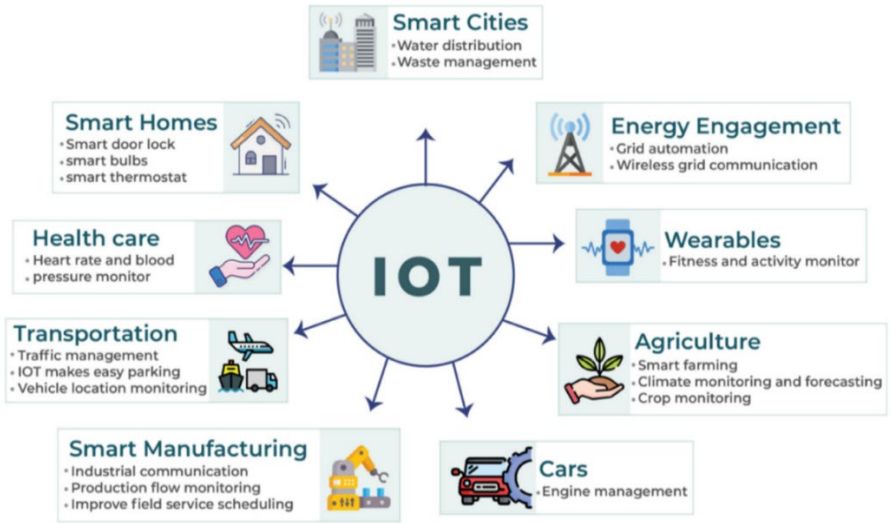

Рисунок 1.1 – Схематичне зображення сфер застосування IoT

Технологія IoT стала можливою завдяки розвитку бездротових мереж та датчиків, які можуть приймати та відправляти дані в режимі реального часу. Ці дані можуть бути оброблені та проаналізовані за допомогою спеціального програмного забезпечення та алгоритмів машинного навчання, що дозволяє створити автоматичну систему, яка може відслідковувати, аналізувати та управляти пристроями та процесами в реальному часі.

Однією з головних переваг технології IoT є покращення ефективності та оптимізації процесів. Система IoT дозволяє автоматично відслідковувати та аналізувати дані про використання різноманітних пристроїв та ресурсів, таких як електроенергія, вода та газ. На основі цих даних система може рекомендувати оптимальний режим роботи, що дозволяє зменшити споживання енергії та зменшити витрати.

Однак, з введенням технології IoT, з'являються нові проблеми та виклики, пов'язані з безпекою даних та конфіденційністю користувачів. Оскільки різні пристрої можуть обмінюватися між собою, це може призвести до можливого розголошення особистої інформації користувачів. Крім того, збільшення кількості підключених пристроїв може спричинити проблеми зі стійкістю мережі та збільшити ризик кібератак. Приклади можливих атак показано на рис. 1.2.

Щоб забезпечити безпеку систем IoT [3], необхідно забезпечити захист від несанкціонованого доступу до пристроїв та мережі. Для цього використовуються різні техніки, такі як шифрування, аутентифікація та авторизація, що дозволяють захистити дані користувачів та мережу від несанкціонованого доступу.

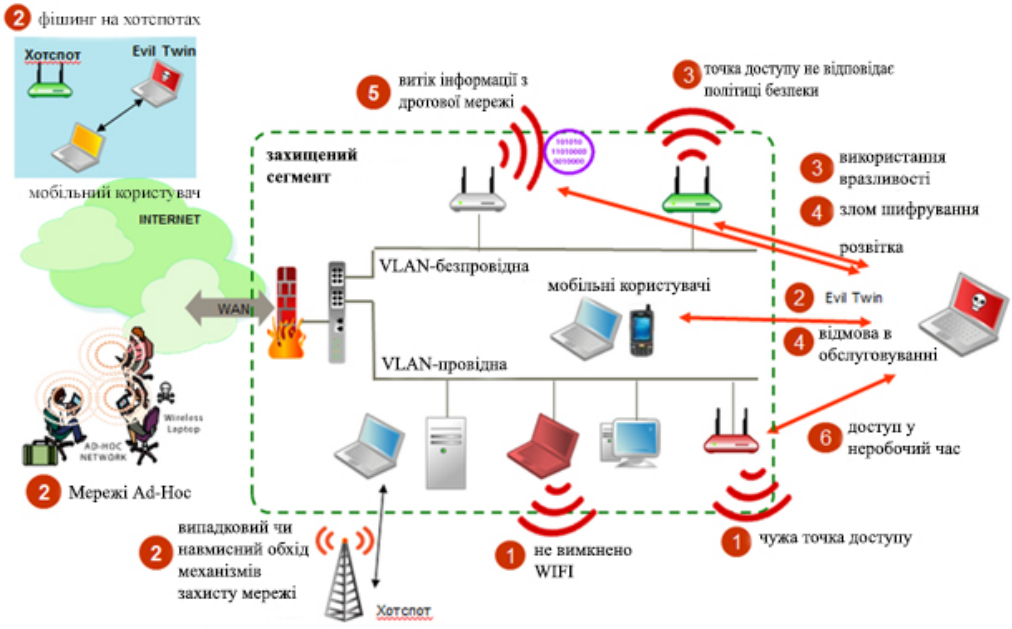

Рисунок 1.2 – Схематичне зображення можливих атак на мережу

Крім того, важливо забезпечити захист від кібератак та інших загроз, пов'язаних з використанням систем IoT. На сьогоднішній день розробники систем IoT використовують різні техніки для забезпечення безпеки, такі як мультифакторна аутентифікація, забезпечення безпечного зберігання та передачі даних, контроль доступу до мережі та пристроїв та багато іншого.

Окрім захисту, розвиток технології IoT зумовлює необхідність у стандартизації та інтеграції пристроїв, що дозволить підвищити їх сумісність та ефективність. Стандартизація дозволить створювати відкриті стандарти та протоколи, які будуть використовуватися різними виробниками пристроїв та мереж. Це забезпечить легку інтеграцію різних пристроїв та мереж у єдину систему, що дозволить покращити їх функціональність та зменшити вартість.

Технологія IoT має великий потенціал у різних галузях, включаючи будівництво, транспорт, охорону здоров'я, виробництво та інше. У будівництві IoT може бути використано для автоматизації різних процесів, включаючи опалення, вентиляцію та кондиціювання повітря, освітлення, системи безпеки та інші.

В транспорті IoT може бути використано для збільшення безпеки дорожнього руху, моніторингу стану транспортних засобів, покращення маршрутизації та зменшення заторів. У галузі охорони здоров'я IoT може бути використано для моніторингу стану хворих та надання дистанційної медичної допомоги.

У виробництві IoT може бути використано для автоматизації різних процесів, включаючи моніторинг стану обладнання та виробничих ліній, планування виробничих процесів, збільшення ефективності використання ресурсів та інші. В цілому, IoT може змінити спосіб, яким працюють різні галузі, забезпечуючи більш ефективну та продуктивну роботу, зменшення витрат та покращення безпеки.

Завдяки розвитку технології IoT, з'являється можливість створення інтелектуальних розумних будинків, які можуть бути повністю автоматизованими та контрольованими дистанційно. Це дає можливість

користувачам забезпечувати безпеку, комфорт та енергоефективність свого будинку за допомогою мобільного додатку або голосового управління.

Однак, для реалізації інтелектуальних розумних будинків необхідно вирішувати складні інженерні та технічні проблеми, пов'язані зі збиранням, обробкою та збереженням великої кількості даних, а також з підключенням та сумісністю різноманітних пристроїв.

Інтелектуальні розумні будинки можуть бути здатні виконувати різноманітні завдання, такі як вимикання та вмикання побутових приладів, регулювання освітлення та опалення, контроль доступу та безпеки тощо.

Одним з ключових компонентів інтелектуальних розумних будинків є система збирання та аналізу даних. Завдяки цій системі можна збирати дані з різних датчиків та пристроїв, аналізувати їх та надавати користувачам звіти та рекомендації з покращення енергоефективності та оптимізації використання ресурсів.

Іншим важливим аспектом інтелектуальних розумних будинків є їх здатність до взаємодії з користувачем. Користувачі можуть взаємодіяти з системою за допомогою мобільного додатку або голосового управління, що дає їм можливість контролювати свій будинок з будь-якого місця та в будь-який час.

Інтелектуальні розумні будинки також можуть забезпечувати високий рівень безпеки та захисту користувачів та їх майна. Наприклад, система контролю доступу може вимагати в ідентифікації користувача для входу в будинок, а системи відеоспостереження та виявлення вторгнень можуть допомогти запобігти крадіжкам та іншим порушенням безпеки.

У цілому, технологія IoT та інтелектуальні розумні будинки можуть допомогти в розвитку "розумного" житла, що може бути енергоефективнішим, комфортнішим та безпечнішим для користувачів. Однак, як і у будь-якій технології, існують певні виклики та обмеження, які потрібно враховувати під час розробки та використання інтелектуальних розумних будинків.

1.2 Огляд сучасних IoT систем «Розумний будинок»

Перш ніж почати розумітися на тонкощах і функціональних можливостях різних систем автоматизації житла, для зручності виділимо основні критерії їх оцінювання. Це можуть бути наступні моменти:

- Функціонал (можливості системи);
- можливість масштабування (додавання обладнання для розширення функціоналу);
- порядок підключення системних елементів між собою.
- спосіб керування (через ПК, смартфон, панель управління);
- канали зв'язку із користувачем (Інтернет, GSM, радіоканал);
- інтелектуальність системи;
- дальність роботи сигналу;
- вартість.

1) AJAX

Устаткування Ajax [4] працює на зашифрованому та захищеному двосторонньому радіозв'язку Jeweller власної розробки, має повну автономність від електромережі завдяки резервному джерелу живлення - хабу, характеризується стильним дизайном усіх своїх пристроїв. Логотип компанії показаний на рис. 1.3.

Рисунок 1.3 – Логотип компанії «AJAX»

Переваги:

- простий монтаж;
- бездротовий канал зв'язку між системними елементами;

- широка зона дії сигналу (до 2000 м);
- Wi-Fi та GSM-зв'язок;
- встановлення за QR-кодом та управління за допомогою смартфона (iOS, Android);
- підключення до 100 пристроїв.

Недоліки:

- Відсутність автономності датчиків;
- немає власної камери відеоспостереження;
- робота тільки з власними приладами;
- відсутність інноваційних приладів (усі прилади компанії мають аналоги);
- висока ціна;
- відсутність рішень зі штучним інтелектом.

2) BROADLINK

Устаткування «Розумний дім» BroadLink [5] є комплектом цифрових пристроїв, створених для раціонального управління побутовою технікою, а також освітлювальною, енергетичною, охоронною та іншими системами в будинку. Логотип компанії показаний на рис. 1.4.

Рисунок 1.4 – Логотип компанії «Broadlink»

Переваги:

- Простота налаштування;
- має широкий асортимент пристроїв;
- великі можливості масштабування;
- функціонує без центрального хаба;

- бездротова взаємодія пристроїв між собою;

  контролюється через Wi-Fi.

Недоліки:

- Невелика дальність дії сигналу (до 50 м);
- відсутність резервного живлення;
- застаріле ПО;
- відсутність інновацій;
- висока ціна;
- відсутність рішень зі штучним інтелектом.

Як можемо бачити, тема IoT стрімко розвивається і має конкуренте середовище. Проте, кожне готове рішення має свої недоліки і потенційному покупцю важко знайти універсальне і безкомпромісне рішення своїх проблем.

1.3. Огляд сучасних протоколів технологій IoT мереж

На сьогоднішній день існує кілька популярних протоколів для побудови систем "розумний будинок" [6]. Розглянемо троє найпоширеніших із них.

1) Wi-Fi

   Wi-Fi є одним з найпопулярніших протоколів для систем "розумний будинок" завдяки своїй широкій доступності та високій швидкості передачі даних.

   Переваги Wi-Fi включають:

   - Широкий вибір пристроїв: Wi-Fi підтримується більшістю розумних пристроїв, що забезпечує більшу гнучкість у виборі та інтеграції в систему "розумний будинок";
   - висока швидкість передачі даних: Wi-Fi забезпечує високу швидкість передачі даних, що особливо важливо для передачі відео, стрімінгового відтворення та інших вимогливих застосунків;

- широке покриття: Wi-Fi має достатньо широкий охоплення, що дозволяє покривати великі приміщення або будинки без необхідності в додаткових пристроях підсилення сигналу.

Недоліки Wi-Fi:

- Високе споживання енергії: Пристрої, які використовують Wi-Fi, можуть споживати більше енергії порівняно з іншими протоколами, що може зменшувати тривалість роботи від батареї;
- можливі перешкоди: Оскільки Wi-Fi використовує загальнодоступний діапазон частот, можливі проблеми з перешкодами від інших бездротових пристроїв або сусідніх Wi-Fi мереж.

2) Zigbee

Zigbee є ще одним популярним протоколом для систем "розумний будинок". Він відомий своєю надійністю та можливістю підключення великої кількості пристроїв.

Переваги Zigbee включають:

- Ефективне використання енергії: Zigbee спеціально розроблено для низького споживання енергії, що дозволяє пристроям працювати на батарейках тривалий час без заміни;
- велика масштабованість: Мережі Zigbee можуть включати в себе велику кількість пристроїв, що забезпечує гнучкість при розширенні системи "розумний будинок";
- самоорганізація: Пристрої Zigbee можуть самоорганізовуватись в мережах, що спрощує процес підключення нових пристроїв та забезпечує надійну роботу системи.

Недоліки Zigbee:

- Обмежена швидкість передачі даних: У порівнянні з Wi-Fi швидкість передачі даних в мережі Zigbee нижча, що може бути проблемою для передачі відео чи інших високопропускних додатків;

- обмежена сумісність: Деякі пристрої на ринку можуть не підтримувати протокол Zigbee, тому важливо переконатися у сумісності обраних пристроїв;
- потреба в координаторі мережі: Для роботи мережі Zigbee потрібний координатор, що може підвищити вартість системи та складність її встановлення;
- неможливість будь-якій людині побудувати свою мережу на основі цього протоколу.

3) Z-Wave

Z-Wave також є популярним протоколом для систем "розумний будинок". Він відомий своєю надійністю та широким вибором сумісних пристроїв.

Переваги Z-Wave включають:

- Висока надійність: Z-Wave використовує механізм маршрутизації, що забезпечує надійну передачу даних у системі;
- широкий вибір пристроїв: На ринку доступні багато пристроїв, які підтримують протокол Z-Wave, що дозволяє будувати систему з різноманітними функціями;
- простота налаштування: Конфігурація та налаштування мережі Z-Wave зазвичай є простим процесом, що дозволяє швидко розгорнути систему.

Недоліки Z-Wave:

- Обмежена масштабованість: Мережі Z-Wave можуть бути обмежені в кількості підключених пристроїв, що може бути проблемою для великих систем "розумний будинок";
- вартість: продукти, які підтримують протокол Z-Wave, часто коштують трохи більше порівняно з аналогічними пристроями на інших протоколах;
- неможливість будь-якій людині побудувати свою мережу на основі цього протоколу.

Усі протоколи, наприклад такі як Wi-Fi, Zigbee та Z-Wave, мають свої переваги та недоліки, які слід враховувати при будівництві системи "розумний будинок". Обравши протокол, зможемо самостійно побудувати систему, враховуючи свої потреби та можливості.

1.4 Огляд сучасних мікроконтролерів для побудови системи «Розумний будинок»

У данному розділі будуть розглянуті такі мікроконтролери, як: ESP-32, ESP-8266 та Rasberry Pi 3 Model B. Розглянемо кожен із них:

1) ESP-32

ESP-32 є мікроконтролером, розробленим компанією Espressif Systems [7]. Він має двоядерний процесор Xtensa LX6 з тактовою частотою до 240 МГц, а також модулі Wi-Fi та Bluetooth. Один із можливих пристроїв на базі ESP-32 показано на рис. 1.5.

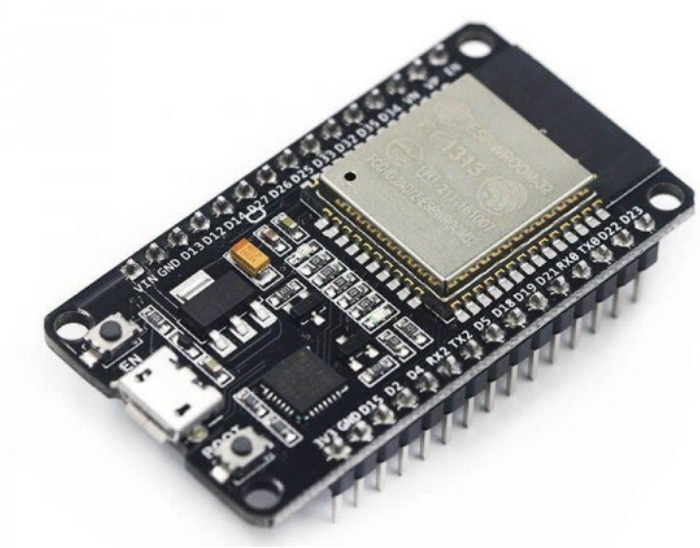

Рисунок 1.5 – Мікроконтролер NodeMCU ESP-32

Деякі основні характеристики ESP32:

- Потужність: ESP32 має достатню обчислювальну потужність для керування пристроями розумного будинку, такими як освітлення, системи безпеки, розумні розетки і т.д.;
- Wi-Fi: ESP32 підтримує протокол Wi-Fi, що дозволяє пристроям у вашому розумному будинку спілкуватися між собою та підключатися до Інтернету;
- bluetooth: Він також має вбудований модуль Bluetooth, що дозволяє підключати пристрої, сумісні з Bluetooth, наприклад, датчики, дверні замки та інші;
- енергоефективність: ESP32 має ефективне споживання енергії, що особливо важливо для бездротових пристроїв у розумному будинку:
- розширюваність: ESP32 має різні порти та інтерфейси для підключення додаткових модулів та датчиків.

2) ESP-8266

ESP8266 - це ще один мікроконтролер від Espressif Systems [8]. Він також має модуль Wi-Fi, але не має вбудованого Bluetooth. Один із можливих пристроїв на базі ESP-32 показано на рис. 1.6.

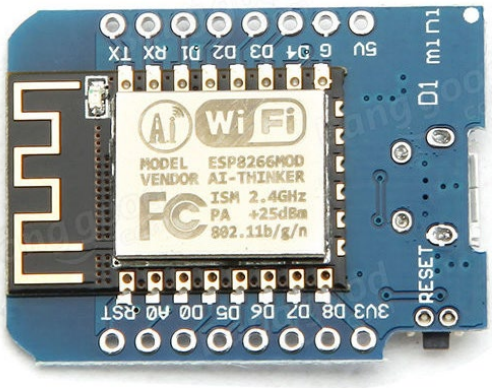

Рисунок 1.6 – Мікроконтролер Wemos D1 mini на базі ESP-8266

Ось деякі особливості ESP-8266:

- Бюджетний варіант: ESP8266 зазвичай доступніший за ціною, ніж ESP-32 та Raspberry Pi 3 Model B, що робить його привабливим варіантом для економних проектів розумного будинку;
- Wi-Fi: ESP8266 підтримує протокол Wi-Fi та може використовуватися для підключення пристроїв до домашньої мережі та Інтернету;
- обмежені ресурси: У відміну від ESP-32, у ESP-8266 одноядерний процесор з нижчою тактовою частотою, і він може бути обмежений в обчислювальних можливостях для складних додатків;
- розширюваність: ESP-8266 має деякі GPIO-піни для підключення додаткових пристроїв та датчиків, але їх може бути обмежена кількість у порівнянні з ESP-32 та Raspberry Pi 3 Model B.

