МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

На здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук.

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна технологія управління розумним будинком»

Здобувачки групи ІН.м–22 Любченко Діани Юріївни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Діана ЛЮБЧЕНКО

(підпис) (ім'я та прізвище)

Керівник,
доцентка кафедри комп'ютерних
наук, к.ф.-м.н. Галина ОЛЕКСІЄНКО _____

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-22 Любченко Діана Юріївна

1. Тема роботи: «Інформаційна технологія управління розумним будинком»
затверджую наказом по СумДУ від *«6» грудня 2023 року № 1412-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 18 грудня 2023 року.*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз предметної області. 2) Вибір методу рішення. 3) Практична реалізація SHS*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, класифікація інклюзивних людей, постановка й формування завдань дослідження*		
2	*Огляд технологій та принципів для проектування систем розумного будинку*		
3	*Огляд протоколів зв'язку для реалізації системи*		
4	*Розробка системи розумного будинку для квартири*		
5	*Розробка локального блоку керування*		
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 88 стр., 63 рис., 4 додатки, 31 використане джерело.

Обґрунтування. На сьогоднішній день, достатню обізнаність у сфері Smart House System (SHS), мають здебільшого люди великого достатку, так як саме на таких людей націлена рекламна компанія, яка включає в себе презентації великих будинків, які побудовані з врахуванням потаємних механізмів та автоматики в межі однієї екосистеми. Наприклад, просто ввівши на популярному відеохостингу запит «Smart House», користувач побачить вище описану рекламну кампанію.

Актуальність дипломної роботи зумовлена бажанням розсунути рамки сприйняття SHS, для середньостатистичної людини, особливо для інклюзивних людей, число яких під час війни, нажаль, лише зростає. Адже, при елементарному, автоматичному вимиканні світла здорова людини відчуває лише комфорт, а для інклюзивної людини це вже може бути великою допомогою.

Об’єктом дослідження є розумний будинок.

Метою роботи розробка програмного забезпечення та підбір елементної бази для керування розумним будинком.

Методи дослідження: розробка локального збирача інформації за допомогою мови програмування wiring, розробка центрального хабу.

Результати – розроблено систему розумного будинку, яка має можливість керування світлом, відслідковує температуру в різних квартирних зонах, має системи аварійного перекриття природного газу та труби водопостачання, має великий потенціал для майбутніх оновлень

ІНФОРМАЦІЙНА СИСТЕМА, MQTT, CoAP, WIRING, NODEMCU, PYTON, IoT, ІНКЛЮЗИВНІСТЬ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IT – інформаційні технології;

SHS – Smart House System;

API – Application Programming Interface;

ATEX – EU-type examination certificate;

IoT – Internet of Things;

OSI – The Open Systems Interconnection model;

CoAP – Constrained Application Protocol;

REST – Representational State Transfer;

URI – Uniform Resource Identifier;

MQTT – Message Queuing Telemetry Transport;

AMQP – Advanced Message Queueing Protocol;

XMPP – Extensible Messaging and Presence Protocol;

DDS – Data Distribution Service;

QoS – Quality of Service;

WPA – WI-FI Protected Access;

GPIO – General Purpose Interrupt Output.

ВСТУП

Актуальність. На сьогоднішній день, достатню обізнаність у сфері SHS, мають здебільшого люди великого достатку, так як саме на таких людей націлена рекламна компанія, яка включає в себе презентації великих будинків, які побудовані з врахуванням потаємних механізмів та автоматики в межі однієї екосистеми. Наприклад, просто ввівши на популярному відеохостингу запит «Smart House», користувач побачить вище описану рекламну кампанію.

Актуальність дипломної роботи зумовлена бажанням розсунути рамки сприйняття SHS, для середньостатистичної людини, особливо для інклюзивних людей, число яких під час війни, нажаль, лише зростає. Адже, при елементарному, автоматичному вимиканні світла здорова людини відчуває лише комфорт, то для інклюзивної людини це може бути великою допомогою.

Об’єкт дослідження. Розумний будинок.

Предмет дослідження. Процес розробки системи розумного будинку, з підбором необхідних компонентів, з огляду на потреби інклюзивного користувача.

Гіпотеза. Реалізація системи з графічним інтерфейсом, з можливістю легкого додавання нових функцій, націлених на конкретних, унікальних користувачів, можна досягнути за допомогою системи мікроконтролерів.

Наукова новизна. На відміну від існуючих систем розумного будинку, описана у даній роботі система дозволить використовувати ієрархічну структуру для сепарування обов’язків, де це доцільно. Одночасно зберігаючи переваги її найближчих конкурентів.

Структура. Дане робота складається зі вступу, огляду основних концепцій побудови систем керування розумним будинком, постановки задачі проектування, вибір методики рішення поставленої задачі, створення структурної схеми SHS, вибір елементної бази, розробки керуючого коду для різних частин системи, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих SHS

Apple HomeKit – це платформа для розумного будинку, розроблена Apple, яка надає засоби для керування та автоматизації побутовими пристроями та системами вдома за допомогою пристроїв, які працюють лише на базі iOS, таких як iPhone, iPad, Apple TV або Apple Watch, а також голосових команд через Siri [1]. HomeKit створено з фокусом на безпеці та конфіденційності користувача, інтерфейс користувача на різних пристроях виробництва Apple зображений на рис. 1.1.

Рисунок 1.1 – Інтерфейс Apple HomeKit на різних пристроях Apple [1]

Apple HomeKit дозволяє користувачам керувати різними побутовими пристроями, такими як освітлення, термостати, дверні замки, камери безпеки і т.д., за допомогою одного додатку на пристроях Apple. Ви можете вимикати світло, налаштовувати температуру, відкривати двері та багато іншого безпосередньо зі свого смартфону чи планшету.

Голосовий асистент Apple Siri, інтегрований в HomeKit, це дозволяє користувачам використовувати голосові команди для керування підключеними пристроями. Наприклад, коли користувач скаже «Hey Siri, turn off the light in the bedroom» – система вимкне лампу, яка записана як спальня. Нажаль, на сьогодні

голосовий помічник Siri не розуміє українську мову, тому спілкування з ним можливе англійською або російською.

HomeKit підтримує створення автоматизованих правил, які активуються на підставі певних умов. Користувач може налаштувати увімкнення світла, при відкритті вхідних дверей, або реагувати на рух, при наявності датчиків руху. Це допомагає зекономити енергію і значно підвищити комфорт.

Компанія Apple ставить великий акцент на безпеку та конфіденційність. Дані, які генеруються і обмінюються між пристроями та додатками HomeKit, зашифровані та зберігаються локально на пристрої користувача, а не на серверах Apple чи в хмарних сервісах.

HomeKit підтримує різні бездротові стандарти, такі як Wi-Fi, Bluetooth і Zigbee. Для того щоб пристрій був сумісним з HomeKit, виробник повинен пройти сертифікацію від Apple. Це гарантує, що пристрій відповідає стандартам безпеки та функціональності HomeKit. Це робить його сумісним з широким спектром побутових пристроїв, на рис. 1.2 зображене клеймо сумісності.