3) Rasberry Pi 3 Model B

Raspberry Pi 3 Model B [9] – це мініатюрний одноплатний комп'ютер, який має більше можливостей, ніж мікроконтролери ESP-32 і ESP-8266. Мікроконтролер показано на рис. 1.7.

Рисунок 1.7 – Мікроконтролер Rasberry Pi 3 Model B

Деякі ключові особливості Raspberry Pi 3 Model B в контексті розумного будинку включають:

- Повноцінний комп'ютер: Raspberry Pi Model B працює під управлінням операційної системи, такої як Raspbian, і може

виконувати багато завдань, включаючи управління розумним будинком;

- Wi-Fi та Ethernet: Raspberry Pi Model B має вбудований Wi-Fi та Ethernet для підключення до мережі та Інтернету, що дозволяє пристроям у розумному будинку обмінюватися даними;
- потужність: Raspberry Pi Model B має потужніший процесор та більше оперативної пам'яті порівняно з ESP-32 та ESP-8266, що дозволяє виконувати складніші завдання та обробляти великі обсяги даних;
- розширюваність: Raspberry Pi Model B має багато GPIO-пінів, USB-портів, HDMI-виходів та інших інтерфейсів, які дозволяють підключати різні пристрої та датчики

В таблиці 1.1 наведена порівняльна характеристика мікроконтролерів

Таблиця 1.1 – Порівняльна характеристика мікроконтролерів

Характеристика	ESP-32	ESP-8266	Raspberry Pi 3 Model B
Процесор	Двоядерний, Xtensa LX6	Одноядерний, Xtensa LX106	Чотирьохядерний, ARM Cortex-A53
Тактова частота	До 240 МГц	До 120 МГц	1.2 ГГц
Wi-Fi	Так	Так	Так
Bluetooth	Так	Ні	Так
Ethernet порт	Ні	Ні	Так
Цифрові піни	16	11	26
Аналогові піни	12	1	0
Оперативна пам'ять	520 КБ	80 КБ	1 ГБ
Операційна система	ні	ні	Raspbian

1.5 Постановка завдання

Задача дипломної роботи – проектування та розробка IoT системи із залученням технологій штучного інтелекту для використання в мережі «Розумному будинку» для приватного будинку. План будинку зображений на рис. 1.8. IoT система повинна включати такі пристрої:

1) Пристрій фіксації відкриття;
2) пристрій фіксації витіку газу:
3) пристрій фіксації протікання;
4) пристрій фіксації температури та вологості;
5) пристрій керування ІЧ-пристроями;
6) пристрій дистанційного керування електроприладами;
7) камера спостереження.

Потрібно розробити програми для мікроконтролерів пристроїв та розробити програму розпізнавання людей с камер відеоспостереження. Система штучного інтелекту повинна видавати рекомендації щодо створення нових сценаріїв на основі даних з пристроїв. Для керування системою передбачається використання мобільного додатку.

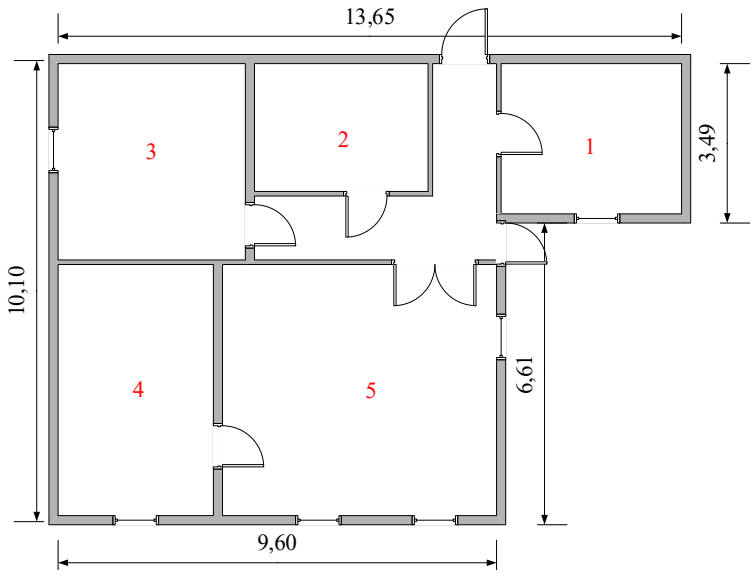

Рисунок 1.8 – План будинку

2 ПРОЕКТУВАННЯ ІОТ СИСТЕМИ ОБ'ЄКТА «РОЗУМНИЙ БУДИНОК»

2.1 План будинку і розміщення IoT елементів

Розміщення елементів IoT на плані будинку показано на рис. 2.1

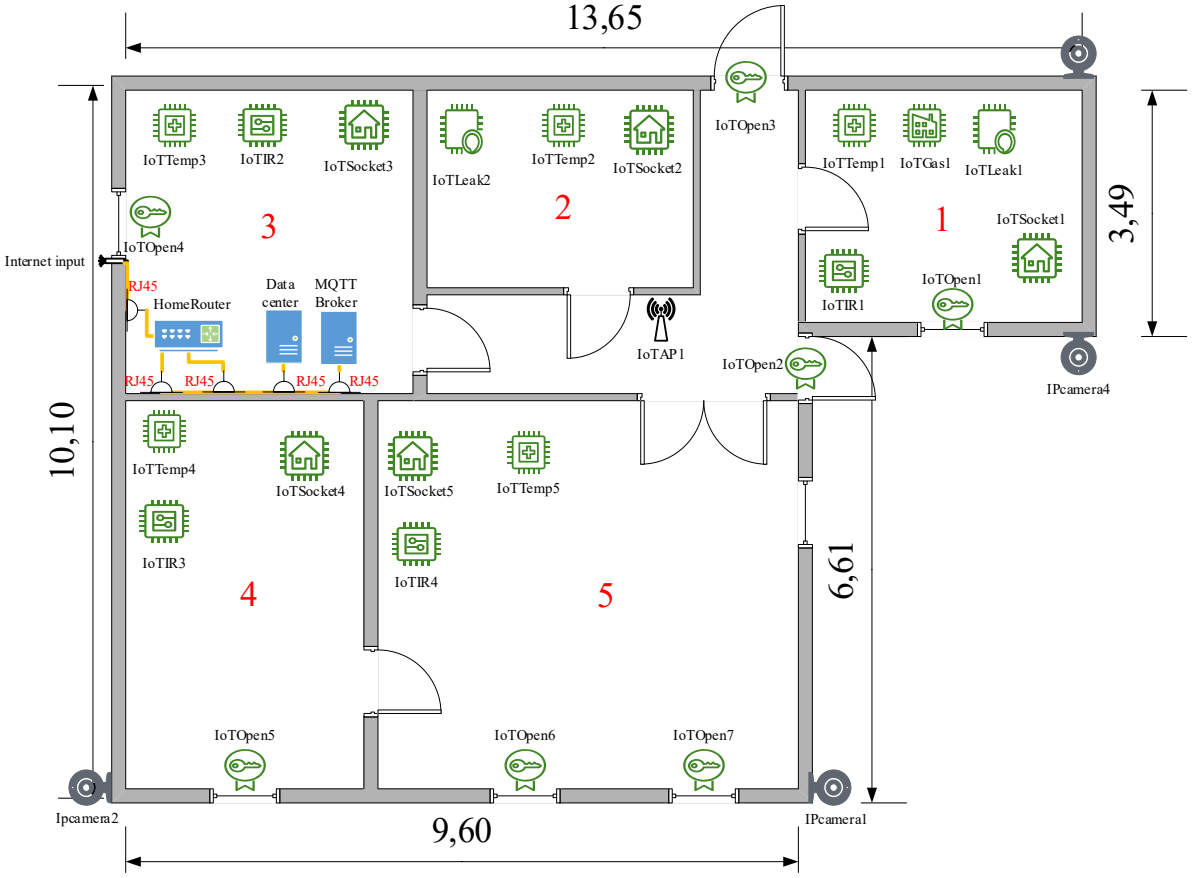

Рисунок 2.1 – План будинку і розміщенням IoT елементів

Опишемо приміщення будинку:

- Приміщення 1 – використовується для приготування та споживання їжі. Містить вузли підключення води та газу. Має одне вікно, що відкривається. Є електроприлади, зокрема які керуються ІЧ-командами.
- Приміщення 2 – ванна кімната. Має вузли підключення води. В ній розташований лічильник. Має електроприлади.

- Приміщення 3 – серверна. Має одне вікно, що відкривається. Є електроприлади, зокрема які керуються ІЧ-командами.
- Приміщення 4 – спальна кімната. Має одне вікно, що відкривається. Є електроприлади, зокрема які керуються ІЧ-командами.
- Приміщення 5 – гостинна кімната. Має три вікна, два з яких відчиняються. Є електроприлади, зокрема які керуються ІЧ-командами.

2.2 Проектування функціонального набору IoT системи об’єкта «Розумний будинок»

Опишемо пристрої з рис. 2.1:

1) Пристрій фіксації відкриття – відповідає за посилання сигналу на сервер при відкритті вузлу, на якому встановлений (двері, вікно, тощо). Для зручності, повинен бути безпровідним і мати живлення від акумулятору. Складається з:

- Датчик відкриття. Має бути надійним та споживати якнайменше електроенергії. Підключення має здійснюватися через цифровий порт.
- Модуль зчитування напруги. Відповідає за надання даних про напругу на акумуляторі. Виходячи з цих даних, можна дізнатися відсоток заряду акумулятора. Підключення повинне проводитися по шині I2C.
- Мікроконтролер. Відповідає за встановлення зв’язку та передавання показників з приладу. Має містити модуль WiFi, точку доступу WiFi, мінімум 3 цифрових порти (2 для підключення шини I2C та ще один для підключення датчику) та мати режим «глибокого сну». Відповідно, має містити контакт, при подачі сигналу на який, мікросхема має виходити з режиму «глибокого сну». Має споживати якнайменше електроенергії.
- Акумулятор. Має мати ємкість не менш ніж 2000 мА.
- Модуль захисту акумулятору. На вхід має підключатися акумулятор зі змінною напругою (в залежності від заряду акумулятору вона буде

змінюватися). Цей модуль повинен захистити акумулятор від перерозряду, таким чином продовжуючи термін його роботи.

- DC–DC підвищуючий перетворювач. На вхід підключається модуль захисту акумулятору, а на виході, незалежно від заряду акумулятору, повинна бути напруга 5 V.

Для зручності, позначається як «IoTOpen».

2) Пристрій фіксації витіку газу – відповідає за подання сигналу на сервер у випадку виявлення витіку газу. Для зручності, повинен бути безпровідним і мати живлення від акумулятору. Складається з:

- Датчик газу. Має бути надійним та споживати якнайменше електроенергії. Повинен мати аналогове підключення та цифрове підключення.
- Модуль зчитування напруги. Відповідає за надання даних про напругу на акумуляторі. Виходячи з цих даних, можна дізнатися відсоток заряду акумулятора. Підключення повинне проводитися по шині I2C.
- Мікроконтролер. Відповідає за встановлення зв’язку та передавання показників з приладу. Повинен мати режим «глибокого сну». Має містити модуль WiFi, точку доступу WiFi, мінімум 3 цифрових порти (2 для підключення шини I2C та 1 для виходу мікроконтролеру з режиму «глибокого сну») та 1 аналоговий порт для підключення датчику. Має споживати якнайменше електроенергії.
- Акумулятор. Має мати ємкість не менш ніж 2000 мА.
- Модуль захисту акумулятору. На вхід має підключатися акумулятор зі змінною напругою (в залежності від заряду акумулятору вона буде змінюватися). Цей модуль повинен захистити акумулятор від перерозряду, таким чином продовжуючи термін його роботи.
- DC–DC підвищуючий перетворювач. На вхід підключається модуль захисту акумулятору, а на виході, незалежно від заряду акумулятору, повинна бути напруга 5 V.

Для зручності, позначаєтья як «IoTGas».

3) Пристрій фіксації протікання – відповідає за подання сигналу на сервер у випадку виявлення протікання. Для зручності, повинен бути безпровідним і мати живлення від акумулятору. Складається з:

- Датчик протікання. Має бути надійним та споживати якнайменше електроенергії. Повинен мати аналогове підключення.
- Модуль зчитування напруги. Відповідає за надання даних про напругу на акумуляторі. Виходячи з цих даних, можна дізнатися відсоток заряду акумулятора. Підключення повинне проводитися по шині I2C.
- Мікроконтролер. Відповідає за встановлення зв'язку та передавання показників з приладу. Має містити модуль WiFi, точку доступу WiFi, мінімум 2 цифрових порти (для підключення шини I2C) та один аналоговий для підключення датчику. Повинен мати режим «глибокого сну». Має споживати якнайменше електроенергії.
- Акумулятор. Має мати ємкість не менш ніж 2000 мА.
- Модуль захисту акумулятору. На вхід має підключатися акумулятор зі змінною напругою (в залежності від заряду акумулятору вона буде змінюватися). Цей модуль повинен захистити акумулятор від перерозряду, таким чином продовжуючи термін його роботи.
- DC–DC підвищуючий перетворювач. На вхід підключається модуль захисту акумулятору, а на виході, незалежно від заряду акумулятору, повинна бути напруга 5 V.

Для зручності, позначається як «IoTLeak».

4) Пристрій фіксації температури – відповідає за передання показників температури на сервер. Для зручності, повинен бути безпровідним і мати живлення від акумулятору. Складається з:

- Датчик температури. Має точно вимірювати показники температури і споживати при цьому якнайменше електроенергії. Підключення має бути аналоговим.

- Модуль зчитування напруги. Відповідає за надання даних про напругу на акумуляторі. Виходячи з цих даних, можна дізнатися відсоток заряду акумулятора. Підключення повинне проводитися по шині I2C.
- Мікроконтролер. Відповідає за встановлення зв’язку та передавання показників з приладу. Має містити модуль WiFi, точку доступу WiFi, мінімум 2 цифрових порти (для підключення шини I2C) та один аналоговий для підключення датчику. Повинен мати режим «глибокого сну». Має споживати якнайменше електроенергії.
- Акумулятор. Має мати ємкість не менш ніж 2000 мА.
- Модуль захисту акумулятору. На вхід має підключатися акумулятор зі змінною напругою (в залежності від заряду акумулятору вона буде змінюватися). Цей модуль повинен захистити акумулятор від перерозряду, таким чином продовжуючи термін його роботи.
- DC–DC підвищуючий перетворювач. На вхід підключається модуль захисту акумулятору, а на виході, незалежно від заряду акумулятору, повинна бути напруга 5 V.

Для зручності, позначається як «IoTTemp».

5) Пристрій керування ІЧ-пристроями – відповідає за подання сигналу на ІЧ-пристрої при надходженні команди з серверу. Також, має повторювати ІЧ-команди, які надійшли від інших джерел. Для швидкого опрацювання команд, повинен бути провідним та мати живлення 220 Вольт змінного струму. Складається з:

- ІЧ-приймач. Відповідає за прийом ІЧ-команд. Аналогове підключення.
- ІЧ-відправник. Відповідає за відправлення ІЧ-команд. Аналогове підключення.
- Мікроконтролер. Відповідає за встановлення зв’язку та передавання інформації з приладу. Має містити модуль WiFi, точку доступу WiFi та не менше ніж 2 аналогових порти для підключення модулів, які описані вище.

- Перетворювач напруги з 220 Вольт змінного струму на 5 Вольт постійного струму. На вхід має підключатись джерело живлення 220 Вольт змінного струму, а на виході має бути 5 Вольт постійного струму. Має забезпечувати потужність не менше 1,5 Ватт.

  Для зручності, позначається як «IoTIR».

6) Пристрій дистанційного керування електроприладами – відповідає за подачу струму до підключеного до нього пристрою. Має мати живлення від мережі 220 Вольт змінного струму. Складається з:

- Реле. Відповідає за переключання стану підключеного пристрою в залежності від команди з мікроконтролеру. Цифрове підключення.
- Мікроконтролер. Відповідає за встановлення зв'язку з сервером та керування реле. Має містити модуль WiFi, точку доступу WiFi та не менше ніж 1 цифровий порт для підключення реле.
- Перетворювач напруги з 220 Вольт змінного струму на 5 Вольт постійного струму. На вхід має підключатись джерело живлення 220 Вольт змінного струму, а на виході має бути 5 Вольт постійного струму. Має забезпечувати потужність не менше 1,5 Ватт.

  Для зручності, позначається як «IoTSocket».

7) Камера спостереження – відповідає за потокову передачу відео на сервер. Повинна передавати інформацію по WiFi та мати живлення від мережі 220 Вольт змінного струму. Має бути готовим рішенням (тобто вироблена не власноруч) та підтримувати протокол передачі даних RTCP. Для зручності, позначається як «IPcamera».
8) HomeRouter – маршрутизатор із вбудованою точкою доступу. Має з'єднувати усі пристрої в одну мережу. Має містити можливості виділення статичних локальних ip-адрес, підключення не менше ніж 30 пристроїв а також підключення сторонніх точок доступу, які розширюватимуть покриття існуючої мережі.
9) Data center – сервер, який оброблює відео з усіх відеокамер. Також, на ньому запущена головна програма, яка керує системою, та система

штучного інтелекту, яка видає рекомендації щодо автоматизації сценаріїв користувача.

10) MQTT broker – сервер MQTT. Має мати достатньо обчислювальної потужності для поставлених задач.

11) Точка доступу – має підключатись до існуючої мережі маршрутизатору та розширювати її покриття.

Побудуємо схему підключення пристроїв, зображену на рис. 2.2.

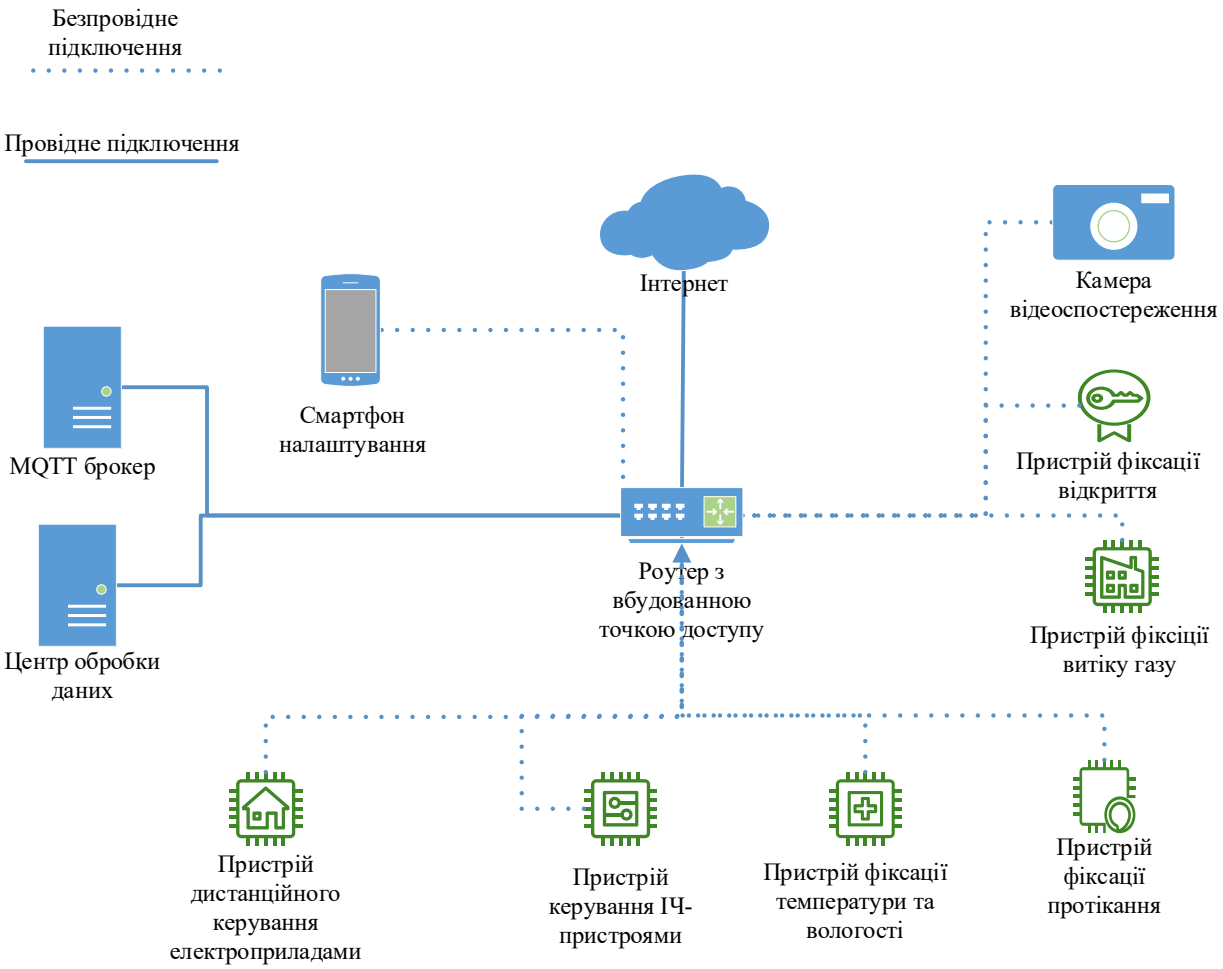

Рисунок 2.2 – Схема підключення пристроїв

2.3 Проектування функцій IoT системи

Побудуємо алгоритм функціювання мікроконтролерів пристроїв. По-перше, слід виділити фрагмент блок схеми, який для всіх пристроїв є однаковим. На рис. 2.3 зображений фрагмент блок-схеми, який відповідає за отримання мікроконтролером даних для підключення до мережі. Мікроконтролер виступає у ролі точки доступу, очікуючи підключення. Як тільки це трапиться, почнеться отримання даних, які мікроконтролер перевірить та запише у енергонезалежну пам’ять. Для подальшої зручності, позначимо цей фрагмент як «Отримання даних TCP»

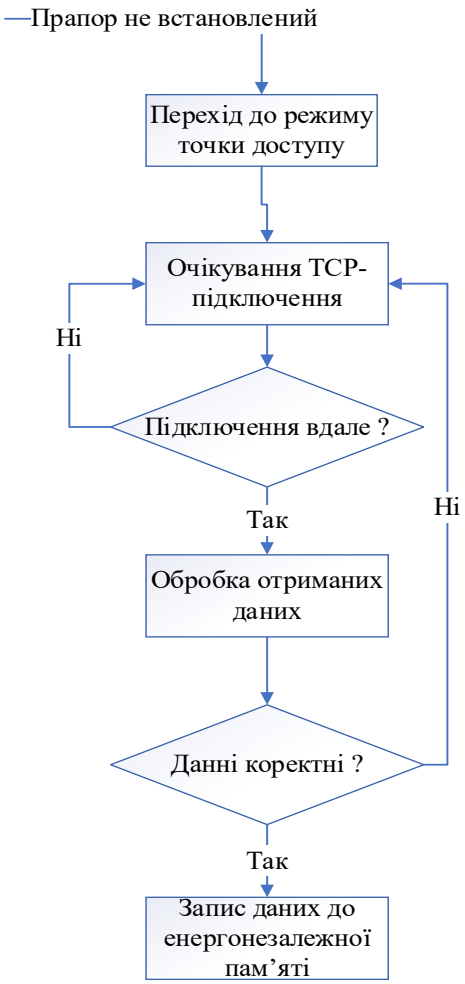

Рисунок 2.3 – Фрагмент блок-схеми для блоку отримання даних по TCP-каналу

Розглянемо принцип роботи пристрою «IoTTemp» за допомогою блок-схеми, зображеній на рис. 2.4.

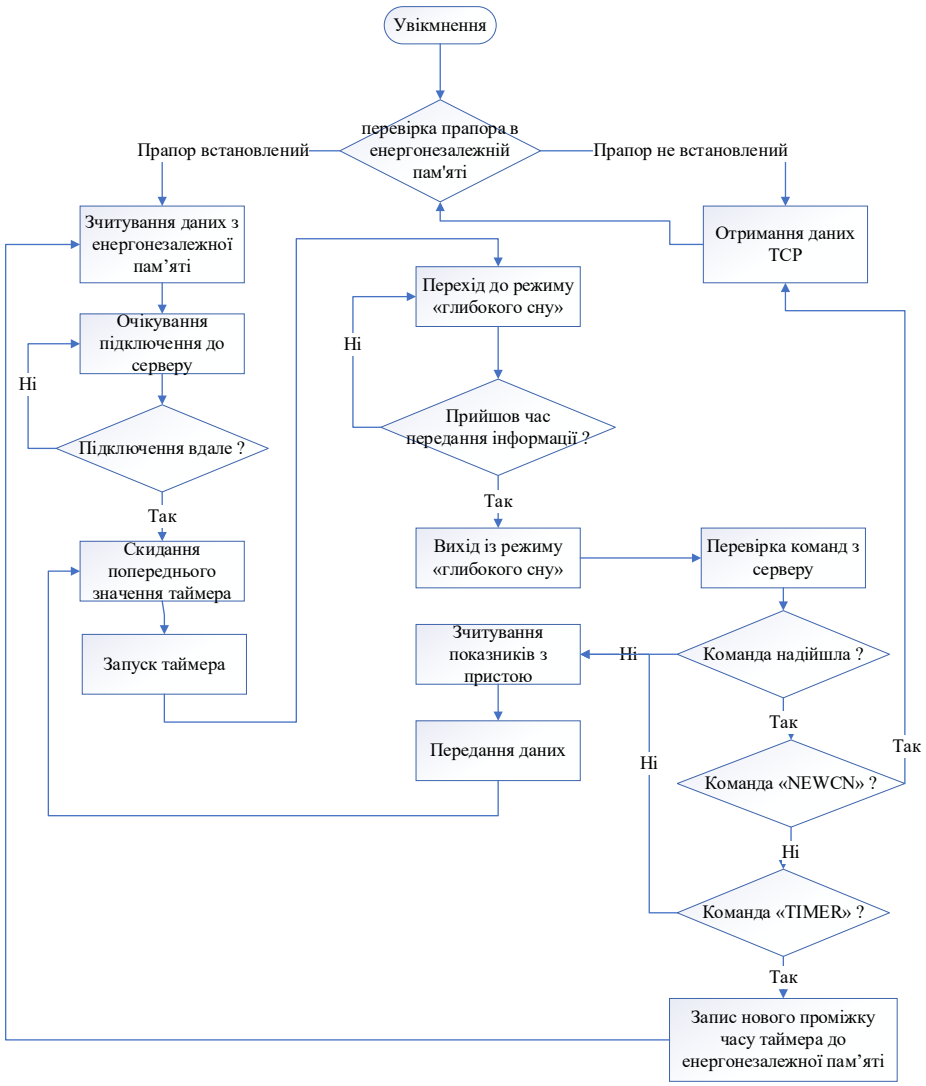

Рисунок 2.4 – Блок-схема роботи пристрою «IoTTemp»

При першому включенні або після перезавантаження перевіряється прапор в енергонезалежній пам’яті, якій відповідає за відстеження першого запуску мікроконтролеру. Якщо це дійсно перше включення, пристрій переходить у режим точки доступу та очікує дані для підключення. Після вдалого підключення, мікроконтролер переходить у режим «глибокого сну» для збереження заряду акумулятору. Після прокидання, мікроконтролер

зчитує дані для підключення з енергонезалежної пам’яті та перевіряє, чи не надійшла команда з серверу. Підтримуються 2 команди: TIMER(приходить у форматі «TIMER» «числове значення») та NEWCN. Командою TIMER задається період «глибокого сну» мікроконтролеру у секундах. Команда NEWCN переводить мікроконтролер у режим точки доступу для очікування нового підключення.

Тепер, розглянимо блок-схему пристрою «IoTGas», яка зображена на рис. 2.5. Варто відзначити, що дана блок-схема застосовна і для пристроїв «IoTOpen» та «IoTLeak».

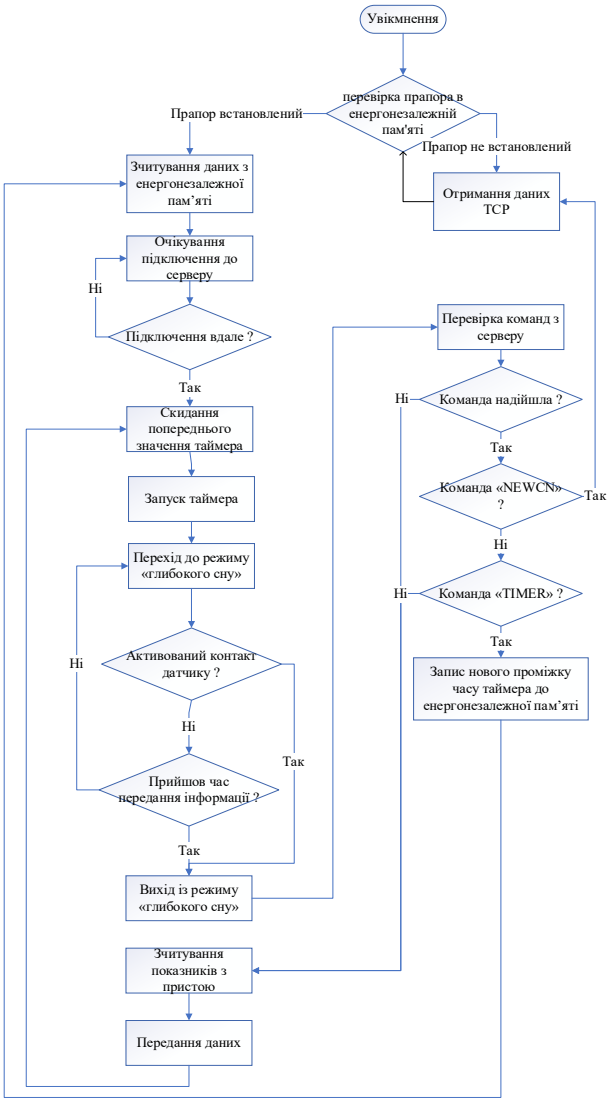

Рисунок 2.5 – Блок схема роботи пристрою «IoTGas»

Ця блок-схема в значній мірі схожа з попередньою за виключенням одного моменту – активація мікроконтролеру за сигналом з датчику. Якщо сигнал з датчику не приходить – з заданою періодичністю відправляються дані про процент заряду акумулятору.

Далі, розглянемо роботу пристрою «IoTIR», продемонстрованої на блок-схемі на рис. 2.6.

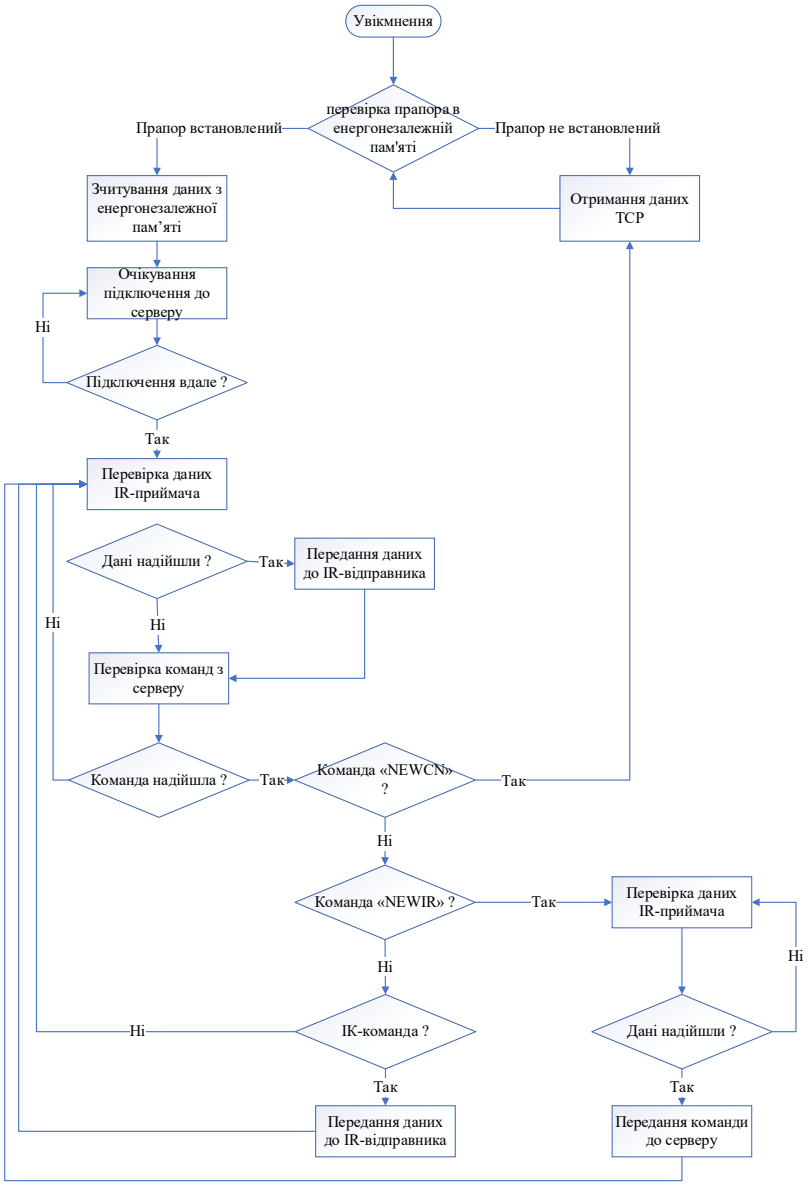

Рисунок 2.6 – Блок-схема роботи пристрою «IoTIR»

Даний пристрій приймає три види команд: вже знайома NEWCN, NEWIR та ІК-команда. При отриманні команди NEWIR, пристрій переходить у режим очікування ІК-команди. При отриманні, відправляє її до серверу. Якщо ж пристрій отримав дані у форматі ІК-команди, то ця команда відправляється до ІК-світлодіоду.

Заключним пристроєм у розгляданні є «IoTSocket». Даний пристрій являє собою кероване реле, яке може перебувати у двох положеннях - включено та відключено, в залежності від від команди, що прийшла (1 або 0 відповідно). Блок-схема представлена на рис. 2.7.

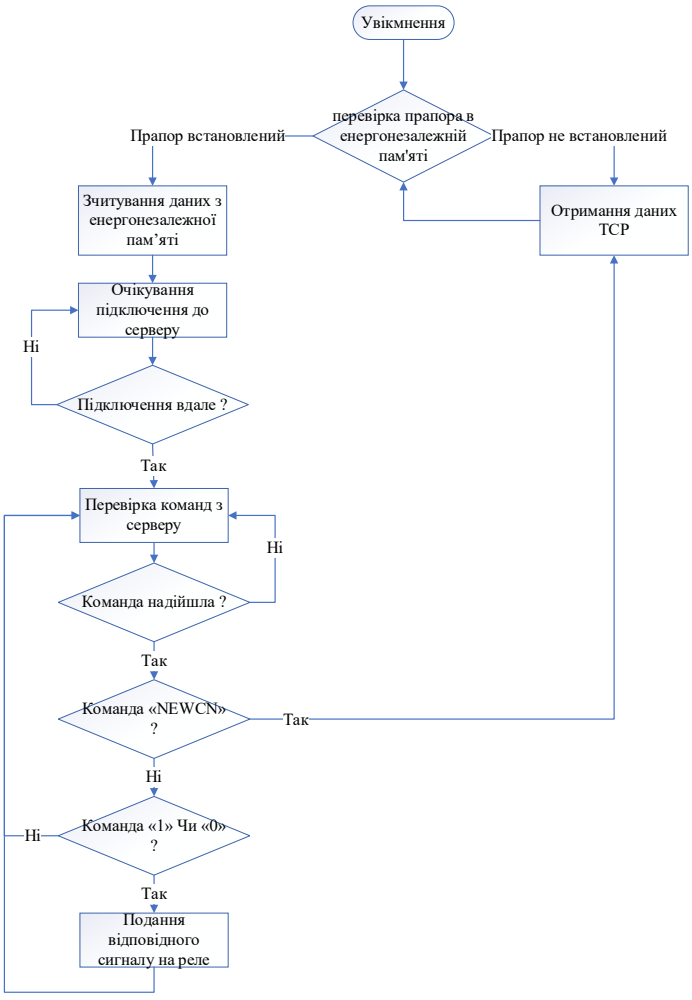

Рисунок 2.7 – Блок-схема роботи пристрою «IoTSocket»

В таблиці 2.1 приведено опис функціоналу пристроїв.

Таблиця 2.1 – Опис функціоналу пристроїв

Назва	Позначення	Функції
Пристрій фіксації відкриття	IoTOpen	При першому з’єднанні виступає у ролі точки доступу та отримує дані по протоколу TCP для підключення до WiFi-мережі та MQTT-брокеру. Записує ці дані до енергонезалежної пам’яті. При отриманні команди NEWCN знову переходить у режим точки доступу. При отриманні команди TIMER встановлює нове значення таймеру. Кожен заданий проміжок часу відсилає дані про процент заряду акумулятора до MQTT-брокера. При спрацьовуванні датчика відкриття посилає сигнал до MQTT-брокера. Див. рис….
Пристрій фіксації витіку газу	IoTGas	При першому з’єднанні виступає у ролі точки доступу та отримує дані по протоколу TCP для підключення до WiFi-мережі та MQTT-брокеру. Записує ці дані до енергонезалежної пам’яті. При отриманні команди NEWCN знову переходить у режим точки доступу. При отриманні команди TIMER встановлює нове значення таймеру. Кожен заданий проміжок часу відсилає дані про процент заряду акумулятора до MQTT-брокера. При спрацьовуванні датчика витіку газу посилає сигнал до MQTT-брокера.

Продовження таблиці 2.1

Назва	Позначення	Функції
Пристрій фіксації протікання	IoTLeak	При першому з’єднанні виступає у ролі точки доступу та отримує дані по протоколу TCP для підключення до WiFi-мережі та MQTT-брокеру. Записує ці дані до енергонезалежної пам’яті. При отриманні команди NEWCN знову переходить у режим точки доступу. При отриманні команди TIMER встановлює нове значення таймеру. Кожен заданий проміжок часу відсилає дані про процент заряду акумулятора до MQTT-брокера. При спрацьовуванні датчика протікання посилає сигнал до MQTT-брокера. Увесь останній час перебуває у стані глибокого сну.
Пристрій фіксації температури та вологості	IoTTemp	При першому з’єднанні виступає у ролі точки доступу та отримує дані по протоколу TCP для підключення до WiFi-мережі та MQTT-брокеру. Записує ці дані до енергонезалежної пам’яті. При отриманні команди NEWCN знову переходить у режим точки доступу. При отриманні команди TIMER встановлює нове значення таймеру. Кожен заданий проміжок часу відсилає дані про процент заряду акумулятора та дані температури та вологості до MQTT-брокера.