Рисунок 1.2 – Приклад клейма сумісності з HomeKit [1]

HomeKit підтримує інтеграцію з певними конкурентами та іншими платформами розумного будинку, але рівень інтеграції може бути обмеженим. Наприклад, користувач можете інтегрувати деякі термостати Ecobee, освітлення Philips Hue, системи безпеки Samsung SmartThings та інші пристрої в систему HomeKit, але функціональність може відрізнятися в залежності від конкретних моделей.

Samsung SmartThings – це платформа розумного будинку, розроблена компанією Samsung, яка дозволяє користувачам керувати та автоматизувати побутові пристрої з використанням смартфонів та інших пристроїв [2], інтерфейс користувача на різних пристроях зображений на рис. 1.3.

SmartThings працює з різними пристроями різних виробників, зокрема з пристроями на базі Zigbee, Z-Wave, Wi-Fi та Bluetooth, це означає, що користувач може підключати широкий спектр побутових пристроїв, таких як смарт-лампи, датчики, камери, дверні замки, термостати і т.д..

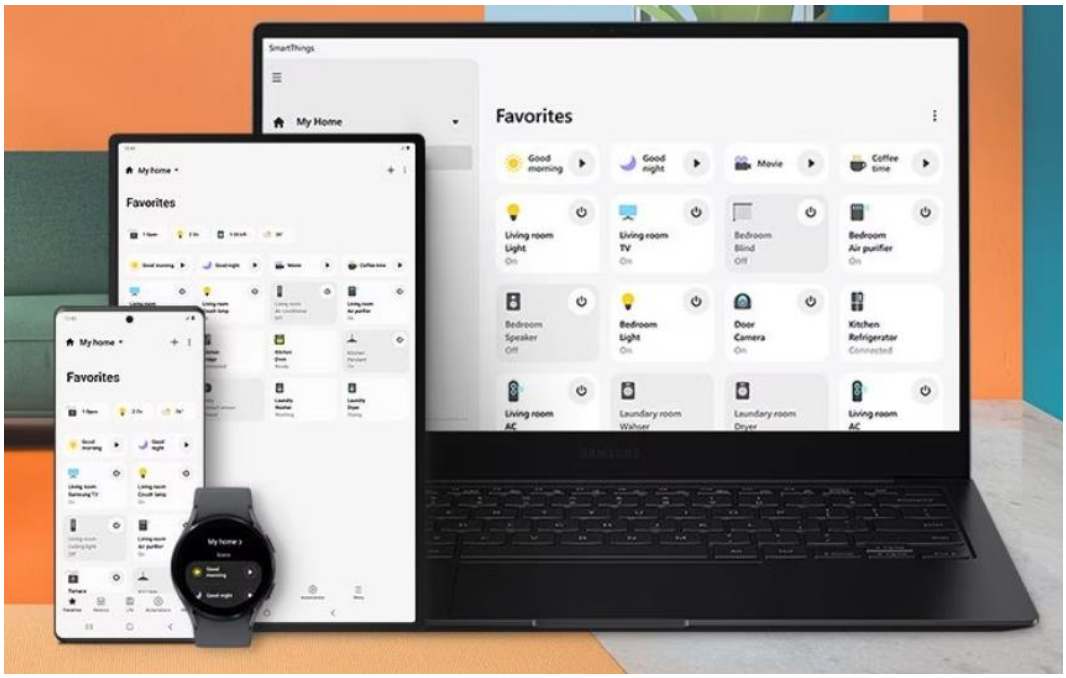

Рисунок 1.3 – Інтерфейс Samsung SmartThings [2]

На відміну від Apple HomeKit, додаток Samsung SmartThings доступний для смартфонів на платформах iOS та Android, що дозволяє користувачам керувати підключеними пристроями, створювати розклади, налаштовувати автоматизацію та отримувати сповіщення про події вдома. Також, ключова відмінність SmartThings від HomeKit, необхідність мати смарт-хаб – SmartThings який підключається до домашньої мережі Wi-Fi користувача і взаємодіє зі смарт-пристроями. Це центральний контрольний пункт, який забезпечує комунікацію між всіма пристроями та додатками SmartThings.

Дані SmartThings зазвичай зберігаються в хмарі, що дозволяє отримати доступ до системи з будь-якої точки з'єднаної з інтернетом. Важливо зауважити, що Samsung прагне забезпечити безпеку та захист особистих даних користувачів тому приділяє увагу цьому аспекту, при розгортанні серверів.

SmartThings дозволяє створювати автоматизовані сценарії та правила, приклад яких зображений на рис. 1.4. Наприклад, при виході з дому вимикати всі джерела світла, зменшувати температуру термостату та включати систему безпеки. Система SmartThings інтегрується з різними голосовими асистентами, такими як Amazon Alexa та Google Assistant. Це дозволяє користувачам керувати побутовими пристроями за допомогою голосових команд. Мовне питання SmartThings аналогічне HomeKit – лише англійська та російська мови.

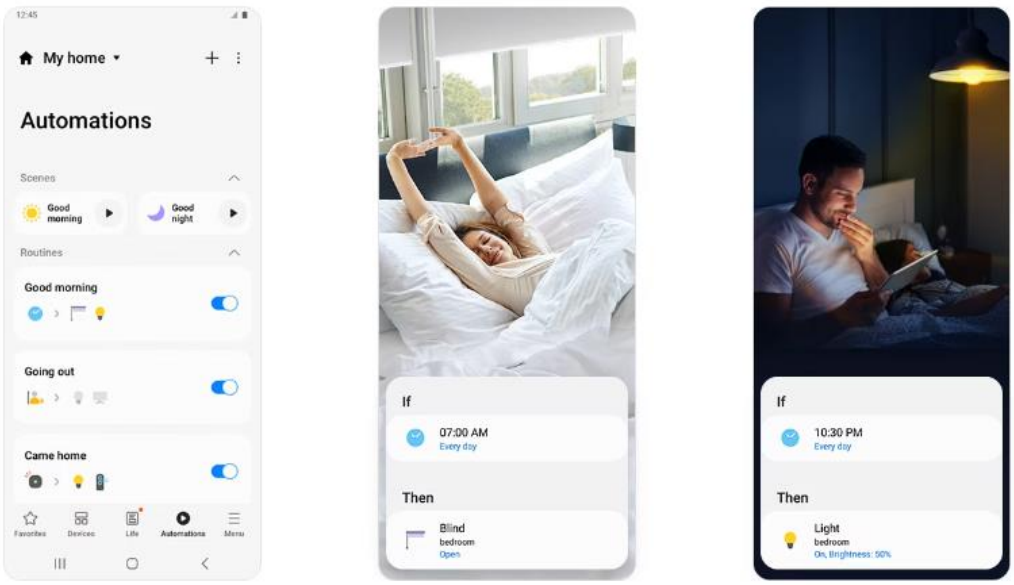

Рисунок 1.4 – Налаштування правил в програмі SmartThings [2]

Система надсилає сповіщення користувачеві про певні події в будинку, такі як виявлення руху, відкриття дверей, спрацювання датчика диму тощо. Це дозволяє користувачам вчасно реагувати на події вдома.