Продовження таблиці 2.1

Назва	Позначення	Функції
Пристрій керування ІЧ-пристроями	IoTIR	При першому з’єднанні виступає у ролі точки доступу та отримує дані по протоколу TCP для підключення до WiFi-мережі та MQTT-брокеру. Записує ці дані до енергонезалежної пам’яті. При отриманні команди NEWNET знову переходить у режим точки доступу. При отриманні команди NEWIR очікує IR-команду та відправляє її до MQTT-брокеру. Увесь останній час працює у режимі повторювання IR-команд.
Пристрій дистанційного керування електроприладами	IoTSocket	При першому з’єднанні виступає у ролі точки доступу та отримує дані по протоколу TCP для підключення до WiFi-мережі та MQTT-брокеру. Записує ці дані до енергонезалежної пам’яті. При отриманні команди NEWCN знову переходить у режим точки доступу. При отриманні команди ON, замикає реле так, щоб електричний струм міг протікати. При отриманні команди OFF, замикає реле так, щоб електричний струм не міг протікати. У разі успіху команди, відправляє до MQTT-брокеру команду SUCCESS, в іншому випадку відправляє FAILED.

Продовження таблиці 2.1

Назва	Позначення	Функції
MQTT-брокер	MQTT broker	сервер, що координує повідомлення між різними клієнтами. Брокер відповідає за отримання повідомлень, ідентифікацію клієнтів, підписаних на кожне повідомлення, та надсилання їм повідомлень. Зберігає усю історію повідомлень та в окремому файлі IR-команди.
Сервер обробки даних	Data center	сервер, на якому запущено програми розпізнавання образів з відеокамер, а також запущено основну програму, яка відповідає за взаємодію системи з користувачем через мобільний додаток. Додатково, на ньому запущено програму, яка аналізує дії користувача та видає пропозиції щодо автоматизації.
Роутер з вбудованою точкою доступу	Home router	відповідає за з'єднання елементів системи в одну мережу. Також, надає доступ користувачеві до сервера обробки даних, який знаходиться в демілітаризованій зоні.

3 РОЗРОБКА ІОТ СИСТЕМИ ОБ'ЄКТА «РОЗУМНИЙ БУДИНОК»

3.1 Вибір обладнання IoT мережі

Оберемо обладнення, найбільш відповідне під умови з 2 розділу, а саме:

1) У якості мікроконтролеру для приладів «IoTOpen», «IoTGas», «IoTLeak», «IoTTemp» та «IoTIR» було обрано NodeMCU на базі ESP-32, так як менш потужніші мікроконтролери не впорались би с задачами пристроїв. Однак, для пристрою «IoTSocket» було обрано Wemos D1 mini на базі ESP-8266.
2) У якості MQTT-брокеру була обрана Rasberry Pi 3, опис якої знаходиться у першому розділі (див. розділ 1.4).
3) У якості акумулятору обрано акумуляторну батарею Samsung 18650, зображену на рис. 3.1. Вона має достатню ємкість та відзначається високою надійністю відносно інших виробників. Максимальне значення напруги – 4.2 V, мінімальне – 2.8 V.

Рисунок 3.1 – Акумуляторна батарея Samsung 18650

4) У якості захисту акумулятору від перерозряджання, було обрано модуль TP4056, зображений на рис. 3.2. Даний модуль не дозволяє батареї віддавати напругу нижче ніж 2.8 V. дозволяє пропускати через себе струм близько 1 ампера, чого з запасом вистачить.

Рисунок 3.2 – Модуль захисту акумулятору TP4056

5) Для виміру напруги на акумуляторі було обрано модуль на основі INA219, який зображено на рис. 3.3. Він призначений для вимірювання таких параметрів постійного струму як напруга, струм та споживана потужність. Модуль виконаний на мікросхемі INA219 – вимірювачем струму та напруги з нульовим дрейфом і має малими розмірами та вагою при дуже великих можливостях та високій точності вимірювань. Мікросхема вимірює параметри протікання струму у будь-якому напрямку з автоматичним перемиканням полярності виміру.

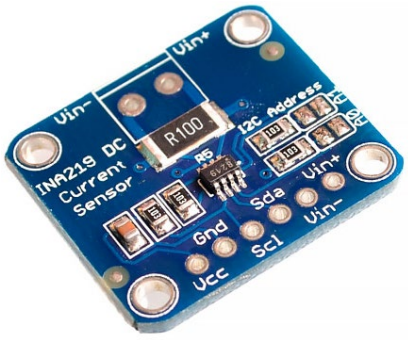

Рисунок 3.3 – Модуль виміру постійного струму на основі INA219

6) Для гарного живлення елементів пристрою необхідний підвищуючий DC-DC модуль, який на вхід приймає напругу від 2.8 до 4.2 V, а на виході – завжди 5 V. Зображення такого модулю на рис. 3.4.

Рисунок 3.4 – Підвищуючий DC-DC перетворювач

7) У якості AC-DC перетворювача був обраний HLK-PM1, зображений на рис. 3.5. Його характеристики:
   - Вхідна напруга: 100…240V (AC).
   - Вихідна напруга: 5V (DC).
   - Вихідний струм: 600 мА.
   - Діапазон робочих температур -20°C +60°C.
   - Вага: 18гр.
   - Розміри: 34x20x15 мм.

Рисунок 3.5 – AC-DC перетворювач HLK-PM1

8) У якості датчика температури виступає DHT-22, зображений на рис. 3.6. Приведемо характеристики:
   - Напруга живлення: 3 .. 6 V.

- Діапазон вимірювання температури: -40°C..+80°C.
- Похибка вимірювання температури: 0.5°C.
- Крок вимірювання температури: 0.1 °C.

Рисунок 3.6 – Датчик температури DHT-22

9) У якості інфрачервоного приймача виступає VS1838B, зображення якого показано на рис. 3.7. Характеристики:
   - Напруга живлення: 2.7 - 5.5 V.
   - Максимальна дальність дії: 20 м.
   - Споживаний струм: не більше 1.5 мА.
   - Кут огляду: 90 градусів.

Рисунок 3.7 – Інфрачервоний приймач VS1838B

10) У якості інфрачервоного передавача обраний модуль KY-005, зображений на рис. 3.8. При передачі даних ІЧ-канал не чутливий до електромагнітних перешкод радіодіапазону і не створює перешкод для

радіоприймачів. Застосування в оптопарах вигідно завдяки слабкій чутливості ІЧ фотоприймача до звичайного освітлення. Модуль ІЧ-світлодіодів KY-005 полегшує макетування схем і дозволяє зробити більш простим вивчення властивостей світлодіодів .

Рисунок 3.8 – Модуль KY-005

11) У якості датчика газу обрано MQ-5, показаний на рис. 3.9. Характеристики:

- Тип детектованого газу: зріджений вуглеводневий, метан, коксовий газ.
- Діапазон вимірювання: пропан: 200-10000 ppm, ізобутан: 200-10000 ppm, природний газ: 200-10000 ppm.
- Перевищення концентрації газу над середовищем: 5 разів.
- Робоча температура -10...50 °C.
- Напруга живлення: 5 Вольт.
- Вихідний сигнал High/Low та аналоговий.
- Габаритні розміри: 32x22x30 мм .

Рисунок 3.9 – Модуль MQ-5

12) Реле обрано моделі SLA-12VDC-SL-C, яке зображено на рис. 3.10. Це компактний модуль потужного реле з можливістю вибору рівня сигналу, що управляє. При невеликих розмірах модуль дозволяє комутувати навантаження зі струмом до 30А. При цьому для керування реле достатньо 5 мА струму сигналу. Для захисту входу та обмеження струму на платі встановлено струмообмежувальний резистор.

Рисунок 3.10 – Реле SLA-12VDC-SL-C

13) датчик протікання води обраний Т1592, зображений на рис. 3.11. Характеристики:

- Робоча напруга: DC 3-5V.
- Робочий струм: менше 20 мА.
- Тип датчика: аналоговий.
- Площа виявлення 40мм х 16мм.
- Робоча температура: 10 – 30 градусів.
- Вологість: 10%-90% без конденсації.

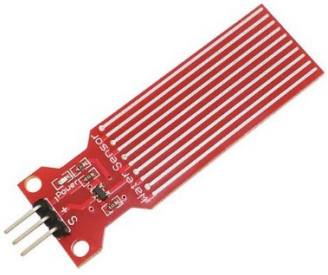

Рисунок 3.11 – Датчик протікання Т1592

14) Датчик відкриття SP-01, зображений на рис. 3.12, застосовується як елемент охоронної системи у разі спроби проникнення крізь двері, вікна і ворота. Під'єднується до будь-якого мікроконтролеру двома проводами. У разі спрацювання розмикає контакт. Не вимагає подавання живлення. Підходить для будь-яких поверхонь.

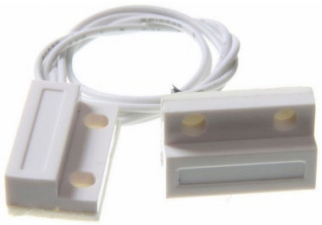

Рисунок 3.12 – Датчик відкриття SP-01

15) В якості маршрутизатору було обрано TP-Link Archer C80 [10], зображений на рис. 3.13. Характеристики:

- Частота роботи Wi-Fi: 5 ГГц + 2.4 ГГц (дводіапазонний).
- Швидкість LAN портів: 1 Гбіт/с.
- Інтерфейси: 1 x WAN порт 10/100/1000, 4 x LAN портів 10/100/1000.
- Швидкість Wi-Fi, Мбіт/с: 1900 Мбіт/сек.

Рисунок 3.13 – Маршрутизатор TP-Link Archer C80

16) У якості WiFi-камери обрано модель Tapo C320WS [11], зображена на рис. 3.14. Перевагою Tapo C320WS є те, що вона обладнана 4-мегапіксельною матрицею. Тому вона підходить практично для будь-якого місця встановлення. Корпус камери відповідає ступеню захисту IP66, що говорить про її повний захист від потрапляння всередину пилу, бруду та води, що дозволяє розміщувати камеру на вулиці. Гарантується робота при температурі від -20 до +45 °C при вологості 10-90%. За рахунок інфрачервоного підсвічування реалізовано функцію ведення спостереження в нічний час або в місцях без освітлення на відстань до 30 метрів.

Рисунок 3.14 – Tapo C320

3.2 Принципові схеми підключення пристроїв

Розглянемо схему підключення пристрою «IoTTemp», зображену на рис. 3.15.

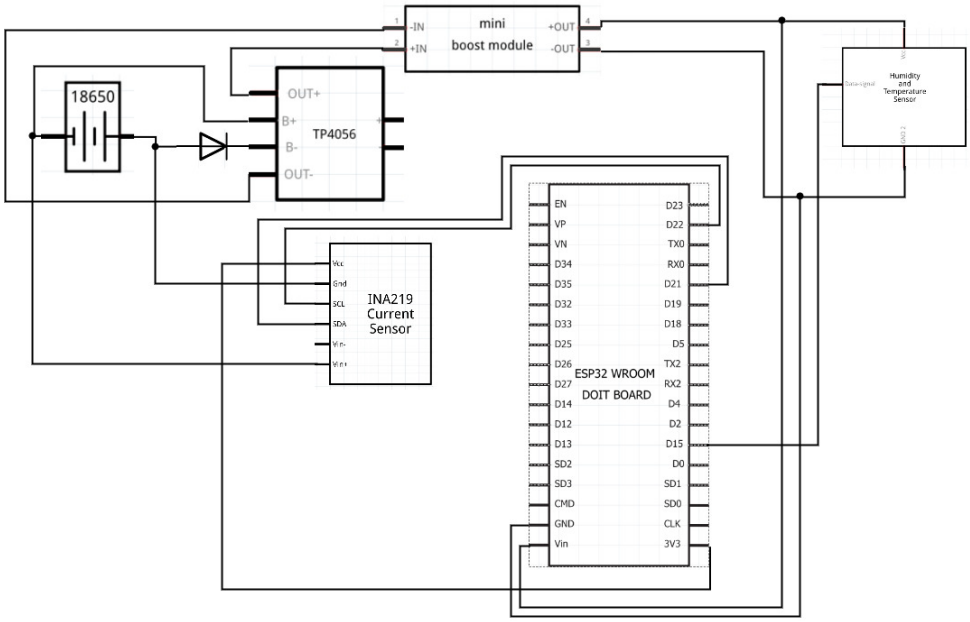

Рисунок 3.15 – Схема підключення «IoTTemp»

На цій схемі бачимо декілька цікавих моментів, які варті уваги. Наприклад, діод на вході у модуль TP4056 потрібен для захисту від переполюсовки. Але при цьому при проходженні через діод струм падає, що є проблемою для модулю INA219. Тому рішенням цієї проблеми стає підключення «мінусу» акумулятора до піну «GND» модуля INA219 перед діодом. Завдяки цьому даний модуль вимірює напругу точно. Інформацію передає через послідовну шину даних I2C. Також, на NodeMCU, окрім живлення, приходить ще підключення від датчику температури DHT22. Оскільки даний датчик є аналоговим, підключається він до аналогового порту D15.

Розглянемо схему підключення пристрою «IoTIR», зображену на рис. 3.16.

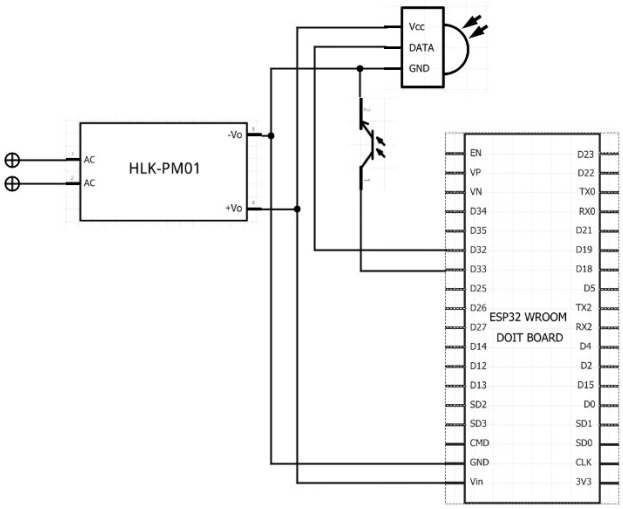

Рисунок 3.16 – Схема підключення «IoTIR»

На даній схемі ми можемо бачити AC-DC перетворювач HLK-PM1, на вхід якого подається струм 220 V AC. А на виході вже 5 V DC. Від нього йде живлення на NodeMCU та IR receiver. IR led бере живлення від піну, до якого підключений (D33). В свою чергу, IR receiver передає інформацію через пін D32.

Розглянемо схему підключення «IoTSocket» на рис. 3.17. Можемо бачити реле, яке керується сигналом цифрового порту D4 з Wemos D1 mini.

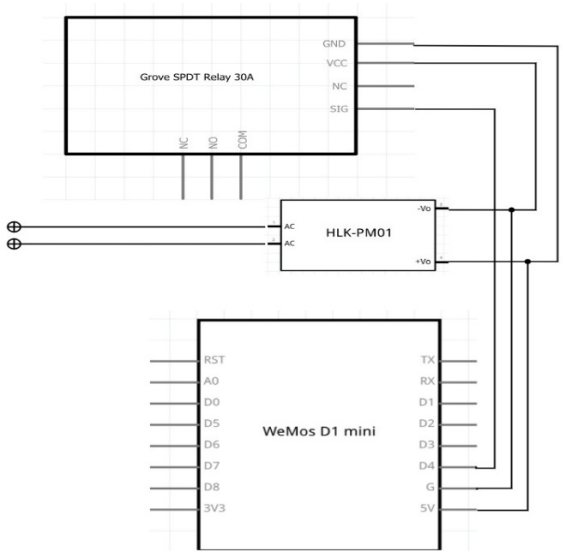

Рисунок 3.17 – Схема підключення «IoTSocket»

Розглянемо схему підключення пристрою «IoTGas», зображену на рис. 3.18. Бачимо схожу схему з «IoTTemp» (рис. 3.15). Варто звернути увагу на підключення датчику MQ-5, а саме на 2 лінії даних. Та, що підключена до піну D18 є цифровим сигналом. Якщо датчик зафіксує викид газу, він активує цю лінію даних. В той же час, лінія, що підключена до піну D34, являється аналоговою.

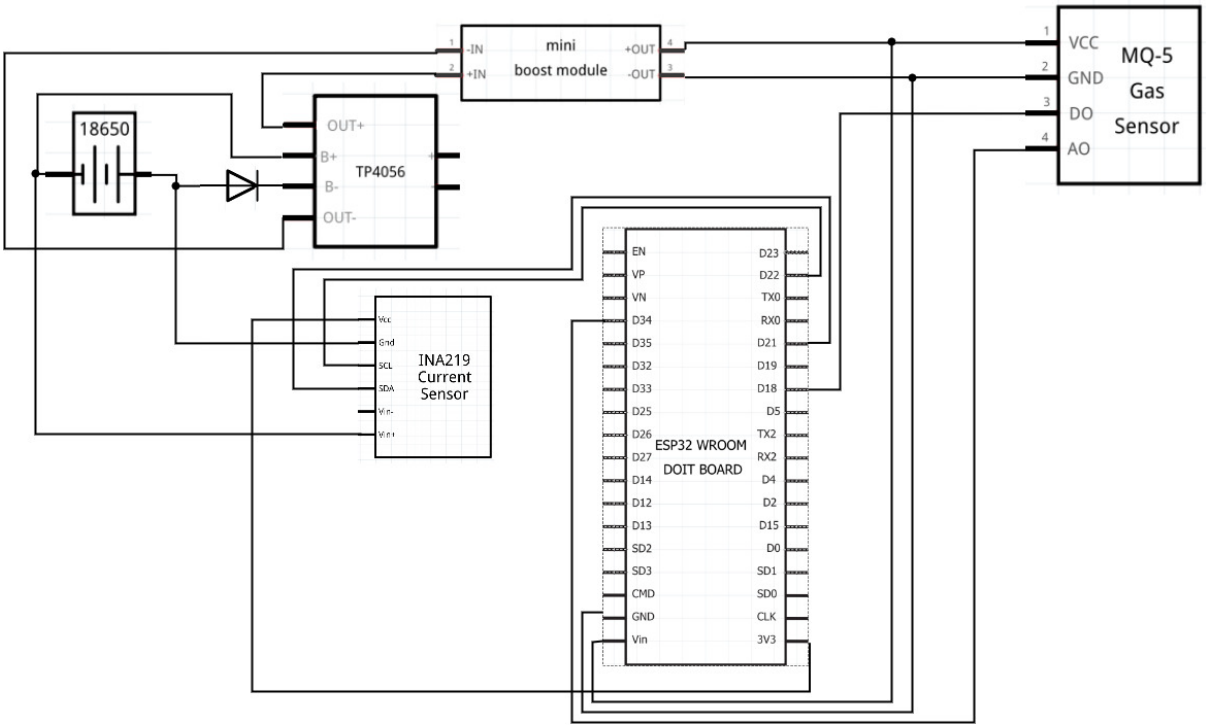

Рисунок 3.18 – Схема підключення «IoTGas»

Розглянемо схему підключення пристрою «IoTOpen», зображений на рис. 3.19. Варто відмітити підключення Open Sensor. Поки Open Sensor знаходиться у стані спокою, до піну D4 поступає сигнал, але як тільки Open Sensor змінить свій стан – сигнал пропадає. Саме на цю дію і реагує мікроконтролер.