Samsung постійно розширює функціональність SmartThings шляхом співпраці з новими виробниками та розробниками додатків, що робить екосистему більш універсальною та розширеною. Свою зацікавленість Samsung

показує пропонуючи власне API SmartThings Developer Workspace, яке дозволяє розробникам створювати свої власні рішення для розумного будинку та інтегрувати їх з платформою SmartThings. Виробники техніки, які співпрацюють з Samsung, позначають власну техніку відповідним клеймом, яке зображене на рис. 1.5, для кращої ідентифікації продуктів клієнтами торгової марки.

Рисунок 1.5 – Приклад клейма сумісності з SmartThings [2]

SmartThings підтримує велику кількість пристроїв від різних виробників, що робить його універсальним для підключення різноманітних пристроїв до однієї системи. Він інтегрується з багатьма відомими виробниками, такими як Philips Hue для освітлення, Ecobee для термостатів, Ring для камер безпеки, і іншими.

Порівнюючи SHS від Samsung з HomeKit від Apple можна побачити суттєві відмінності, одна з яких – децентралізація. Так як Apple має власну операційну систему, на яку, простим користувачам, неможливо поставити сторонні додатки, вони відмовились від смарт-хабу. Його роль може виконуватись будь-яким потужним пристроєм – телевізор, смартфон або планшет. Samsung, в свою чергу рекомендують ставити смарт-хаб, який буде оброблювати всю інформацію власноруч та відправляти її в свою хмару.

1.2 Класифікація інклюзивних людей з боку законодавства

На мою думку, SHS можуть значно покращити якість життя для інклюзивних людей, тому в межах дипломної роботи варто в першу чергу опиратись на їх потреби, бо те що для здорової людини звичайна дія, для інклюзивної людини може бути неможливим для повторення.

Оскільки поняття інклюзії охоплює широкий спектр аспектів, то неможливо врахувати потреби всіх, необхідно конкретизувати його за допомогою конкретних систем та заходів, що враховують потреби осіб з обмеженими можливостями. Держава розподіляє цих громадян на п'ять категорій інвалідності, щоб краще адаптувати соціальні та підтримуючі програми для їхніх унікальних потреб [3]:

- фізичні недоліки (порушення опорно-рухового апарату);
- порушення інтелекту і психічні захворювання;
- порушення функцій слуху (слабочуючі та глухі);
- порушення функцій зору (слабозорі та сліпі);
- порушення роботи внутрішніх органів.

Опираючись на категорії, які нам описує законодавство в такому вигляді [4], неможливо сформувати список систем для задоволення проблем цих людей. Давайте проведемо новий розподіл людей за ступенем важкості їхньої інвалідності:

- I група – особа, яка повністю втратила працездатність і вимагає постійного догляду;
- II група – особа здатна до самообслуговування, але не здатна до праці в звичайних виробничих умовах;
- III група – людина може працювати в спеціальних, полегшених умовах.

Отримавши класифікацію такого виду, можемо побудувати інфографіку, взявши конкретні дані за звітом соціального захисту МОЗ за 2022 рік. Складемо зручну таблицю з кількістю осіб з інвалідністю та наведемо дані в табл. 1.1.

Таблиця 1.1 – Кількість інклюзивних людей на початок року [3], [4]

Рік / Тис.	2001	2006	2011	2016	2019	2020	2021	2022	2023
I група	337,7	337,7	310,5	250,3	226,3	222,3	215,0	207,2	204,9
II група	1337,0	1128,4	1078,7	919,0	896,1	900,8	897,1	886,7	888,7
III група	768,5	906,5	1155,7	1291,2	1375,7	1416,0	1449,1	1469,7	1472,0
Загалом	2443,2	2372,6	2544,9	2460,5	2498,1	2539,1	2561,2	2563,6	2565,6

Для зручного розуміння табличних значень, візуалізуємо їх в графік кількості інклюзивних людей та зобразимо на рис. 1.6.

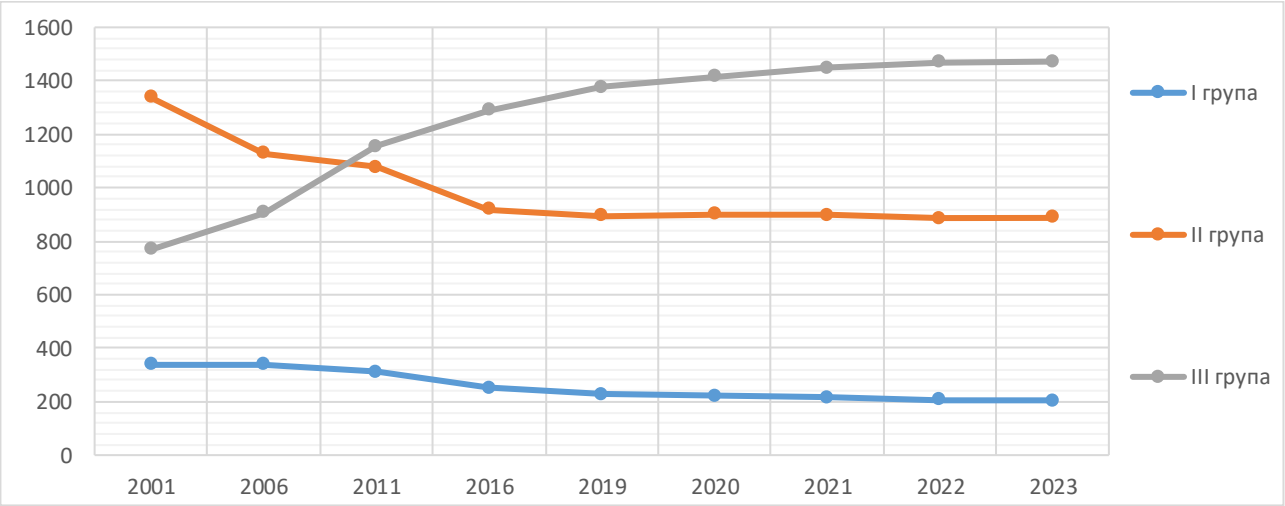

Рисунок 1.6 – Кількість інклюзивних людей на початок року

Проаналізувавши графік можна зрозуміти, що протягом часу спостерігається зменшення кількості осіб із II та I групами інвалідності, але кількість осіб із II та III групи виявляється значно більше, ніж осіб із I групою. Для більш широкого розуміння статистичних даних за 2022 рік та створення образу середньостатистичної особи, яка стає інклюзивною, занесемо у табл. 1.2. причини інклюзивності, місце проживання особи, її стать та вік.

Таблиця 1.2 – Кількість визнаних уперше інклюзивних людей у 2022 році

Причина	Жінки – 18-54 роки Чоловіки – 18-59 років		Жінки – 55 років і старші Чоловіки– 60 років і старші	
	Міська місцевість	Сільська місцевість	Міська місцевість	Сільська місцевість
Від нещасного випадку чи профзахворювання	1524	581	292	91
Від загального захворювання	58100	34831	23336	10068
З числа військово-службовців	3776	1541	417	82
З дитинства	6368	4170	64	2
Визнано особами з інвалідністю	69768	41123	24109	10243

Для кращого сприйняття, наведеної в таблиці вище, інформації – візуалізуємо її, на рис. 1.7 зображена інфографіка, яка показує співвідношення причин отримання інклюзії в міській місцевості.