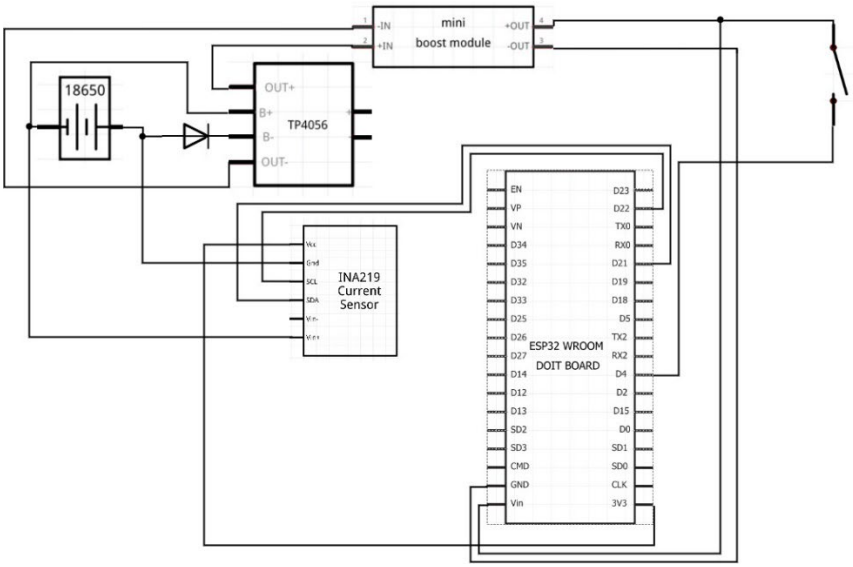

Рисунок 3.19 – Схема підключення «IoTOpen»

Розглянемо схему підключення пристрою «IoTLeak», зображений на рис. 3.20. Water Sensor, підключений до піну D34, являється аналоговим датчиком.

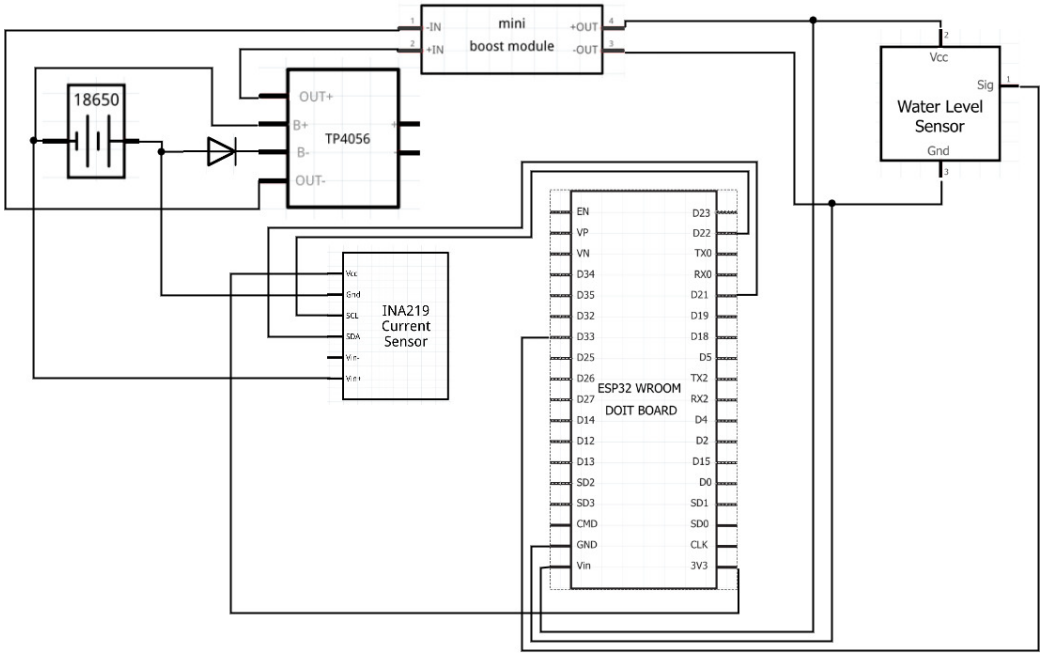

Рисунок 3.20 – Схема підключення пристрою «IoTLeak»

3.3 Розробка функцій безпечного підключення мікроконтролерів

Для розробки функцій мікроконтролерів було обрано мову C++ для мікроконтролерів сімейства ESP. При першому включені, мікроконтролер не має даних для підключення до Wi-Fi мережі. Для вирішення цієї проблеми, мікроконтролер запрограмований переходити до режиму точки доступу. Після цього, мікроконтролер запускає TCP-сервер для прийому даних. Для реалізації цього функціоналу перейдемо до налаштувань (рис. 3.21).

```
const char* password = "espespesp";
const uint16_t serverPort = 1337;
const u_int8_t chanel = 1;
const u_int8_t maxConnections = 1;
const u_int8_t hiden = 1;
IPAddress local_IP(192,168,100,1);
IPAddress gateway(192,168,100,1);
IPAddress subnet(255,255,255,0);
```

Рисунок 3.21 – Налаштування точки доступу для мікроконтролеру

Розглянемо кожен параметр:

- password – пароль для підключення до точки доступу;
- serverPort – порт TCP-серверу;
- chanel – канал WiFi, на якому запущена точка доступу;
- maxConnections – кількість користувачів, що одночасно підключаються:
- hiden – при встановленій 1, точка доступу скриває свою назву;
- local_IP – IP-адреса точки доступу;
- gateway – стандартний маршрут точки доступу;
- subnet – маска мережі точки доступу.

Після об’явлення налаштувань, переходимо до запуску точки доступу (рис. 3.22).

```
void setupAP(char* ssid)
{
  WiFi.mode(WIFI_AP);
  if (WiFi.softAP(ssid, password, chanel, hiden, maxConnections,
false)){
    Serial.println(«AP created!»);
  }else{
    Serial.println(«Error. AP not created!»);
  }
  if (WiFi.softAPConfig(local_IP, gateway, subnet)){
    Serial.println(«AP set up!»);
  }else{
    Serial.println(«Error! AP not set up!»);
  }
  delay(100);
  IPAddress apIP = WiFi.softAPIP();
  Serial.print(«AP IP address: «);
  Serial.println(apIP);
}
```

Рисунок 3.22 – Запуск точки доступу

В якості параметру, функції setupAP передається назва точки доступу. Після того, як точка доступу була успішно створена, в серійний порт виводиться її IP-адреса.

Наступний крок – очікування мікроконтролером з’єднання по TCP-каналу (рис. 3.23).

```
void getMessageTCP(WiFiClient& client, byte* payload){
    Serial.println("Start TCP...");
    server.begin();
    while(!client){
        client = server.available();
    }
    Serial.println("Client connected!");
    if(client.connected())
    {
        while(client.available())
        {
            client.readBytes(payload, MESSAGE_BUFFER);
        }
    }
    client.stop();
}
```

Рисунок 3.23 – Очікування даних по TCP-каналу

Оскільки TCP-з’єднання не є безпечним, дані при передачі шифруються зі сторони клієнта алгоритмом AES-256. Даний алгоритм є одним з найпоширеніших і надійних алгоритмів симетричного шифрування. Він використовує 256-бітові ключі для шифрування та дешифрування даних. Основні кроки алгоритму AES-256:

1) Ініціалізація ключа:
   - Алгоритм AES-256 вимагає 256-бітовий ключ, який повинен бути заздалегідь згенерований.
   - Ключ розбивається на 8 блоків по 32 біта кожний.
   - Кожний блок використовується для ініціалізації підключів, які потім використовуються в основних раундах шифрування.
2) Початковий раунд:
   - Вихідний текст (plaintext) розбивається на блоки розміром 128 біт.
   - Кожен блок проходить через операцію "AddRoundKey", де блок об'єднується з підключем за допомогою побітової операції XOR.
3) Основні раунди:
   - В AES-256 використовується 14 основних раундів шифрування.
   - Кожен раунд складається з чотирьох основних операцій: SubBytes, ShiftRows, MixColumns та AddRoundKey.
   - SubBytes: Замінює кожен байт блоку на відповідний байт з S-блока (замінна таблиця).
   - ShiftRows: Зсуває кожен рядок блоку на певну кількість байтів вліво.
   - MixColumns: Перетворює стовпці блоку, перемішуючи їх значення.
   - AddRoundKey: Блок об'єднується з підключем за допомогою XOR.
4) Заключний раунд:

Заключний раунд відрізняється від основних раундів тим, що він не включає операцію MixColumns.

5) Розшифрування:
   - Процес розшифрування AES-256 аналогічний процесу шифрування, але використовує зворотні операції.

- Кожен раунд починається з операції AddRoundKey, потім застосовуються зворотні операції SubBytes, ShiftRows та MixColumns (за винятком заключного раунду).
- Відновлюється розшифрований текст (plaintext).

Скористаємось бібліотекою mbedtls, в якій реалізовано даний алгоритм для розшифрування даних на стороні мікроконтролеру (рис. 3.24).

```
void AESdecrypt(const unsigned char* aesKey, char* message,
size_t messageLength, unsigned char* decryptedData){
  size_t encryptedDataLength;
  unsigned char decoded[messageLength];
  encryptedDataLength = b64_decode((char*)decoded, message,
messageLength);
  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_dec(&aes, aesKey, 256);
  for (size_t i = 0; i < encryptedDataLength; i += 16) {
    mbedtls_aes_crypt_ecb(&aes, ESP_AES_DECRYPT, decoded + i,
decryptedData+i);
  }
  Serial.println("Decode message:");
  for (size_t i = 0; i < encryptedDataLength; i++) {
    Serial.print((char)decryptedData[i]);
  }
  Serial.println();
  mbedtls_aes_free(&aes);
}
```

Рисунок 3.24 – Розшифрування даних, переданих TCP-клієнтом

Розглянемо що приймає у якості аргументів дана функція:

- aesKey – симетричний ключ шифрування, яким були зашифровані дані. Він передається на мікроконтролер разом із програмою;
- message – зашифровані дані;
- messageLength – довжина зашифрованих даних;
- decryptedData – посилання на змінну, в яку будуть записані розшифровані дані.

Розшифровані дані знаходяться у форматі JSON та мають наступний вигляд:

```
{
   "ssid": "********",
   "password": "********",
   "mqttBroker": ""********",
   "mqttPort": "********",
   "subscribeTopic": "********",
   "publishTopic": "********"
}
```

Наступний крок – записати ці дані в енергонезалежну пам’ять (рис. 3.25).

```
void putEEPROMinfo(){
  EEPROM.begin(MEMORY_SIZE);
  EEPROM.write(INIT_ADDR, INIT_KEY);
  EEPROM.put(DATA_ADDR, infoLocal);
  EEPROM.commit();
```

Рисунок 3.25 – Запис даних до енергонезалежної пам’яті мікроконтролера.

Розглянемо детальніше:

- MEMORY_SIZE – число, яке характеризує скільки пам’яті зарезервовано під нові дані;
- INIT_ADDR – число яке характеризує місце в енергонезалежній пам’яті, де буде зберігатися ключ ініціалізації.
- DATA_ADDR – число, яке характеризує початок місця, де будуть зберігатися дані для підключення до мережі.
- infoLocal – структура, в якій зберігаються дані для підключення до мережі.

Розглянемо режим «глибокого сну» для ефективного управління споживанням енергії. Основні аспекти режиму глибокого сну:

- Знижене споживання енергії: Режим «глибокого сну» дозволяє мікроконтролеру входити в стан низького споживання енергії. У цьому

режимі багато функціональних блоків, таких як ядро CPU, пам'ять, периферійні пристрої, вимикаються або переводяться в сплячий режим.

- Вибір режиму «глибокого сну»: існує декілька рівнів «глибокого сну» з різними режимами вимкнених компонентів. Наприклад, режим "Modem Sleep" вимикає Wi-Fi модуль, але залишає системний таймер активним. Режим «Light Sleep» вимикає CPU, але зберігає периферійні пристрої.
- Зовнішній спусковий механізм: Для виходу з режиму «глибокого сну» потрібна зовнішня подія, така як сигнал з GPIO піна або таймер. Коли така подія відбувається, мікроконтролер прокидається і продовжує свою роботу.

Розглянемо вихід мікроконтролера із стану «глубокого сну» по таймеру. Для початку, увесь код мікроконтролера повинен бути написаний у функції Setup(). При старті, мікроконтролер виконує код з функції Setup() першим, а потім вже переходить до функції Loop(), яка є безкінечним циклом та вважається головною. На початку функції Setup() пишемо наступну команду: esp_sleep_enable_timer_wakeup(TIMER * uS_TO_S_FACTOR), де TIMER – це кількість секунд, яку мікроконтролер повинен провести у режимі «глибокого сну». Тепер, коли виконуюча програма дійде до команди esp_deep_sleep_start(), мікроконтролер перейде у режим «глибокого сну».

Інші функції мікроконтролерів пристроїв наведено у ДОДАТКУ А.

3.4 Розробка керуючої програми

Для створення керуючої програми було обрано мову програмування Python. Вона виконує функції надсилання, прийому та зберігання даних від пристроїв. Через цю програму мобільний додаток керує приладами в мережі. Розглянемо основні моменти програми. Розпочнемо з функції додавання нового приладу. Для цього, потрібно декілька файлів:

- Файл «device_list.csv». Містить назви усіх пристроїв та відповідний їм тип (IoTTemp, IoTSocket, тощо).
- Файл «typeDevice.csv». Містить типи пристроїв та відповідну їм категорію (пристрій збору даних або керуючий пристрій).

Розглянемо код, зображений на рис. 3.26.

```
def addDevice(nameDevice, typeDevice):
    invalid_chars = '/\\:*?"<>|'
    for i_char in invalid_chars:
        if i_char in nameDevice:
            print("Error in addDevice. Invalid sign cannot be
present in the device name")
            return 1
    types = pd.read_csv("devices/typeDevice.csv")
    devices = pd.read_csv("devices/device_list.csv")
    if typeDevice not in types.type.values:
        print("Error in addDevice. Can`t find type device")
        return 2
    if nameDevice in devices.name.values:
        print("Error in addDevice. A device with the same name
already exists")
        return 3
    with open("devices/device_list.csv", "a", newline='') as
csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([nameDevice, typeDevice])
    path = f"devices/{nameDevice}"
    os.mkdir(path)
    with open(f"{path}/history.csv", "w", newline='') as
csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow(header_csv[typeDevice])
    category = categoryDevice(nameDevice)
    if category == "data":
        with open(f"{path}/settings.txt", "w") as file:
            file.write(f"TIMER:{DEFAULT_TIMER}")
    print("device added successfully")
    return 0
```

Рисунок 3.26 – Функція додавання пристрою

В якості аргументів приймаються назва пристрою та його тип. В цій функції використовуються декілька перевірок. Перша – це перевірка на недопустимі символи в назві пристрою. Оскільки буде створюватися каталог

в якості назви якого буде назва пристрою, в ній не повинно бути символів, недопустимих для створення каталогу. Далі йде перевірка на правильність вхідних даних. Якщо назва пристрою із аргумента вже існує – функція інформує про це і припиняє виконання. Те саме відбувається, якщо отримано тип даних, інформації про який немає у відповідному файлі. При успішності перевірок, дані записуються у файл «device_list.csv». Також, створюється каталог з назвою пристрою, в якому створюється файл «history.csv», в якому будуть зберігатися дані, отримані або з пристрою. Якщо пристрій категорії збору даних, додатково створюється файл «settings.txt», в якому зберігається налаштування пристрою. За замовчуванням, в нього записується налаштування відправки даних раз у хвилину. Налаштування можна змінювати. Якщо пристрій додано успішно, про це повідомляється та функція завершується.

Розглянемо функцію створення нового сценарію. Для цього, потрібен файл «scenarios.csv», який містить у собі:

- Назву сценарію;
- пристрій, значення якого активує сценарій;
- знак порівняння значення (більше, менше або дорівнює);
- порівнювальне значення;
- пристрій, на який буде відправлене значення у випадку виконання сценарію;
- значення, яке буде відправлене на пристрій у випадку виконання сценарію.

Сценарії дозволяють автоматично керувати пристроями залежно від умови. Підтримується відправка команди по часу. Розглянемо код, зображений на рис. 3.27.

Йдуть перевірки кожного переданого аргументу. Зокрема, чи вже існує назва сценарію та чи є вже такий сценарій, лише із іншою назвою. Якщо всі перевірки успішно пройдені, новий сценарій записується у відповідний файл,

повідомляється про успішне створення нового сценарію та функція завершується.

```
def addScenario(nameScenario,
triggerDevice,triggerValue,operation, slaveDevice, valueToSend):
    devices = pd.read_csv("devices/device_list.csv")
    if triggerDevice not in devices.name.values and
triggerDevice != "time":
        print(f"Unable to create script. Device {triggerDevice}
not found")
        return 1
    elif slaveDevice not in devices.name.values:
        print(f"Unable to create script. Device {slaveDevice}
not found")
        return 2
    if categoryDevice(slaveDevice) != "actuator":
        print("Unable to create script. the slave device must be
actuator")
        return 3
    if operation not in [">", "<", "="]:
        print("invalid operation type")
        return 4
    compareArr = [triggerDevice,operation, str(triggerValue),
slaveDevice, str(valueToSend)]
    with open("devices/scenarios.csv", 'r') as csvFile:
        csvData = csvFile.read()
        csvArr = csvData.split('\n')
    for line in csvArr:
        if nameScenario == line.split(",")[0]:
            print(f"script with name {nameScenario} already
exists")
            return 5
        if compareArr == line.split(",")[1:]:
            print("script with the same set exists")
            return 6
    with open("devices/scenarios.csv", 'a') as csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([nameScenario, triggerDevice, operation,
triggerValue, slaveDevice, valueToSend])
    print("script created successfully")
    return 0
```

Рисунок 3.27 – Функція додавання сценарію

Розглянемо функцію перевірки сценарію на рис. 3.28. Вона виконується кожного разу, коли пристрій відправляє дані або, у випадку з керуючими пристроями, коли пристрій приймає команду.

```
def checkScenario(nameDevice, valueDevice, MQTTclient):
     data = pd.read_csv("devices/scenarios.csv")
    if categoryDevice(nameDevice) == "data" and
len(str(valueDevice)) > 10:
        jsonLoads = json.loads(valueDevice)
        valueDevice = int(jsonLoads['value'])
    for i in range(len(data)):
        lineData = data.iloc[i]
        if lineData.triggerDevice == nameDevice:
            type = typeDevice(lineData.slaveDevice)
            match lineData.operation:
                case '=':
                    if lineData.triggerValue == valueDevice:
                        print("scenario activated")
                        print(f"Data {lineData.valueToSend} send
to  {lineData.slaveDevice} ")
                        sendFunction[type](lineData.valueToSend,
lineData.slaveDevice, type, MQTTclient)
                case '>':
                    if nameDevice == "time":
                        return
                    if valueDevice > int(lineData.triggerValue):
                        print("scenario activated")
                        print(data.iloc[i])
                        sendFunction[type](lineData.valueToSend,
lineData.slaveDevice, type, MQTTclient)
                case '<':
                    if nameDevice == "time":
                        return
                    if  valueDevice <
int(lineData.triggerValue):
                        print("scenario activated")
                        print(data.iloc[i])
                        sendFunction[type](lineData.valueToSend,
lineData.slaveDevice, type, MQTTclient)
    return None
```

Рисунок 3.28 – Функція перевірки даних для виконання сценарію

Розглянемо перевірки. Якщо пристрій, для якого перевіряється участь у сценарії, належить до категорії збору даних, то передані в функцію дані пристрою оброблюються таким чином, щоб отримати саме показники датчика.

Це потрібно оскільки повідомлення з пристрою містить у собі ще й інші дані, такі як час відправки та процент заряду батареї. Після цього, здійснюється пошук сценарію, якій містить в собі передану назву пристрою. Сама назва пристрою не перевіряється, оскільки дана функція викликається лише іншими функціями програми, які і містять відповідну перевірку.

Важливо зауважити, що назва пристрою, передана в якості аргумента, порівнюється лише з назвою пристрою, значення якого активує сценарій. Якщо такий сценарій знайдено, здійснюється перевірка виконання сценарію. Якщо перевірка виконується, дані відправляються відповідно до сценарію.

Розглянемо функцію приймання даних, зображену на рис. 3.29.

```
    def on_message(client, userdata, msg):
        message = msg.payload.decode()
        typeDevice = msg.topic.split(':')[0]
        device = msg.topic.split(':')[1].split('/')[0]
        print(f"newMessage from {msg.topic}")
        match typeDevice:
            case "IoTTemp":

receiveMessage.IoTGasOpenLeak_MessageHandler(message, device)
                scenario.checkScenario(device, message,
MQTTclient)
            case "IoTIR":
                receiveMessage.IoTIR_MessageHandler(message,
device)
            case "IoTGas":

receiveMessage.IoTGasOpenLeak_MessageHandler(message, device)
                scenario.checkScenario(device, message,
MQTTclient)
            case "IoTOpen":

receiveMessage.IoTGasOpenLeak_MessageHandler(message, device)
                scenario.checkScenario(device, message,
MQTTclient)
            case "IoTLeak":

receiveMessage.IoTGasOpenLeak_MessageHandler(message, device)
                scenario.checkScenario(device, message,
MQTTclient)
```

Рисунок 3.29 – Функція приймання даних

Це функція викликається коли на будь-який топік MQTT-брокеру приходить повідомлення. В якості аргументу передається саме повідомлення та топік, на яке воно прийшло.

Розглянемо назву топіку який приймає дані: «тип пристрою:назва пристрою /fromDevice». Відповідно, назва топіку, до якого передаються дані зі сторони серверу виглядає наступним чином: «тип пристрою:назва пристрою /toDevice». Така структура спрощує роботу з пристроями та дозволяє зручніше оброблювати дані. Знаючи це, можемо отримати тип пристрою та його назву.

Далі, відповідно до типу пристрою, відбувається обробка та запис даних з пристрою, після чого перевіряється, чи задіяний цей пристрій у сценаріях.

Повний код керуючої програми наведено у ДОДАТКУ Б.

3.5 Створення підсистеми аналізу використання IoT-системи

Дана система аналізує історію спрацьовувань пристроїв з метою пошуку закономірностей в їх. Тобто, є якщо взаємозв’язок між двома пристроями, система повинна його знайти. Система побудована на рекурентній нейронній мережі за допомогою мови Python. Використовані бібліотеки – Pandas, Numpy та TensorFlow.

Імпортуємо необхідні бібліотеки (рис. 3.30):

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

Рисунок 3.30 – Необхідні бібліотеки

Далі, завантажуємо дані (рис. 3.31):

```
data = pd.read_csv('dataset.csv')
X = data[['IoTTemp']].values
y = data['IoTSocket'].values
```

Рисунок 3.31 – Завантаження даних

В файлі «dataset.csv» (рис. 3.32) присутні дані про всі пристрої системи, але усі разом вони не потрібні. Для коректної роботи системи, пристрої зрівнюються між собою.

opening_sensor	water_leak_sensor	temperature	gas_leak_sensor	power_socket	ir_control
0	0	22.406473045760546	578	0	87882247
0	0	13.169940527852317	488	0	15478878
0	1	16.209777353152802	634	0	46128129
0	1	27.465962397823414	229	1	44550325
1	2	11.956137737814762	1023	0	71449993
1	2	12.503922549681498	444	0	39915838
0	3	37.68271571540183	755	1	54772085
1	3	11.756527877319376	962	0	20311825
1	4	36.598126457214	4	1	64732500

Рисунок 3.32 – Зміст файлу «data.csv»

Наступним кроком є нормалізація та поділ даних на навчальну та тестову вибірки (рис. 3.33). Нормалізація у нейронних мережах - це процес приведення вхідних даних до стандартного діапазону або норми, з метою поліпшення швидкості навчання та стійкості моделі.

Одна з найпоширеніших форм нормалізації - це "зсув та масштабування". При такій нормалізації значення вхідних даних перетворюються таким чином, щоб вони знаходилися в певному діапазоні, зазвичай від 0 до 1. Це досягається шляхом віднімання мінімального значення від кожного значення та поділу на різницю між максимальним і мінімальним значеннями.

Нормалізація має кілька переваг:

- Зменшення залежності від початкових значень. Нейронні мережі можуть бути чутливими до початкових значень вхідних даних. Нормалізація допомагає уникнути цієї чутливості та поліпшує здатність моделі до узагальнення на нових даних.
- Забезпечення стійкості навчання. Нормалізація даних сприяє стійкому навчанню моделі, оскільки зменшує різницю у шкалі значень між вагами нейронів у різних шарах мережі.

Враховуючи ці переваги, нормалізація є важливим кроком при підготовці даних для навчання нейронних мереж.

```
X = (X - np.mean(X)) / np.std(X)
train_size = int(len(data) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Рисунок 3.33 – Нормалізація та поділ даних на тестову та навчальну вибірки

Далі, йде створення моделі (рис. 3.34).

```
model = Sequential()
model.add(LSTM(32, input_shape=(1, 1)))
model.add(Dense(1, activation='sigmoid'))
```

Рисунок 3.34 – Створення моделі

Створена нейронна мережа з одним шаром LSTM (Long Short-Term Memory) та одним повнозв'язаним шаром Dense. LSTM шар призначений для роботи з послідовними даними. LSTM шар має 32 нейрони. Dense шар має один нейрон і використовує функцію активації sigmoid, яка обмежує вихідний сигнал до значень між 0 та 1.

І, нарешті, навчання моделі (рис. 3.35).

```
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=16, verbose=1)
```

Рисунок 3.35 – Навчання моделі

Розглянемо аргументи функції компіляції моделі:

- loss=«binary_crossentropy». Використовується функція втрати «binary_crossentropy», що підходить для бінарної класифікації, де модель намагається передбачити два класи.
- optimizer=«adam». Використовується алгоритм оптимізації "adam", який ефективно оновлює ваги мережі на основі градієнтів.

- metrics=[«accuracy»]. Вимірювання точності використовується для оцінки продуктивності моделі під час навчання.

Розглянемо аргументи функції тренування моделі:

- epochs=10. Кількість епох навчання, тобто кількість разів, коли модель пройде через весь набір даних.
- batch_size=1. Кількість прикладів даних, що використовуються для одного оновлення ваг моделі.
- verbose=1. Налаштування виведення під час навчання, де значення 1 означає виведення прогресу навчання на кожній епоції.

Порівняємо результати моделі для різних пристроїв. Ось, наприклад, результат моделі для пристроїв «Temperature» та «Socket», зображений на рис. 3.36.

```
Epoch 1/10
50/50 [==============================] - 2s 2ms/step - loss: 0.6934 - accuracy: 0.4812
Epoch 2/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5138
Epoch 3/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6930 - accuracy: 0.5138
Epoch 4/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6930 - accuracy: 0.5138
Epoch 5/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6930 - accuracy: 0.5138
Epoch 6/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6928 - accuracy: 0.5138
Epoch 7/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6929 - accuracy: 0.5088
Epoch 8/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6927 - accuracy: 0.5125
Epoch 9/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6928 - accuracy: 0.5063
Epoch 10/10
50/50 [==============================] - 0s 2ms/step - loss: 0.6928 - accuracy: 0.5188
7/7 [==============================] - 1s 1ms/step - loss: 0.6970 - accuracy: 0.4750
Test Loss: 0.6970
Test Accuracy: 0.4750

Process finished with exit code 0
```

Рисунок 3.36 – Результат моделі для приладів Temperature та Socket

Метрика «loss» (втрата) використовується для вимірювання величини помилки моделі під час навчання. Вона вказує, наскільки добре модель

передбачає вихідні значення в порівнянні зі звичайними або очікуваними значеннями.

Метрика «accuracy» використовується для оцінки продуктивності моделі під час навчання. Вона вимірює відношення кількості правильно класифікованих прикладів до загальної кількості прикладів у наборі даних.

Як можемо бачити, результати погані, що свідчить про те, що між пристроями немає зв’язку. Спробуємо змінити прилад «Temperature» на прилад «opening_sensor». Результат показано на рис. 3.37

```
Epoch 1/10
50/50 [==============================] - 2s 1ms/step - loss: 0.6336 - accuracy: 0.9925
Epoch 2/10
50/50 [==============================] - 0s 1ms/step - loss: 0.5724 - accuracy: 0.9850
Epoch 3/10
50/50 [==============================] - 0s 2ms/step - loss: 0.4981 - accuracy: 0.9850
Epoch 4/10
50/50 [==============================] - 0s 2ms/step - loss: 0.4134 - accuracy: 0.9825
Epoch 5/10
50/50 [==============================] - 0s 1ms/step - loss: 0.3329 - accuracy: 0.9862
Epoch 6/10
50/50 [==============================] - 0s 2ms/step - loss: 0.2673 - accuracy: 0.9850
Epoch 7/10
50/50 [==============================] - 0s 2ms/step - loss: 0.2187 - accuracy: 0.9887
Epoch 8/10
50/50 [==============================] - 0s 2ms/step - loss: 0.1835 - accuracy: 0.9925
Epoch 9/10
50/50 [==============================] - 0s 2ms/step - loss: 0.1577 - accuracy: 0.9925
Epoch 10/10
50/50 [==============================] - 0s 1ms/step - loss: 0.1384 - accuracy: 0.9925
7/7 [==============================] - 0s 1ms/step - loss: 0.1361 - accuracy: 0.9950
Test Loss: 0.1361
Test Accuracy: 0.8732
```

Рисунок 3.37 – Результат моделі для приладів opening_sensor та Socket

І ось вже зовсім інший результат. Можна з впевненістю сказати, що модель працює та здатна знаходити зв’язки між даними з пристроїв.

3.6 Розробка підсистеми розпізнавання осіб

Для розробки програми використовується бібліотека OpenCV мови програмування Python. Розглянемо основні етапи програми. Для початку, імпортуємо необхідну бібліотеку та завантажимо тренувальні дані (рис. 3.38)

```
import cv2
face_cascade = cv2.CascadeClassifier("venv/Lib/site-
packages/cv2/data/haarcascade_frontalface_default.xml")
```

Рисунок 3.38 – Імпорт бібліотеки та тренувальних даних

Після цього, ініціалізуємо відеопоток з потрібної камери (рис. 3.39).

```
# Ініціалізація відеопотоку
video_capture = cv2.VideoCapture(rtcp_url)
```

Рисунок 3.39 – Ініціалізація відеокамери

Тепер, коли ми отримали доступ до камери, починаємо пошук людей з зображення. Для цього захоплюємо кадр з відеопотоку та переводимо його у відтінки сірого (рис. 3.40).

```
# Захоплення кадру з відеопотоку
    ret, frame = video_capture.read()
    # Перетворення кадру на відтінки сірого
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Рисунок 3.40 – Захоплення кадру

Далі йде розпізнавання. Для цього викликаємо метод detectMultiScale. Він використовується для виявлення об'єктів на зображенні або відео за допомогою каскадів Хаара. Цей метод приймає на вхід зображення (або кадр відео) і шукає об'єктів, заданих необхідним класифікатором. Він повертає список прямокутних областей (координати прямокутника), де було виявлено об'єкти. Розглянемо аргументи цієї функції:

- image: Вихідне зображення, на якому виконується виявлення об'єктів.
- scaleFactor: Масштабний коефіцієнт, що визначає, як багато разів зображення зменшується на кожній ітерації виявлення об'єктів.

Найменше значення призводить до детальнішого пошуку, але потребує більше обчислювальних ресурсів.

- minNeighbors: Мінімальна кількість сусідів, які повинні бути виявлені для того, щоб вважати прямокутну область, що містить об'єкт. Цей параметр може допомогти усунути помилкові виявлення, якщо він встановлений надто високим значенням.
- minSize та maxSize: Опціональні аргументи, що задають мінімальний та максимальний розміри об'єктів, які потрібно виявити. Це може бути корисним для встановлення обмежень на розміри об'єктів.

Якщо об’єкти виявлені, вони виділяються зеленим квадратом. Використання цього методу приведено в рис. 3.41

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=8, mnSize=(30, 30))
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

Рисунок 3.41 – розпізнавання та виділення осіб

Протестуємо дану програму (рис. 3.42).

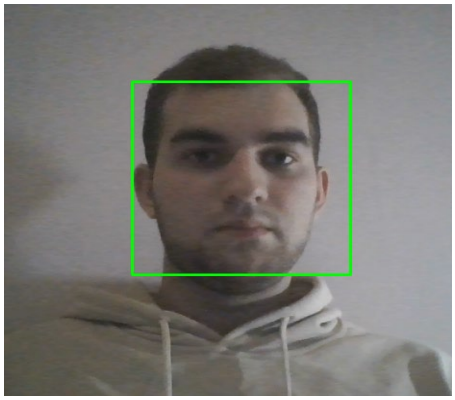

Рисунок 3.42 – Тестування програми

Як можемо бачити, програма успішно справляється з розпізнаванням людей на відео.

3.7 Створення мобільного додатку клієнта IoT-системи

Мобільний додаток був створений за допомогою мови Python та бібліотеки KivyMD. За його допомогою користувач повинен виконувати керування системою. Розглянемо головну сторінку програми, зображений на рис. 3.43.

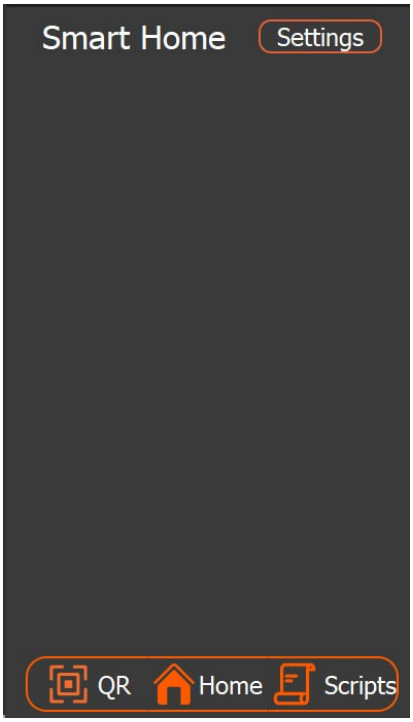

Рисунок 3.43 – Головний екран мобільного додатку

На головній сторінці знаходяться усі пристрої системи. Оскільки зараз не один пристрій ще не підключено, сторінка пуста. Також, можемо бачити меню вибору екрану, в якому три позиції:

- QR. На цій сторінці розташовується сканер QR-кодів, за допомогою якого підключаються нові пристрої.
- Home. Головна сторінка, на якій зображено усі пристрої в мережі.
- Scripts. На цій сторінці розміщуються усі сценарії.

Розглянемо меню «Settings», яке показано на рис. 3.44.

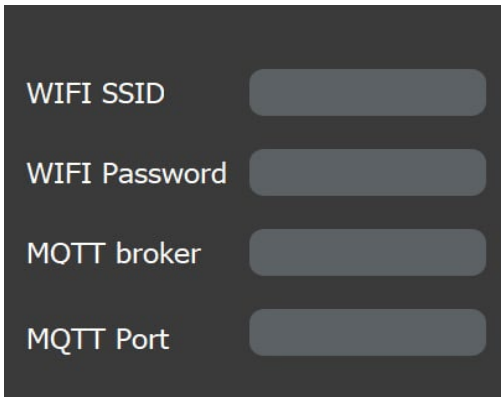

Рисунок 3.44 – Меню налаштування

В цьому меню необхідно ввести налаштування, які будуть відправлятись на прилад при першому підключенні, а саме:

- Назва мережі;
- пароль від мережі;
- адреса MQTT-брокеру;
- порт підключення MQTT-брокеру.

Розглянемо сторінку «Scripts», зображену на рис. 3.45.

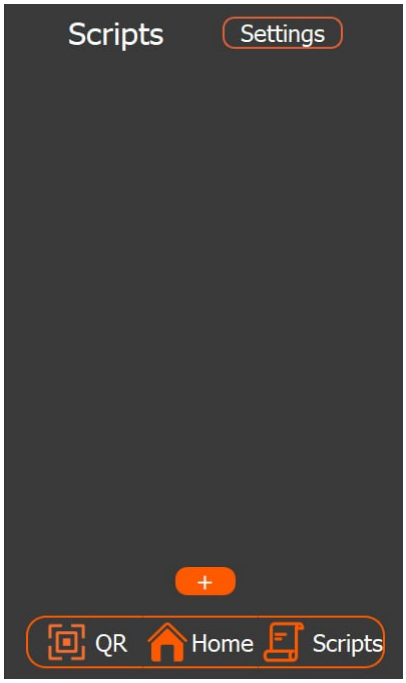

Рисунок 3.45 – Сторінка сценаріїв

На цій сторінці знаходяться усі сценарії системи. Оскільки зараз не один сценарій не створено, сторінка пуста. Також, можемо бачити кнопку додавання сценарію, обозначену як «+». При натисканні, відкривається вікно додавання сценарію, яке показано на рис. 3.46.

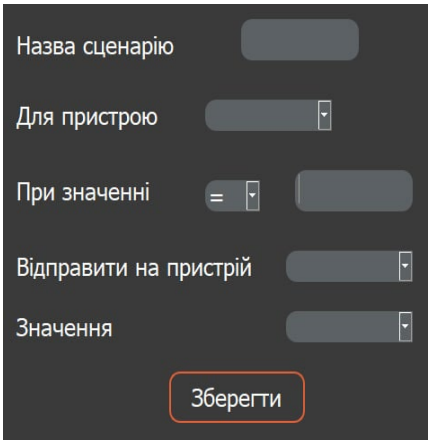

Рисунок 3.46 – Вікно створення сценарію

Опишемо поля цього вікна, усі з яких є обов’язкові до заповнення:

- «Назва сценарію». Пропонує ввести назву сценарію у відповідне поле;
- «Для пристрою». У випадаючому списку міститься перелік пристроїв системи, один з яких потрібно обрати та чиє значення якого буде відслідковуватись;
- «При значенні». Пропонує обрати знак (>, <, =) та невід’ємне значення для порівняння;
- «Відправити на пристрій». У випадаючому списку міститься перелік керуючих пристроїв системи, один з яких потрібно обрати для відправки на нього команди при спрацьовуванні сценарію;
- «Значення». У випадаючому списку міститься перелік команд для обраного пристрою. Наприклад, для пристрою типу «IoTSocket» буде міститиметься значення 0 і 1.

Код програми для мобільного додатку міститься у ДОДАТКУ В.

3.8. Розробка макету та тестування IoT системи

Для тестування системи зберемо прилади типу «IoTTemp» та «IoTSocket». Розберемо підключення, їх взаємодію та створення сценарію для цих пристроїв. Розпочнемо з пристрою типу «IoTTemp». На рис. 3.47 та рис 3.48 зображено зібраний пристрій у закритому та відкритому корпусі відповідно.