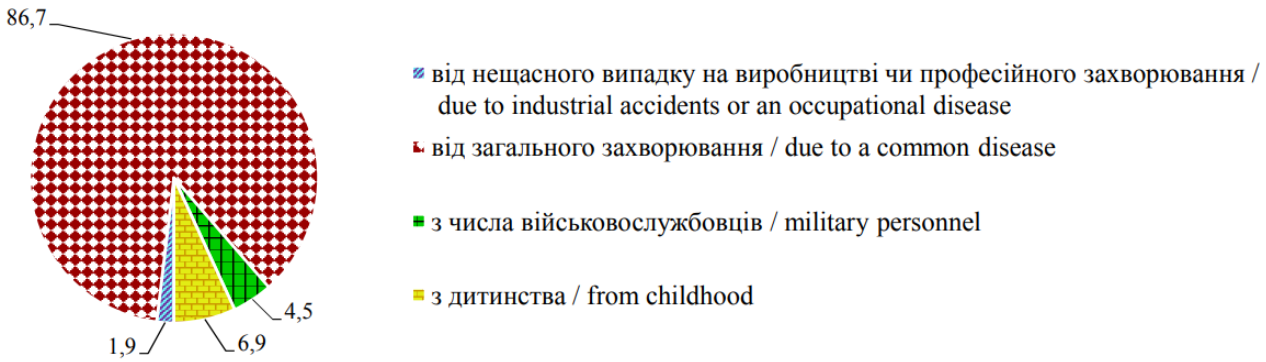

Рисунок 1.7 – Причини отримання інклюзії у міській місцевості [4]

Візуалізуємо також дані, які стосуються сільської місцевості, на рис. 1.8 зображена інфографіка, яка показує співвідношення причин отримання інклюзії у сільській місцевості.

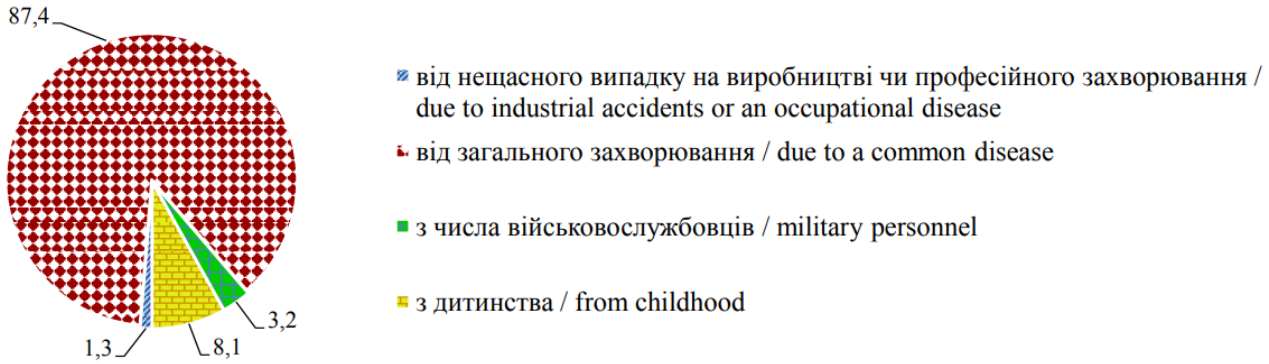

Рисунок 1.8 – Причини отримання інклюзії у сільській місцевості [4]

На підставі наведених вище даних, можна сформулювати характеристику середньостатистичної інклюзивної особи, яка може бути використана для визначення вимог до розроблювальної системи:

- особа частіше проживає в квартирі чим в приватному будинку;
- вік особи знаходиться в діапазоні 18-59 років, вона отримала статус інваліда протягом свого життя;
- стать особи не важлива;
- статус інвалідності отримала під час загального захворювання;
- особа має III групу інвалідності;
- ця особа вже перехворіла або ж ще лікується.

Розумний дім представляє собою житловий простір, в якому пристрої пов'язані між собою, що дозволяє власнику контролювати їх стан, управляти ними та створювати правила їх роботи. Система розумного будинку надає контроль особам із обмеженими можливостями, допомагаючи їм насолоджуватися самостійним життям, полегшуючи повсякденні обов'язки та зменшуючи турботи їхніх близьких. Знаючи загальний портрет, середньостатистичної інклюзивної людини та можливості сучасного часу в області SHS, можна сформулювати конкретні вимоги до проектованої системи, для закриття базових потреб людини. Використання сучасних технологій дозволяють створювати унікальні прилади, які можу допомогти інклюзивним людям почувати себе краще.

1.3 Формування адаптивних розсилок

Оскільки більшість процесів як для виробничих, так і для побутових систем мають періодичний характер, то вивчення циклічних характеристик даних, що збираються пристроями IoT, є виправданим і дасть змогу краще розуміти суть явищ, що в них відбуваються.

Циклічність – це властивість роботи системи, яка прагне зберегти свій стан у межах рівноваги. Для дослідження в роботі застосовано метод дискретного перетворення Фур'є і на основі розрахованих параметрів гармонійного ряду зроблено висновок про частотні характеристики даних. Подібний підхід набув свого застосування в роботах з дослідження частотно-часових характеристик різних послідовностей даних.

Для проведення досліджень, в рамках однокімнатної квартири, застосуємо датчики температури. Пристрої IoT встановлені в жилому приміщенні з обладнанням, що мають певну тепловіддачу.

Розглянемо дискретні моменти часу спостереження $\{t_n\}$, де $n = [0, N-1]$, кількість відліків. Час між спостереженнями $\Delta t = (t_{n+1} - t_n) + \tau$, де τ – затримки передавання даних. Значення τ настільки мале, що період дискретизації можна вважати постійним, рівним $T = (t_{n+1} - t_n)$, тривалість спостережень дорівнює NT. Уявімо послідовність спостережуваних даних як функцію дискретного аргументу, що набуває довільних позитивних значень у дискретні проміжки часу nT – $x(t) = \{x(nT)\}$, де $x(nT)$ – n-й результат спостережень в момент nT.

Функція x(t) – періодична, у загальному випадку властивість періодичності досягається шляхом повторення даних з періодом NT. Для періодичної функції може бути виконано спектральне розкладання у вигляді ряду Фур'є й отримано дискретний спектр X(k), $k \in Z$, що складається з гармонік, кратних $\Delta\omega = 2\pi/T$. Оскільки вихідна функція x(t) – дискретна, достатньо визначити спектральні значення для k = [0, N-1]. Коефіцієнти розкладання в ряд Фур'є можуть бути

отримані за формулою (1.1) дискретного перетворення, яке залежить тільки від індексу вхідного сигналу.

$$X(k) = \sum_{n}^{N-1} x(n) e^{-i\frac{2\pi}{N}nk} \quad (1.1)$$

Перетворення дає змогу за N вимірювань значень x(t), отримати N спектральних відліків на одному періоді повторення спектра X(k).

Застосуємо до наших результатів спостережень за кожним із джерел дискретне перетворення Фур'є, розглянемо модулі комплексних чисел отриманих коефіцієнтів. За ними будемо робити висновок про значення амплітуд і частот гармонік. На основі отриманого результату зробимо висновок про частоту зміни спостережуваних подій і сформуємо налаштування для видачі даних, які дадуть змогу обирати значущі події. Таку послідовність дій потрібно виконувати для великого набору даних із заданою періодичністю, щоб оперативно змінювати налаштування часу подій видачі даних. Реалізація цього підходу дасть змогу автоматично формувати адаптивні підписки.