Рисунок 3.47 – Пристрій типу «IoTTemp» у закритому корпусі

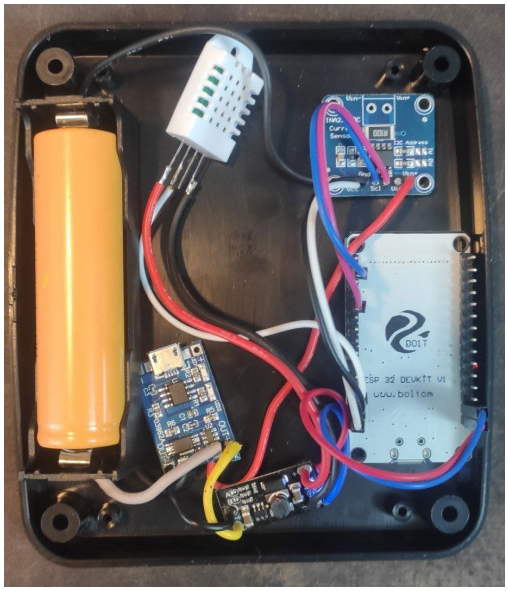

Рисунок 3.48 – Пристрій типу «IoTTemp» у відкритому корпусі

Тепер, побудуємо пристрій типу «IoTSocket». На рис. 3.49 та рис 3.50 зображено зібраний пристрій у закритому та відкритому корпусі відповідно.

Рисунок 3.49 – Пристрій типу «IoTSocket» у закритому корпусі

Рисунок 3.50 – Пристрій типу «IoTSocket» у відкритому корпусі

Можемо бачити, що при зборці використовувались лише компоненти, описані у підрозділі 3.1.

Додамо пристрій під назвою «TempExample» типу «IoTTemp» до мережі (рис. 3.51). Для цього, відскануємо у додатку QR-код, який містить:

- Назву мережі, для якої пристрій виступає у якості точки доступу;
- Пароль від мережі;
- порт для підключення до пристрою;
- IP-адресу пристрою у цій мережі.

Рисунок 3.51 – Додавання пристрою «TempExample» до мережі

Теж саме повторюється і для пристрою «SocketExample» типу «IoTSocket». Перейдемо до головного екрану та побачимо, що з`явилися два нових пристрої (рис. 3.52). Оскільки пристрій «TempExample» був доданий перший, він вже встиг відправити дані.

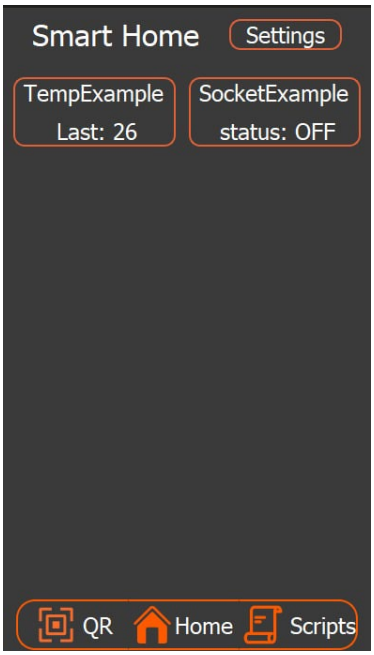

Рисунок 3.52 – Головний екран після додавання пристроїв

При натисканні на елемент «TempExample», бачимо вікно з інформацією з пристрою (рис. 3.53).

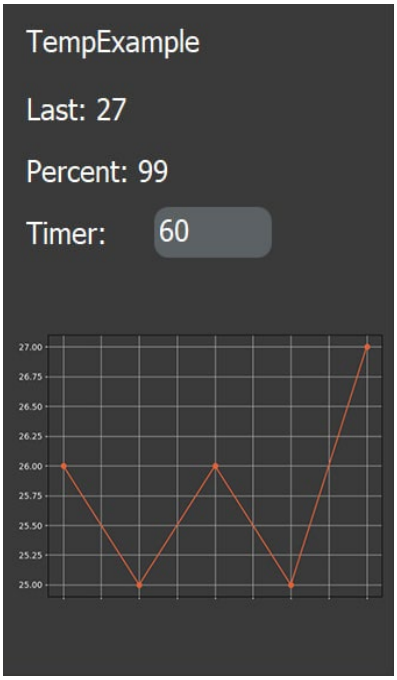

Рисунок 3.53 – Вікно пристрою «TempExample»

Розглянемо елементи вікна пристрою «TempExample»:

- Last. Остання отримана температура з пристрою;
- Percent. Показник заряду акумулятора пристрою;
- Timer. Значення, яке відповідає з яким періодом у секундах пристрій відправляє дані. За замовченням, встановлено значення 60;
- графік, якій відображає змінення температури при кожному отриманні даних.

Розглянемо вікно пристрою «IoTSocket», зображене на рис. 3.54.

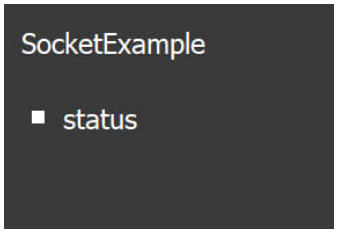

Рисунок 3.54 – Вікно пристрою «IoTSocket»

Можемо бачити елемент «status», який відповідає за вмикання та вимикнення пристрою. Якщо поле ліворуч від надпису «staus» є пустим, це свідчить про вимкнутий стан пристрою.

Розглянемо створення сценарію. Заповнимо відповідні поля як показано на рис. 3.55.

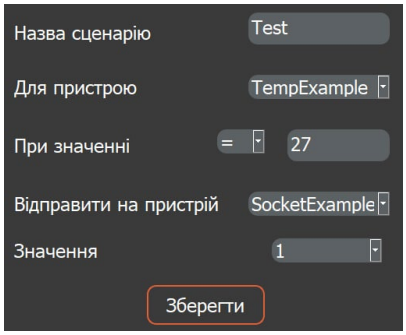

Рисунок 3.55 – Створення сценарію

При натисканні кнопки «Зберегти», ми потрапляємо назад до сторінки сценаріїв, але тепер на ній є новий сценарій (рис. 3.56).

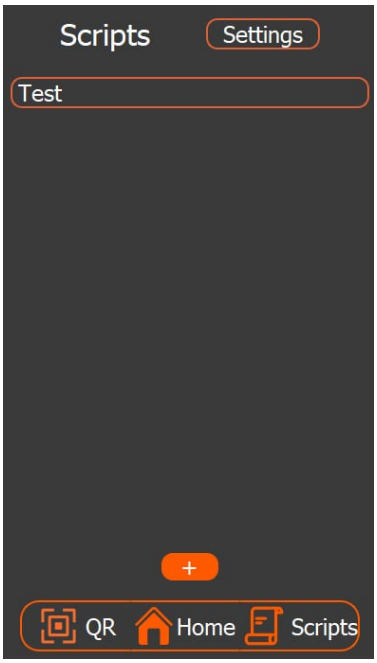

Рисунок 3.56 – Новий сценарій

Відтепер, при значенні температури 27 градусів, на пристрій «IoTSocket» буде відправлятися команда 1, яка свідчить про вмикання пристрою.

ВИСНОВКИ

Розумний будинок представляє собою інноваційне рішення, яке забезпечує гнучкість та інтегрованість у своїй розробці. Система здатна працювати з новими пристроями, які підключаються після налаштування, що робить її досить масштабованою та пристосованою до змін.

Однак, важливо відзначити, що при розробці системи була приділена велика увага приватності користувачів та конфіденційності інформації. Система забезпечує захист від несанкціонованого доступу та забезпечує гарантований рівень безпеки для користувачів.

В процесі виконання дипломної роботи проведено аналіз сучасного обладнання мережі IoT та його елементів для системи «Розумний будинок».

Було проведено огляд сучасних технологій та протоколів сучасних IoT-мереж, таких як Zigbee, Z-wave та WiFi. Проведено огляд сучасних рішень для побудови мереж та систем «Розумний будинок».

В роботі також виконано проектування фізичної та логічної топології IoT-мережі та алгоритмів роботи пристроїв системи.

При розробці IoT-системи обране доцільне обладнання для реалізації IoT мережі розумного будинку та розроблені принципові схеми построїв. Для налаштування та конфігурування розроблена керуюча програма, яка успішно виконує управління пристроями розумного будинку та взаємодіє з мобільним додатком. Створена система рекомендацій доцільно пропонує користувачу сценарії використання пристроїв. Розроблена програма розпізнавання осіб успішно справляється зі своєю задачею. Створений мобільний додаток клієнта є зручним у повсякденному використанні. Проведене макетування та тестування IoT системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Єлесін О. О., Волощук Л. А. Застосування методів штучного інтелекту при розробці IoT системи розумного будинку / Інформатика, інформаційні системи та технології: тези доповідей XX всеукраїнської конференції студентів і молодих науковців. Одеса, 28 квітня 2023 р. – Одеса, 2023. – с.212-213.
2. Опис IoT [Електронний ресурс] – Режим доступу: https://www.it.ua/knowledge-base/technology-innovation/internet-veschej-internet-of-things-iot
3. Захист IoT [Електронний ресурс] – Режим доступу: https://corewin.ua/blog/attacks-on-iot-how-protect/
4. Система Ajax [Електронний ресурс] – Режим доступу: https://ajax.systems/ua/
5. Система BroadLink [Електронний ресурс] – Режим доступу: https://broadlink.com.ua/
6. Протоколи IoT системи [Електронний ресурс] – Режим доступу: https://wisehome.com.ua/ua/scho-take-protokol-dlya-rozumnogo-budinku-6-populyarnih-vidiv/
7. ESP-32 [Електронний ресурс] – Режим доступу: https://www.espressif.com/en/products/socs/esp32
8. ESP-8266 [Електронний ресурс] – Режим доступу: https://www.espressif.com/en/products/socs/esp8266
9. Raspberry 3 Model B [Електронний ресурс] – Режим доступу: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/
10. Маршрутизатор TP–Link Archer C80 AC1900 [Електронний ресурс] – Режим доступу: https://www.tp-link.com/uk-ua/home-networking/wifi-router/archer-c80/

11. Камера відеоспостереження Tapo C320WS [Електронний ресурс] – Режим доступу: https://www.tp-link.com/uk-ua/home-networking/cloud-camera/tapo-c320ws/

ДОДАТОК А

Код програми мікроконтролеру пристрою «IoTTemp»

```
#include <Arduino.h>
#include "DHTesp.h"
#include <WiFi.h>
#include <WiFiClient.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include "initialWIFI&MQTTsetup.h"
#include "structures.h"
#include <EEPROM.h>
#include <Wire.h>
#include <Adafruit_INA219.h>

Adafruit_INA219 ina219;
char* ssidAP = "DHT client";
const char* clientID = "DHT-1";
WiFiClient client;
PubSubClient MQTTclient(client);
const uint8_t lenCommand = 5;
const uint16_t buffSize = 2048;
connectInfo info;
const uint8_t DHT_PIN = 15;
DHTesp dhtSensor;
struct DHTdata
{
  float temperature;
  float humidity;
  uint_fast8_t percent;
};
DHTdata publishData;
uint64_t TIMER;
struct TIMEdata{
  char time[TIME_LEN+1];
};
void publishDHTdataToMQTT(DHTdata data){
  extern connectInfo info;
  extern PubSubClient MQTTclient;
  if(data.humidity != 0){
    Serial.println("\nStart send message to MQTT...");
    StaticJsonDocument<buffSize> jsonDocument;
    char jsonBuffer[buffSize];
    jsonDocument["temperature"] = data.temperature;
    jsonDocument["humidity"] = data.humidity;
    jsonDocument["percent"] = data.percent;
    serializeJson(jsonDocument, jsonBuffer);
    Serial.print("topic in publish: ");
```

```
    Serial.println(info.publishTopic);
    if (MQTTclient.publish(info.publishTopic, jsonBuffer)){
      Serial.println("Data send to MQTT brocker!");
    }else{
      Serial.println("Error! Data don`t send to MQTT brocker!");
    }
  }else{
    Serial.println("Error! Data don`t send to MQTT brocker!");
  }
}
void callback(char* topic, byte *payload, unsigned int length) {
  if(strlen((char*)payload) < 5){
    Serial.println("Message less than 5 bytes");
    return;
  }
  Serial.println("\nMessage received:");
  char command[lenCommand+1];
  command[lenCommand] = '\0';
  for(int i = 0; i < lenCommand; i++){
    command[i] = (char)payload[i];
  }
  Serial.print("command: ");
  Serial.println(command);
  TIMEdata timeData;
  EEPROM.begin(MEMORY_SIZE);
  EEPROM.get(TIME_ADDR, timeData);
  EEPROM.commit();
  Serial.print("Time from EEPROM: ");
  Serial.println(timeData.time);
  char* data = strtok((char*)payload, "/");
  char* time = strtok(NULL, "/");
  Serial.print("Time from message: ");
  Serial.println(time);
  if(time == NULL){
      return;
    }
  if(strcmp(time, timeData.time) != 0){
    Serial.println("New message !\nRead data...");
    EEPROM.begin(MEMORY_SIZE);
    strcpy(timeData.time, time);
    timeData.time[TIME_LEN] = (char)'\0';
    if(strcmp(command, "NEWCN") == 0){
      EEPROM.put(TIME_ADDR, timeData);
      EEPROM.commit();
      EEPROM.end();
      Serial.println("New data put into EEPROM !");
      fullSetup(ssidAP, clientID, clientID, client, MQTTclient);
    }else if(strcmp(command, "TIMER") == 0){
      data = strtok(data, " ");
      data = strtok(NULL, " ");
      int Localtimer = atoi(data);
      if(Localtimer == 0){
```

```
        Serial.println("failed to convert data string to number
!");
        return;
      }
      Serial.print("New value of TIMER: ");
      Serial.println(Localtimer);
      EEPROM.begin(MEMORY_SIZE);
      EEPROM.write(TIMER_ADDR, Localtimer);
      EEPROM.put(TIME_ADDR, timeData);
      EEPROM.commit();
      EEPROM.end();
      Serial.println("New data put into EEPROM !");
    }else{
      Serial.println("Not format message");
      return;
    }
  }else{
    Serial.println("Not new message ! Data skip.");
    return;
  }
}
void getBatteryPercent(){
  // (x - in_min) * (out_max - out_min) / (in_max - in_min) +
out_min;
  float busvoltage = 0;       // Создаем переменную busvoltage
  busvoltage = ina219.getBusVoltage_V();            // Получение
значение напряжения V
  Serial.print("Bus Voltage:   ");
  Serial.print(busvoltage);
  Serial.println(" V");
  float normalization = (busvoltage - MIN_VOLTAGE) /
(MAX_VOLTAGE - MIN_VOLTAGE);
  uint8_t percent = normalization * 100;
  publishData.percent = percent;
  Serial.print("Percent: ");
Serial.println(publishData.percent);
}
void printInfo(){
  Serial.print("ssid: ");
  Serial.println((char*)info.ssid);
  Serial.print("password: ");
  Serial.println((char*)info.password);
  Serial.print("broker: ");
  Serial.println((char*)info.mqttBroker);
  Serial.print("port: ");
  Serial.println(info.mqttPort);
  Serial.print("publishTopic: ");
  Serial.println((char*)info.publishTopic);
  Serial.print("subscribeTopic: ");
  Serial.println((char*)info.subscibeTopic);
};
```

```
void setup() {
  Serial.begin(115200);
  delay(200);
  EEPROM.begin(MEMORY_SIZE);
  if(EEPROM.read(INIT_ADDR) != INIT_KEY){
    Serial.println("First ON");
    EEPROM.put(TIMER_ADDR, DEFAULT_TIMER);
    fullSetup(ssidAP, clientID, clientID, client, MQTTclient);
  }else{
    Serial.println("Read data from EEPROM...");
    EEPROM.get(DATA_ADDR, info);
    //EEPROM.get(TIMER_ADDR, TIMER);
    TIMER = EEPROM.read(TIMER_ADDR);
    EEPROM.commit();
    EEPROM.end();
    printInfo();
    Serial.print("TIMER: ");
    Serial.println(TIMER);
  }
  esp_sleep_enable_timer_wakeup(TIMER * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " +
String(TIMER) + " Seconds");
  setupWiFi(info);
  MQTTclient.setServer(info.mqttBroker, info.mqttPort);
  MQTTclient.setBufferSize(MESSAGE_BUFFER);
  reconnectMQTT(clientID, clientID, info, MQTTclient);
  MQTTclient.setCallback(callback);
  delay(200);
  for(int i = 0; i < 100; i++){
    MQTTclient.loop();
    delay(10);
  }
  Serial.println("---");
  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);
  TempAndHumidity  readSensor = dhtSensor.getTempAndHumidity();
  publishData.temperature = (round(readSensor.temperature * 10)
/ 10);
  publishData.humidity = (round(readSensor.humidity * 10) / 10);
  Serial.print("Temp: ");
  Serial.println(publishData.temperature);
  Serial.print("Humidity: ");
  Serial.println(publishData.humidity);
  Serial.println("---");

  Serial.println("Start INA219 chip");
  while(!ina219.begin()) {
    Serial.print(".");
    delay(200);
  }
  getBatteryPercent();
  Serial.println("---");
  publishDHTdataToMQTT(publishData);
```

```
  delay(200);
  Serial.println("Going to sleep now");
  esp_deep_sleep_start();
}
void loop() {}
```

Код програми мікроконтролеру пристрою «IoTSocket»

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include "initialWIFI&MQTTsetup.h"
#include "structures.h"
#include <EEPROM.h>
#include <Wire.h>

{
  char messageBuffer[40];
  snprintf(messageBuffer, sizeof(messageBuffer), "Relay status
is :: %s, len :: %u", message, len);
  Serial.println(messageBuffer);
  if (strcmp(message, "0") == 0)
  {
    Serial.println("Turning ON RELAY");
    digitalWrite(relayPin, HIGH);
  }
  else if(strcmp(message, "1") == 0)
  {
    Serial.println("Turning OFF RELAY");
    digitalWrite(relayPin, LOW);
  }
}
void setupWifi()
{
  delay(10);
  Serial.println(F("Adafruit MQTT demo"));
  // Connect to WiFi access point.
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
```

```
  Serial.println(WiFi.localIP());
}

void setup() {
  delay(200);
  Serial.begin(115200);
  pinMode(relayPin,OUTPUT);
  digitalWrite(relayPin, LOW);
  Serial.println("In setup before setupWifi");
  setupWifi();
  relayControl.setCallback(&relayCallback);
  mqtt.subscribe(&relayControl);
}
void loop() {
  // Connect to MQTT
  connectToMQTTServer();
  mqtt.processPackets(10000);
}
```

Код програми мікроконтролеру пристрою «IoTIR»

```
#include <Arduino.h>
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRsend.h>
#include <IRutils.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include "initialWIFI&MQTTsetup.h"
#include "structures.h"
#include <EEPROM.h>

const uint8_t lenCommand = 5;
const uint16_t buffSize = 2048;
WiFiClient client;
PubSubClient MQTTclient(client);
connectInfo info;
//extern connectInfo infoLocal;
byte* payload;
const char* clientId = "IR-1";
const char* user = "IR-1";
char* ssidAP = "IR client";
// IR settings
const uint16_t kFrequency = 38000;  // in Hz. e.g. 38kHz.
const uint8_t kRecvPin = 32;
const uint8_t kTransmitPin = 33;
struct IRdata
{
  uint32_t command;
  uint16_t sizeIRdata;
```

```
  uint16_t* rawArray;
};
IRrecv irrecv(kRecvPin);
IRsend irsend(kTransmitPin);
decode_results results;