Пояснимо підхід до формування адаптивних розсилок на прикладі. Візьмемо вибірку даних, що є результатом вимірювань у N відліках (рис. 1.9).

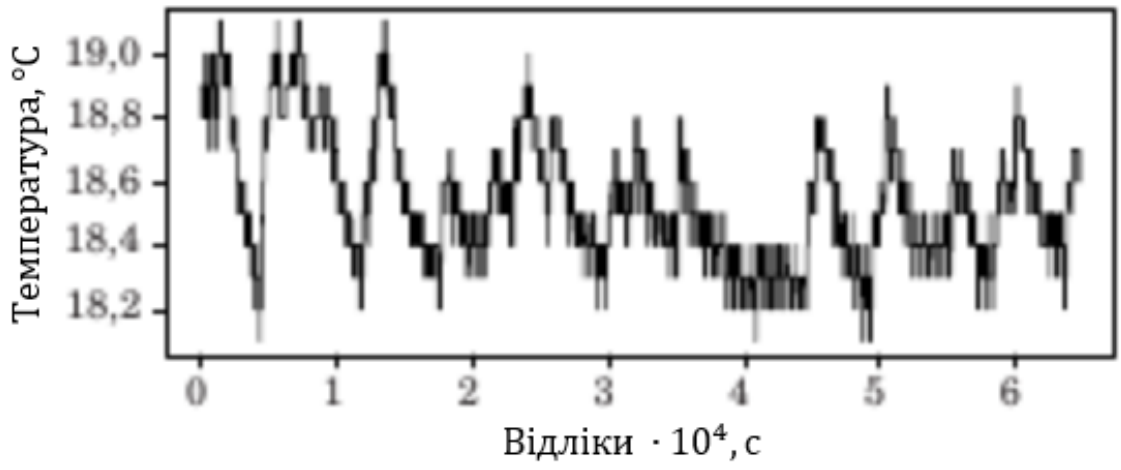

Рисунок 1.9 – Дані у відліках

Застосуємо дискретне перетворення Фур'є (формула 1.1) і побудуємо частотний спектр даних. Вагові коефіцієнти розкладання є комплексним спектром періодичного сигналу (нашої вибірки даних). За коефіцієнтами Фур'є побудуємо mod(X(k)), де mod – функція, що повертає дійсну частину комплексного числа. Графік результатів показано на рис. 1.10.

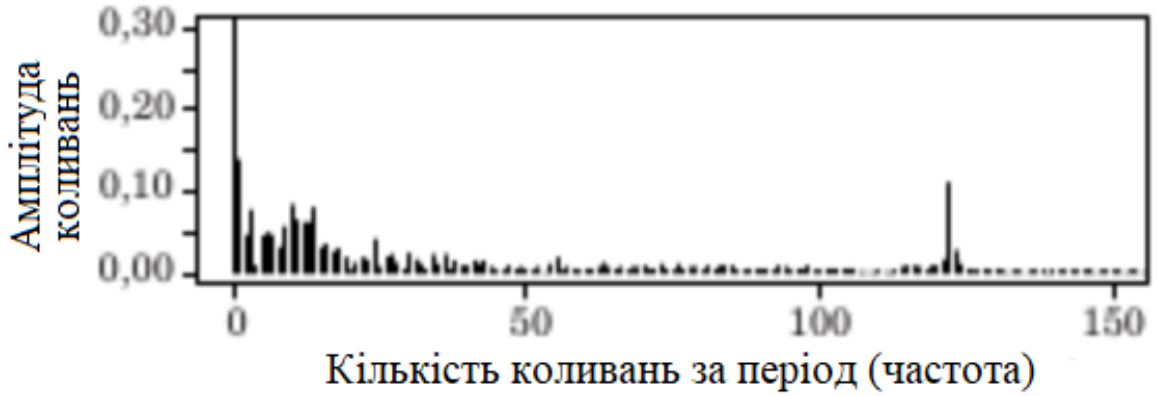

Рисунок 1.10 – Частотний спектр даних

Побудований спектр – дискретний і складається з гармонік, кратних Δω. Його перша гармоніка при k = 0 є основною частотою сигналу і відображає постійну складову даних, для налаштування адаптивних розсилок її не враховуємо, решта частот дискретного спектра (за k≥1) є гармоніками сигналу. Потрібно знайти значення гармонік, амплітуда коливання яких показує динаміку змін у спостережуваних даних. Задамо поріг ε для вибору значень амплітуд, які будемо враховувати під час аналізу. Мінімальне ε може бути визначено з характеристик точності використовуваних датчиків IoT. Виберемо коефіцієнти розкладання X(k) такі, що Re(X(k))>ε і було виконано умову, результат відобразимо у формулі (1.2).

$$X(k-1) \leq X(k) \geq X(k+1) \tag{1.2}$$

За номером коефіцієнта k, визначимо період коливань, що відповідає обраним X(k), розділивши інтервал спостережень NT на номер відліку за допомогою формули (1.3).

$$P(k) = \frac{NT}{k} \tag{1.3}$$

Частоту коливань будемо розраховувати за формулою (2.4).

$$F(k) = \frac{1}{P(k)} \tag{1.4}$$

Виберемо для k, що задовольняють умову формули (1.2), максимальну частоту коливань спектра сигналу, що визначає період дискретизації Fd (формула (1.5)).

$$Fd = max\, F(k) \tag{1.5}$$

Скористаємося теоремою Котельникова [5], згідно з якою, якщо спектр сигналу обмежений частотою F_{max}, то його можна однозначно відновити за його дискретними відбитками, узятими через інтервали часу, з частотою, що щонайменше, удвічі перевищує максимальну частоту сигналу, який ми хочемо виміряти, тобто на кожне коливання сигналу (зміну виміряних значень) повинно припадати щонайменше, два відліки. Звідси випливає, що для спостереження за даними достатньо виконувати вимірювання з частотою дискретизації, дане твердження у вигляді формули (1.6).

$$F_{max} \geq 2Fd \tag{1.6}$$

Мінімальний період дискретизації визначається обернено максимальній часті, формула (1.7).

$$P_{min} = \frac{1}{F_{max}} \tag{1.7}$$

У розглянутому прикладі значення k ∈ {10,14,122} задовольняють умову (1.2) коефіцієнтів розкладання (1.1) з частотами, що відповідають періодам коливань (2.3) P(k) ∈ {6480,4632,525} секунд відповідно. За формулою (1.4)

визначаємо частоту, за формулою (1.5) вибираємо максимальну частоту коливань спостережуваних даних Fd=1/525 Гц і частоту дискретизації можемо вибрати з (1.6) – F_max=Fd·2=2/525 Гц. За формулою (1.7) отримуємо період дискретизації P_{min} = 262,5 с. Знайдений період P_{min} застосовується для налаштування брокера, що забезпечує спостереження за подіями відповідно до динаміки їх змін.

Запропонований підхід можна застосувати для налаштування видачі даних від пристроїв IoT, зокрема для енергонезалежних джерел. Зменшення частоти пересилання даних від таких пристроїв до брокера і збільшення тривалості періодів спокою подовжить час їхнього автономного функціонування. Вибір параметрів дискретизації вимірювань виконується на основі статистики їхньої роботи за формулою (1.7), розглядається тривалий інтервал спостережень. Для уникнення втрати інформативності в моменти неактивності пристроїв IoT під час вибору інтервалу дискретизації слід враховувати мінімальний період вимірювань за максимальної швидкості протікання спостережуваних процесів.

У загальному випадку швидкість може бути отримана на основі зібраної статистики, під час імітації критичних станів, формула (1.8)

$$S_{max} = max_{k=1,N} \frac{|X(k) - X(k-1)|}{\Delta t} \tag{1.8}$$

Для кожного спостережуваного показника задамо значення X_{min} його допустимої зміни, тоді період обчислюється на основі швидкості протікання процесів, формула (1.9)

$$P_S = \frac{X_{min}}{S_{max}} \tag{1.9}$$

Результуючий період дискретизації для роботи пристроїв IoT визначається з результатів розрахунку формул (1.7) і (1.9).

$$P_d = \min(P_s, P_{min}) \tag{1.10}$$

Розглянемо приклад визначення періоду дискретизації для датчиків IoT вимірювання температури навколишнього середовища. Пристрої IoT виконують вимірювання і передають дані брокеру кожні 10 секунд. Припустимо, є умова, що потрібно підтримувати температуру у приміщені у діапазоні 18-24°C. Покажемо, яким має бути режим видачі даних за допустимої зміни на 2 °C.

Швидкість протікання процесів обчислюється з даних, отриманих як за стандартного режиму роботи, так і за імітації не штатної зміни температури, фрагмент графіка вихідних даних наведено на рис. 1.11.

Рисунок 1.11 – Дані для розрахунку максимальної швидкості протікання процесів

За формулою (2.8) максимальна швидкість зміни температури становить 0,04 °C/c, за формулами (1.9) та (1.10) період дискретизації становить 50 с. Проведений розрахунок дає змогу скоротити кількість переданих даних і періодів активності пристроїв IoT у 5 разів.

Наразі розглядаються варіанти впровадження функцій сервісу в код брокера. Це можливо, оскільки в розрахунках використовується швидке дискретне перетворення Фур'є, що має складність $O(N \cdot \log N)$, яка не призводить до істотних обчислювальних витрат. Цей підхід може бути застосований для процесів, що протікають повільно, де період розсилок може бути істотно збільшений, що потребуватиме збільшення періоду вибірки аналізованих даних.

1.4 Постановка завдання проектування

Метою роботи є спроба розробки цілої SHS, для демонстрації можливостей та зацікавлення нової аудиторії, на прикладі SHS для інклюзивної людини. Прототип повинен увібрати в себе переваги конкурентів, бути однаково корисним як для здорових користувачів, так і для інклюзивних, мати як легкий спосіб налаштування, так і більш складний для імплементації унікальних, користувацьких функцій.

Водночас, прототип не повинен бути надто насиченим функціями і, відповідно, не повинен мати високу вартість. Його мета полягає в тому, щоб ознайомити людину із сучасними можливостями технологій у сфері IoT за доступну ціну. Проектована система має задовольняти основні потреби користувача з інвалідністю, який живе в міській місцевості, в багатоквартирному будинку:

- керування освітленням в 3-х кімнатах, з веб-інтерфейсу;
- датчик відкривання 2-х вікон;
- розумний вхідний замок;
- керування опаленням з веб-інтерфейсу;
- відслідковування температури, вологості в 3-х кімнатах;
- відслідковування протікання рідин та газів в 2-х кімнатах;
- сумісність з приладами конкурентів;
- автономність системи;
- web-інтерфейс, для керування будинком.

Для досягнення поставленої мети, необхідно розглянути основні підходи до побудови SHS та обрати найбільш підходящий під наші завдання. На основі вибраного підходу обрати апаратне та програмне забезпечення або ж розробити своє. Також потрібно розглянути популярний мікроконтролер, який дозволяє створювати IoT.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Протокол обміну повідомленнями SHS

Для того, щоб побудувати SHS, яка буде задовольняти всі потреби користувача та відповідати постановці завдання – потрібно визначитися з мережевим протоколом, який буде керуватиме передачею даних і контролюватиме безпеку доставки інформації.

Загальна формула встановлення телекомунікацій у всьому світі представлена моделлю The Open Systems Interconnection model (OSI) [6]. Модель розділяє структуру на 7 різних шарів, як показано на рис. 2.1.

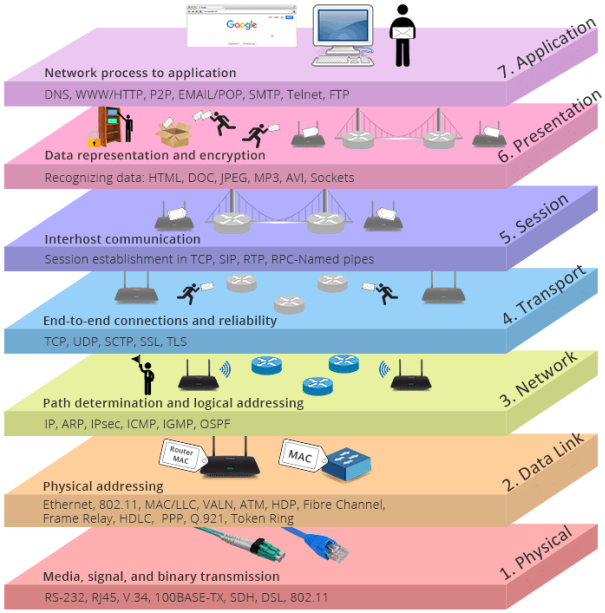

Рисунок 2.1 – Модель OSI [6]

Найвищі рівні 5-7 (сеансовий, презентаційний і прикладний рівень), створені для обробки даних. Прикладний рівень, який є останнім рівнем, буде єдиним рівнем для взаємодії з кінцевими користувачами, саме тут розташовані спеціалізовані додатки для побудови SHS: Constrained Application Protocol (CoAP), Message Queuing Telemetry Transport (MQTT), Advanced Message Queueing Protocol (AMQP), Extensible Messaging and Presence Protocol (XMPP), Data Distribution Service (DDS).

Ще один прикладний рівень, який зазвичай використовується в телекомунікаційних системах – HTTP. Однак, він не підходить для додатків IoT через свої характеристики:

- може підтримувати лише зв’язок «один на один», тобто обслуговуватиме лише одного клієнта для одного брокера/сервера;
- високе енергоспоживання, для кожного встановленого з’єднання «один-до-одного», базове з’єднання TCP також має бути відкритим. Це створить велике навантаження на сервер, отже споживатиме більше енергії, ніж може витримати периферійний пристрій.

Розглядати кожен з них детально, немає необхідності, бо DDS та XMPP зазвичай використовується в промислових частинах IoT, а AMQP вимагає складного дротового підключення, тому розглянемо лише CoAP та MQTT.

2.1.1 Constrained Application Protocol (CoAP)

Протокол CoAP був розроблений робочою групою The Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) у червні 2014 року, його стандарт описаний у документі RFC 7252 [7].