IRdata receiveIRdata(){
  if (irrecv.decode(&results)){
    Serial.print("\nAvaible data.\nHex: ");
    char hex[64];
    sprintf(hex, "%X", results.value);
    Serial.println(hex);
    Serial.print("Data value: ");
    Serial.println(results.value);
    uint16_t size = results.bits;
    uint16_t *raw_array = resultToRawArray(&results);
    size = getCorrectedRawLength(&results);
    Serial.print("Raw array: ");
    for(int i = 0; i < size; i++){
      Serial.print(raw_array[i]);
      Serial.print(", ");
    }
    Serial.println();
    Serial.print("size: ");
    Serial.print(size);
    Serial.println();
    delay(100);
    irrecv.resume();
    IRdata dataArr {results.value, size, raw_array};
    return dataArr;
  }else{
    IRdata dataArr {NULL, NULL, NULL};
    return dataArr;
  }
}
void sendIRdata(IRdata data){
  pinMode(kRecvPin, OUTPUT);
  irsend.sendRaw(data.rawArray, data.sizeIRdata, kFrequency);
  Serial.println("Data was send !");
  delay(250);
  pinMode(kRecvPin, INPUT);
}
void repeatIRdata(){
  IRdata data = receiveIRdata();
  if(data.command != NULL){
    sendIRdata(data);
  }
}
IRdata parsingIRdata(byte *payload, unsigned int length){
  DynamicJsonDocument doc(buffSize);
  deserializeJson(doc, payload, length);
```

```
  const char *name = doc["name"];
  uint32_t command = doc["command"];
  uint16_t size = doc["arraySize"];
  Serial.print("name: ");
  Serial.print(name);
  Serial.println();
  Serial.print("command: ");
  Serial.print(command);
  Serial.println();
  Serial.print("size: ");
  Serial.print(size);
  Serial.println();
  JsonArray data = doc["data"];
  uint16_t raw_array[size];
  u_int16_t len_arr = 0;
  for(JsonVariant item : data) {
    int16_t element = (u_int16_t)item;
    len_arr++;
    raw_array[len_arr-1] = element;
  }
  Serial.print("Raw array: ");
  for(int i = 0; i < size; i++){
    Serial.print(raw_array[i]);
    Serial.print(", ");
  }
  IRdata irData {command, size, raw_array};
  return irData;
}
void publishIRdataToMQTT(IRdata data){
  extern connectInfo info;
  extern PubSubClient MQTTclient;
  if(data.command != NULL){
    Serial.println("\nStart send message to MQTT...");
    StaticJsonDocument<buffSize> jsonDocument;
    char jsonBuffer[buffSize];
    jsonDocument["command"] = data.command;
    jsonDocument["arraySize"] = data.sizeIRdata;
    // array create
    JsonArray dataArray =
jsonDocument.createNestedArray("data");
    copyArray(data.rawArray, data.sizeIRdata, dataArray);
    //
    serializeJson(jsonDocument, jsonBuffer);
    Serial.println("info in publish:");
    Serial.println(info.mqttBroker);
    if (MQTTclient.publish(info.publishTopic, jsonBuffer)){
      Serial.println("Data send to MQTT brocker!");
    }else{
      Serial.println("Error! Data don`t send to MQTT brocker!");
    }
  }else{
    Serial.println("Error! Data don`t send to MQTT brocker!");
```

```
  }
}
void callback(char* topic, byte *payload, unsigned int length) {
  Serial.println("\nMessage received:");
  char command[lenCommand+1];
  command[lenCommand] = '\0';
  for(int i = 0; i < lenCommand; i++){
    command[i] = (char)payload[i];
  }
  IRdata data {NULL, NULL, NULL};
  if(strcmp(command, "NEWIR") == 0){
    Serial.println("Got new IR command !");
    do
    {
      data = receiveIRdata();
      delay(100);
    } while(data.command == NULL);
    publishIRdataToMQTT(data);
  }else if(strcmp(command, "NEWCN") == 0){
    fullSetup(ssidAP, clientId, user, client, MQTTclient);
  }else{
    IRdata data = parsingIRdata(payload, length);
    sendIRdata(data);
  }
}
void printInfo(){
  Serial.print("ssid: ");
  Serial.println((char*)info.ssid);
  Serial.print("password: ");
  Serial.println((char*)info.password);
  Serial.print("broker: ");
  Serial.println((char*)info.mqttBroker);
  Serial.print("port: ");
  Serial.println(info.mqttPort);
  Serial.print("publishTopic: ");
  Serial.println((char*)info.publishTopic);
  Serial.print("subscribeTopic: ");
  Serial.println((char*)info.subscibeTopic);
};
void setup() {
  Serial.begin(115200);
  delay(200);
  irrecv.enableIRIn();
  irsend.begin();
  EEPROM.begin(MEMORY_SIZE);
  if (EEPROM.read(INIT_ADDR) != INIT_KEY){
    Serial.println("First ON");
    fullSetup(ssidAP, clientId, user, client, MQTTclient);
  }else{
    Serial.println("Read data from EEPROM...");
    EEPROM.get(DATA_ADDR, info);
    EEPROM.commit();
```

```
  }
  Serial.println("info in main:");
  printInfo();
  setupWiFi(info);
  MQTTclient.setServer(info.mqttBroker, info.mqttPort);
  MQTTclient.setBufferSize(MESSAGE_BUFFER);
  reconnectMQTT(clientId, user, info, MQTTclient);
  MQTTclient.setCallback(callback);
}
void loop() {
  MQTTclient.loop();  // check MQTT
  repeatIRdata();
}
```

ДОДАТОК Б

Код керуючої програми

```
import csv
import pandas as pd
import os
import shutil
import receiveMessage
import workWithMQTT
import sendMessage
from scenario import categoryDevice
import scenario
from workWithMQTT import MQTTclient

DEFAULT_TIMER = 60
header_csv = {"IoTTemp": ["datetime", "temperature",
"charge"],
                    "IoTOpen": ["datetime", "value",
"charge"],
                    "IoTGas": ["datetime", "value",
"charge"],
                    "IoTLeak": ["datetime", "value",
"charge"],
                    "IoTIR": ["datetime", "value"],
                    "IoTSocket": ["datetime", "value"]}


def addDevice(nameDevice, typeDevice):
    invalid_chars = '/\\:*?"<>|'
    for i_char in invalid_chars:
        if i_char in nameDevice:
            print("Error in addDevice. Invalid sign cannot
be present in the device name")
            return 1
    types = pd.read_csv("devices/typeDevice.csv")
    devices = pd.read_csv("devices/device_list.csv")
    if typeDevice not in types.type.values:
        print("Error in addDevice. Can`t find type device")
        return 2
    if nameDevice in devices.name.values:
        print("Error in addDevice. A device with the same
name already exists")
        return 3
    with open("devices/device_list.csv", "a", newline='')
as csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([nameDevice, typeDevice])
    path = f"devices/{nameDevice}"
```

```
    os.mkdir(path)
    with open(f"{path}/history.csv", "w", newline='') as
csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow(header_csv[typeDevice])
    category = categoryDevice(nameDevice)
    if category == "data":
        with open(f"{path}/settings.txt", "w") as file:
            file.write(f"TIMER:{DEFAULT_TIMER}")
    print("device added successfully")
    return 0

def deleteDevice(nameDevice):
    devices = pd.read_csv("devices/device_list.csv")
    if nameDevice not in devices.values:
        print("Error in deleteDevice. A device with this
name does not exist")
        return 1
    devices = devices.drop(devices[devices['name'] ==
nameDevice].index)
    devices.to_csv("devices/device_list.csv", index=False)
    shutil.rmtree(f"devices/{nameDevice}")
    print("device deleted successfully")
    return 0

def changeTimer(nameDevice, newTimer):
    devices = pd.read_csv("devices/device_list.csv")
    if nameDevice not in devices.name.values:
        print("Error in changeTimer. A device with this
name does not exist")
        return 1
    if categoryDevice(nameDevice) != "data":
        print("Error in changeTimer. it is impossible to
change the timer values for devices of a category that is
not equal to 'data'")
        return 2
    try:
        timer = int(newTimer)
    except:
        print("timer value must be a number")
        return 3
    if timer < 0:
        print("timer value cannot be less than 0")
        return 4
    with open(f"devices/{nameDevice}/settings.txt", "w") as
file:
        file.write(f"TIMER:{timer}")
    print("timer changed successfully")
    return 0

def getTimerValue(nameDevice):
    devices = pd.read_csv("devices/device_list.csv")
```

```
    if nameDevice not in devices.name.values:
        print("Error in getTimerValue. A device with this
name does not exist")
        return 1
    if categoryDevice(nameDevice) != "data":
        print(
            "Error in getTimerValue. it is impossible to
change the timer values for devices of a category that is
not equal to 'data'")
        return 2
    with open(f"devices/{nameDevice}/settings.txt", "r") as
file:
        data = file.read()
    data = data.split(":")
    return int(data[1])

def IoTIR_MessageHandler(message, nameDevice):
    data = json.loads(message)
    command = int(data['command'])
    arraySize = int(data['arraySize'])
    data = data["data"]
    time = datetime.datetime.now()
    jsonData = {
        "command": command,
        "arraySize": arraySize,
        "data": data
    }
    if isExistIR(command) == False:
        json_str = json.dumps(jsonData)
        with open("devices/IRcommands.json", "a",
newline='') as file:
            file.write(json_str)
        with open("devices/IRcommands.json", "a",
newline='') as file:
            file.write('\n')
    else:
        print("This command is already exist")
    print("data saved successfully")

def IoTGasOpenLeak_MessageHandler(message, nameDevice):
    data = json.loads(message)
    value = int(data['value'])
    charge = int(data['percent'])
    time = datetime.datetime.now()
    with open(f"devices/{nameDevice}/history.csv", 'a') as
csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([time, value, charge])
    print("data saved successfully")
    return 0

def isExistIR(command):
```

```
    with open("devices/IRcommands.json", "r") as jsonFile:
        json_str = jsonFile.read()
    json_arr = json_str.split("\n")
    for line in json_arr:
        if line != "":
            data = json.loads(line)
            if data["command"] == command:
                return True
    return False

sendFunction = {"IoTIR": sendMessage.sendTo_IoTIR,
"IoTSocket": sendMessage.sendTo_IoTSocket}

def categoryDevice(nameDevice):
    types = pd.read_csv("devices/typeDevice.csv")
    devices = pd.read_csv("devices/device_list.csv")
    if nameDevice not in devices.name.values:
        print("Error in categoryDevice. A device with this
name does not exist")
        return 1
    type = devices.loc[devices.name ==
nameDevice].type.values[0]
    category = types.loc[types.type ==
type].category.values[0]
    return category

def addScenario(nameScenario,
triggerDevice,triggerValue,operation, slaveDevice,
valueToSend):
    devices = pd.read_csv("devices/device_list.csv")
    if triggerDevice not in devices.name.values and
triggerDevice != "time":
        print(f"Unable to create script. Device
{triggerDevice} not found")
        return 1
    elif slaveDevice not in devices.name.values:
        print(f"Unable to create script. Device
{slaveDevice} not found")
        return 2
    if categoryDevice(slaveDevice) != "actuator":
        print("Unable to create script. the slave device
must be actuator")
        return 3
    if operation not in [">", "<", "="]:
        print("invalid operation type")
        return 4
    compareArr = [triggerDevice,operation,
str(triggerValue), slaveDevice, str(valueToSend)]
    with open("devices/scenarios.csv", 'r') as csvFile:
        csvData = csvFile.read()
        csvArr = csvData.split('\n')
    for line in csvArr:
```

```
        if nameScenario == line.split(",")[0]:
            print(f"script with name {nameScenario} already
exists")
            return 5
        if compareArr == line.split(",")[1:]:
            print("script with the same set exists")
            return 6
    with open("devices/scenarios.csv", 'a') as csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([nameScenario, triggerDevice,
operation, triggerValue, slaveDevice, valueToSend])
    print("script created successfully")
    return 0

def removeScenario(nameScenario):
    scenario = pd.read_csv("devices/scenarios.csv")
    for i in range(len(scenario)):
        if nameScenario == scenario.iloc[i].nameScenario:
            scenario = scenario.drop(i)
    scenario.to_csv("devices/scenarios.csv",
index_label=False)
    print("scenario deleted successfully !")

def checkScenario(nameDevice, valueDevice, MQTTclient):
    print(f"nameDevice: {nameDevice}, valueDevice:
{valueDevice}, len(valueDevice) {len(str(valueDevice))}")
    data = pd.read_csv("devices/scenarios.csv")
    if categoryDevice(nameDevice) == "data" and
len(str(valueDevice)) > 10:
        jsonLoads = json.loads(valueDevice)
        valueDevice = int(jsonLoads['value'])
    for i in range(len(data)):
        lineData = data.iloc[i]
        if lineData.triggerDevice == nameDevice:
            type = typeDevice(lineData.slaveDevice)
            match lineData.operation:
                case '=':
                    if lineData.triggerValue ==
valueDevice:
                        print("scenario activated")
                        print(f"Data {lineData.valueToSend}
send to  {lineData.slaveDevice} ")

sendFunction[type](lineData.valueToSend,
lineData.slaveDevice, type, MQTTclient)
                case '>':
                    if nameDevice == "time":
                        return
                    if valueDevice >
int(lineData.triggerValue):
                        print("scenario activated")
                        print(data.iloc[i])
```

```
sendFunction[type](lineData.valueToSend,
lineData.slaveDevice, type, MQTTclient)
                case '<':
                    if nameDevice == "time":
                        return
                    if  valueDevice <
int(lineData.triggerValue):
                        print("scenario activated")
                        print(data.iloc[i])

sendFunction[type](lineData.valueToSend,
lineData.slaveDevice, type, MQTTclient)
    return None


def typeDevice(nameDevice):
    devices = pd.read_csv('devices/device_list.csv')
    if nameDevice not in devices.name.values:
        print("Error in typeDevice. device does not exist")
        return None
    else:
        type = devices.loc[devices.name ==
nameDevice].values[0][1]
        return type

def on_message(client, userdata, msg):
    message = msg.payload.decode()
    typeDevice = msg.topic.split(':')[0]
    device = msg.topic.split(':')[1].split('/')[0]
    print(f"newMessage from {msg.topic}")
    match typeDevice:
        case "IoTTemp":

receiveMessage.IoTGasOpenLeak_MessageHandler(message,
device)
            scenario.checkScenario(device, message,
MQTTclient)
        case "IoTIR":
            receiveMessage.IoTIR_MessageHandler(message,
device)
        case "IoTGas":

receiveMessage.IoTGasOpenLeak_MessageHandler(message,
device)
            scenario.checkScenario(device, message,
MQTTclient)
        case "IoTOpen":

receiveMessage.IoTGasOpenLeak_MessageHandler(message,
device)
```

```
            scenario.checkScenario(device, message,
MQTTclient)
        case "IoTLeak":

receiveMessage.IoTGasOpenLeak_MessageHandler(message,
device)
            scenario.checkScenario(device, message,
MQTTclient)

def connectAllTopics():
    devices = pd.read_csv("devices/device_list.csv")
    for i, dev in enumerate(devices.name):
        type = devices.iloc[i].values[1]
        topic = f"{type}:{dev}/fromDevice"
        MQTTclient.subscribe(topic)
        print(f"Subscribe - {topic}")

def settingCient(brokerAddress, port):
    MQTTclient.connect(brokerAddress, port)
    MQTTclient.on_message = on_message
    return MQTTclient

def sendTo_IoTIR(command, nameDevice, typeDevice,
MQTTclient: mqtt.Client):
    toSend = searchIRcommand(command)
    if toSend == None:
        print("Error in sendToIoTIR. Command does not
exist")
        return 1
    topic = f"{typeDevice}:{nameDevice}/toDevice"
    MQTTclient.publish(topic, toSend)
    print("data sent successfully")
    time = datetime.datetime.now()
    with open(f"devices/{nameDevice}/history.csv", "a",
newline='') as csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([time, command])
    scenario.checkScenario(nameDevice, command, MQTTclient)
    return 0

def searchIRcommand(command):
    with open("devices/IRcommands.json", "r") as jsonFile:
        json_str = jsonFile.read()
    json_arr = json_str.split("\n")
    for line in json_arr:
        if line != "":
            data = json.loads(line)
            if data["command"] == command:
                return line
    return None
```

```
def sendTo_IoTSocket(command, nameDevice, typeDevice,
MQTTclient: mqtt.Client):
    topic = f"{typeDevice}:{nameDevice}/toDevice"
    MQTTclient.publish(topic, str(command), retain=True)
    time = datetime.datetime.now()
    with open(f"devices/{nameDevice}/history.csv", "a",
newline='') as csvFile:
        writer = csv.writer(csvFile, delimiter=',')
        writer.writerow([time, command])
    scenario.checkScenario(nameDevice, command, MQTTclient)
    print("data sent successfully")
    return 0

def sendTo_IoTGasOpenLeakTemp(command, nameDevice,
typeDevice, MQTTclient: mqtt.Client):
    now = datetime.datetime.now()
    toSend = f"{command}/{now}/"
    topic = f"{typeDevice}:{nameDevice}/toDevice"
    MQTTclient.publish(topic, toSend, retain=True)
    print("data sent successfully")
    return 0
```

ДОДАТОК В

Код мобільного додатку

```
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout
from kivy.uix.image import Image
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivymd.app import MDApp
from kivymd.uix.card import MDCard
from kivymd.uix.icon import MDIcon

class ElementCard(MDCard):
    def __init__(self, **kwargs):
        super(ElementCard, self).__init__(**kwargs)
        self.md_bg_color = [69/255, 55/255, 86/255, 1]
        self.padding = [dp(15), dp(15), dp(15), dp(15)]
        self.spacing = dp(15)
        self.radius = dp(25)
        self.ripple_behavior = True
        self.image = ''
        self.text = ""
        self.items_count = ""
        self.subtext = ''

        self.orientation = 'vertical'
        self.add_widget(Image(source=self.image))
        layout = BoxLayout(orientation='vertical')
        layout.add_widget(Label(text=self.text, halign='center',
font_style='H6'))
        layout.add_widget(Label(text=self.subtext, halign='center',
font_style='Caption'))
        layout.add_widget(Label(text=self.items_count, halign='center'))
        self.add_widget(layout)

class MyScreen(BoxLayout):
    def __init__(self, **kwargs):
        super(MyScreen, self).__init__(**kwargs)
        self.md_bg_color = [56/255, 40/255, 81/255, 1]
        self.orientation = 'vertical'

        top_layout = BoxLayout(size_hint_y=.25, padding=dp(25))
        top_inner_layout = BoxLayout(orientation='vertical')
        top_inner_layout.add_widget(Label(text="QR", font_style='H4'))
        inner_box_layout = BoxLayout(adaptive_size=True, spacing=dp(10))
        inner_box_layout.add_widget(Label(text="Home", text_size=(None,
None), adaptive_width=True))
        inner_box_layout.add_widget(MDIcon(icon='chevron-down'))
        top_inner_layout.add_widget(inner_box_layout)
        top_layout.add_widget(top_inner_layout)
        top_layout.add_widget(MDIconButton(icon="images/notification.png"))
        self.add_widget(top_layout)

        grid_layout = GridLayout(size_hint_y=.75, cols=2, padding=[dp(15),
dp(15), dp(15), dp(35)], spacing=dp(15))
        grid_layout.add_widget(ElementCard(image="images/calendar.png",
text="Calender", subtext="March, Wednesday", items_count="3 Events"))
```

```
        grid_layout.add_widget(ElementCard(image="images/food.png",
text="Groceries", subtext="Broccoli, Apple", items_count="4 Items"))
        grid_layout.add_widget(ElementCard(image="images/map.png",
text="Location", subtext="Lucy Mao going to Office", items_count=""))
        grid_layout.add_widget(ElementCard(image="images/festival.png",
text="Activity", items_count="", subtext="Rose favorite your Post"))
        grid_layout.add_widget(ElementCard(image="images/todo.png", text="To
Do", subtext="Homework, Design", items_count="4 Items"))
        grid_layout.add_widget(ElementCard(image="images/setting.png",
text="Setting", subtext="", items_count="4 Items"))
        self.add_widget(grid_layout)

class MyApp(MDApp):
    def build(self):
        return MyScreen()
```