Цей протокол призначений для взаємодії простих пристроїв, таких як датчики малої потужності, вимикачі та клапани, які можуть бути керовані або контрольовані віддалено через інтернет. Вони широко використовуються в області IoT, а взаємодія між ними породжує інформаційний обмін, відомий як між машинна взаємодія. Ці пристрої часто відомі як пристрої з обмеженими ресурсами, оскільки вони мають обмежені енергетичні ресурси, обсяг пам'яті та потужності.

Протокол CoAP [8] забезпечує ефективну взаємодію з такими пристроями, забезпечуючи низькі енерговитрати та передачу обмеженого обсягу повідомлень. Важливою особливістю цього протоколу є його сумісність з протоколом HTTP, що дозволяє взаємодіяти з сукупністю пристроїв IoT, що формують мережу, з мережею інтернет.

Давайте розглянемо конкретний приклад пристрою з обмеженими ресурсами, такого як датчик. У користувача в приміщенні може бути встановлено кілька датчиків, якими він може керувати через інтернет або безпосередньо через мережу з обмеженими ресурсами, якщо його пристрій використовує протокол CoAP (рис. 2.2).

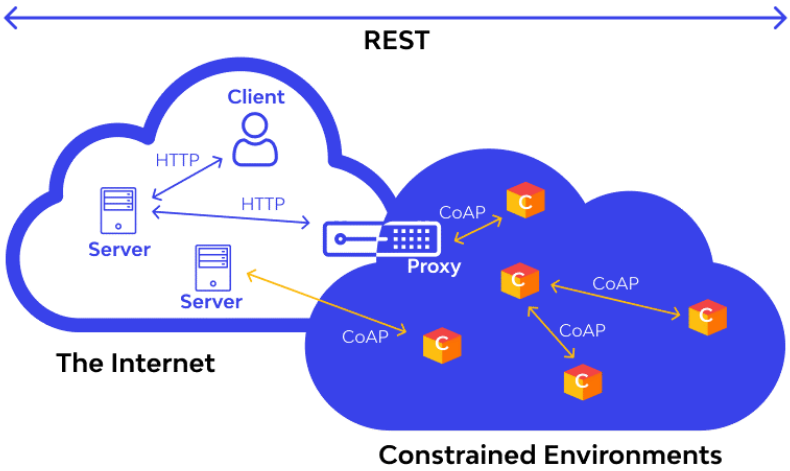

Рисунок 2.2 – Взаємодія з використанням протоколу CoAP [8]

На рисунку 2.2 представлений ланцюг взаємодії «Client-Server-Proxy-Sensor», що ілюструє спосіб взаємодії між користувачем пристрою (HTTP-клієнта) та CoAP-датчиком через інтернет. HTTP-клієнт генерує запити та направляє їх на сервер, у випадку відсутності необхідної інформації на сервері, він звертається до CoAP-проксі. Тут CoAP-проксі виступає як пристрій, що з'єднує мережу інтернет на основі протоколу HTTP та мережу з обмеженими ресурсами, яка використовує протокол CoAP, в якій розташовані датчики. Проксі трансформує повідомлення однієї мережі (протокол HTTP) у повідомлення, зрозумілі для іншої мережі (протокол CoAP).

Якщо запит від пристрою безпосередньо потрапляє в мережу з протоколом CoAP, він направляється до CoAP-датчика (див. рис. 2.2 - ланцюг «Сервер-Датчик»). У цьому випадку сервер отримує всі запити і подальшим чином відправляє їх індивідуально пристрою, чекаючи відповідей.

Протокол CoAP був створений з дотриманням принципів REST-архітектури. Representational State Transfer (REST) представляє собою архітектурний підхід, який дозволяє будувати програми на основі певної групи методів:

- GET, виконує запит для отримання ресурсів та повертає інформацію, яка відповідає ресурсу, зазначеному у Uniform Resource Identifier (URI). Ресурс може бути, наприклад, датчиком;
- PUT, встановлює новий стан ресурсу.;
- POST, відповідає за зміни в стані ресурсу ;
- DELETE, використовують для видалення активних можливостей ресурсу.

У квітні 2017 року був опублікований документ RFC, який розширює протокол CoAP новими методами [9]:

- FETCH, виконує запит для отримання часткової інформації про ресурс на основі параметрів запиту;
- PATCH, відповідає за часткову зміну стану ресурсу.

Ці методи були додані до протоколу з метою задоволення потреб певних додатків, які мають взаємодіяти з ресурсами, отримуючи доступ або змінюючи їх частково, а не повністю.

Протокол CoAP використовує за замовчуванням протокол UDP як транспортний, що дозволяє зменшити розмір службових даних і підвищити ефективність роботи. У випадках, коли це необхідно, можуть використовуватися також протоколи TCP або SCTP. Бувши протоколом прикладного рівня, CoAP використовує шифрування повідомлень за допомогою протоколу Datagram Transport Layer Security (DTLS) [10], особливо в тих випадках, коли потрібно забезпечити безпечне з'єднання.

У протоколі CoAP визначено лише чотири типи повідомлень: Confirmable, Non-confirmable, Acknowledgement та Reset. Всі повідомлення кодуються у бінарному вигляді, формат якого показано на рис. 2.3. Повідомлення

починається з заголовка фіксованого розміру в 4 байти, за яким слідує маркер, опції і корисне навантаження, яке займає всю частину дейтаграми, що залишилася.

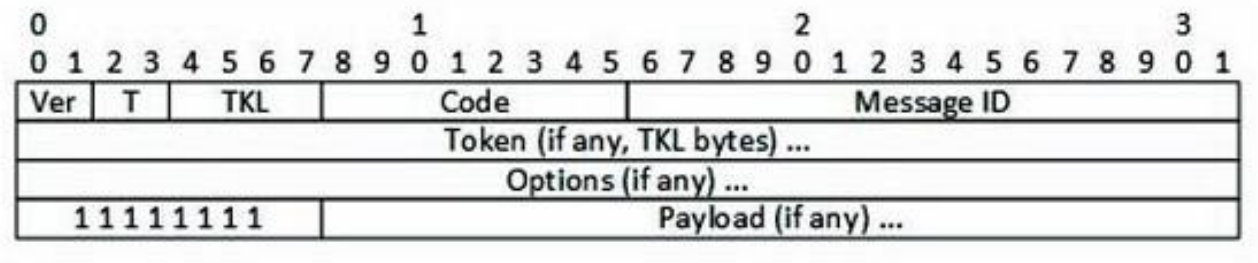

Рисунок 2.3 – Формат повідомлення

У повідомленні поля визначаються наступним чином:

- версія (Ver), двох бітне ціле число, що вказує на номер версії протоколу CoAP;
- тип (T), двох бітне ціле число, вказує на тип повідомлення: Confirmable (0), Non-confirmable (1), Acknowledgement (2) або Reset (3);
- довжина маркера (TKL), чотирьох бітне ціле число, вказує на довжину поля маркера (Token) змінної довжини (від 0 до 8 байт). Довжини від 9 до 15 зарезервовані;
- Код (Code), восьми бітне ціле число, розділене на три бітний та п'яти бітний клас.

Значення «dd» є фіксованими табличними величинами, які вказують на тип запиту, якщо повідомлення є запитом (наприклад, 0.01 представляє запит GET, 0.02 - запит POST). У випадку відповіді, ці значення вказують на код відповіді (наприклад, 2.05 - відповідь із запитаною інформацією (Content), 4.04 - помилка, яка виникає, коли запитуваний пристрій відсутній або тимчасово недоступний (Not Found)). Повний список значень поля Code можна знайти в RFC7252, розділ 12.1.2. [11].

Ідентифікатор повідомлення (Message ID) є шістнадцяти бітовим цілим числом і виступає унікальним ідентифікатором повідомлення. Message ID

визначає повідомлення, дозволяючи визначити, до якого запиту відноситься отримана інформація. Коли CoAP-пристрій відправляє повідомлення Confirmable або Non-confirmable, він випадковим чином генерує значення ідентифікатора для цього повідомлення. У відповідях Acknowledgement або Reset ідентифікатори залишаються такими ж, як і в повідомленні, на яке вони відповідають.

За заголовком розташовується значення маркера (Token), розмір якого може бути від 0 до 8 байт, відповідно до значення поля TKL. Маркер генерується випадковим чином пристроєм CoAP і використовується в межах однієї сесії для встановлення зв'язку між запитом і відповіддю, тобто для угруповання повідомлень в ланцюжок «запит-відповідь». Значення маркера рівне нулю, якщо інші маркери не використовуються на пристрої, до якого надсилається запит, або якщо запити робляться послідовно і в невеликій кількості.

Давайте розглянемо кілька прикладів того, як формуються запити та які повідомлення будуть надіслані. На рис. 2.4, наведено приклад взаємодії між HTTP-клієнтом та CoAP-датчиком, а також трансляцію їхніх повідомлень через CoAP-проксі. Припустимо, що користувач запитує дані про освітленість в кухні, які вимірюються датчиком світла.

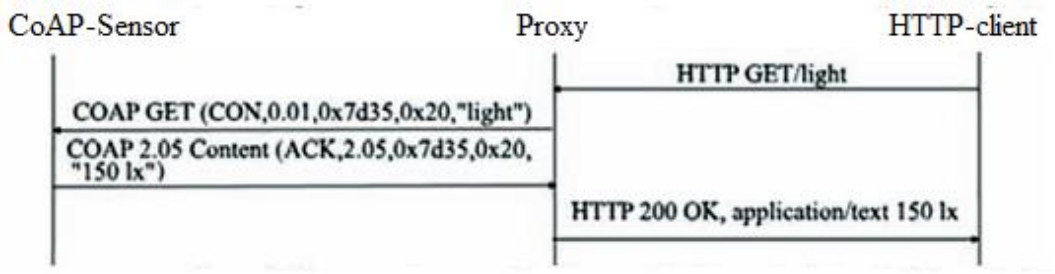

Рисунок 2.4 – Сценарій взаємодії HTTP та CoAP

HTTP-клієнт відправляє запит типу GET для отримання рівня освітленості на кухні. Це повідомлення надходить на проксі, який конвертує HTTP-запит в CoAP-запит. В інформаційному повідомленні Confirmable (CON) передається вимога щодо рівня освітленості, і при цьому призначається ідентифікатор

повідомлення. Цей ідентифікатор використовується в подальшому повідомленні Acknowledgement (ACK), яке містить необхідну інформацію.

Розглянемо деталі формату повідомлення ACK у відповідь на запит (див. рис. 2.5). У полі «Тип» вказується, що це повідомлення ACK (2). Далі йде 1 байт, що вказує на наявність маркера, а за ним слідує код, що свідчить про успішну відповідь (2.05).

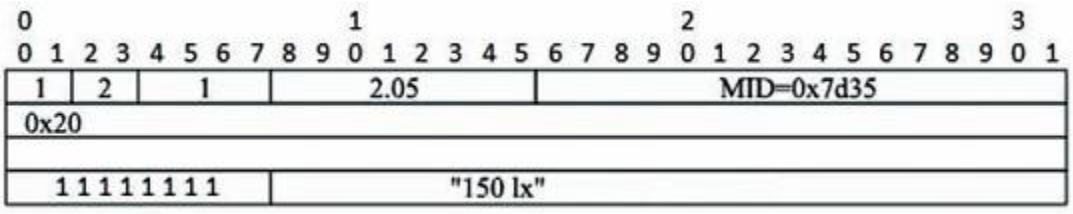

Рисунок 2.5 – Формат повідомлення Acknowledgement

Ідентифікатор повідомлення однаковий як у запиті, так і у відповіді, що дозволяє їх взаємно порівнюватися. Повідомлення завершується передачею в поле корисного навантаження значення рівня освітленості (150 Люкс).

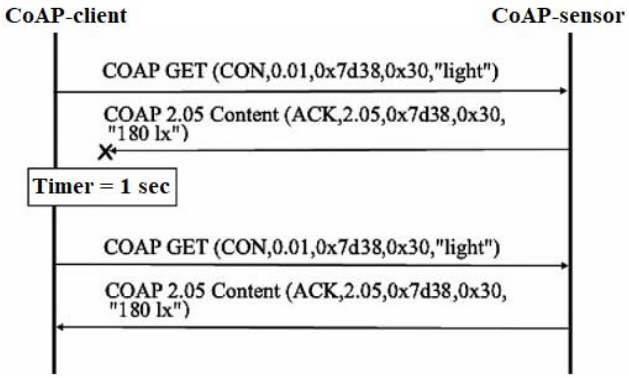

Рисунок 2.6 – Можливий сценарій взаємодії

На рисунку 2.6 представлено демонстраційний приклад взаємодії між CoAP-клієнтом та датчиком. У цьому сценарії розглядається випадок, коли відповідь від CoAP-клієнта була втрачена. У такому випадку буде ініційовано повторну відправку запиту після закінчення таймера очікування відповіді, цей таймер встановлено на 1 секунду відповідно до RFC 6298 [12].

2.1.2 Message Queuing Telemetry Transport (MQTT)

MQTT (Message Queuing Telemetry Transport) – це легкий протокол обміну повідомленнями, спроектований за принципом «публікація-підписка». Він призначений для пристроїв з обмеженими ресурсами та використання в мережах з низькою пропускною здатністю, високою затримкою або ненадійним з'єднанням. MQTT широко використовується в застосунках IoT, де він забезпечує ефективний обмін даними між датчиками, виконавчими механізмами та іншими пристроями.

Під час створення системи Supervisory Control and Data Acquisitio (SCADA) постало завдання забезпечити збір технічної інформації: швидкість потоків води, температура шарикопідшипників, стан керуючих клапанів, стан насосів, рівні в ємностях та інше. При цьому треба врахувати обмежену пропускну здатність каналів зв'язку і обмежену смугу пропускання. Жоден із наявних протоколів не відповідав цим вимогам, тому виникла потреба в новому протоколі з високою якістю обслуговування, двостороннім зв'язком та ефективним використанням смуги пропускання [13].

Протокол MQTT був опублікований у жовтні 2014 року консорціумом Organization for the Advancement of Structured Information Standards (OASIS) та перебуває у відкритому доступі [14].

У червні 2016 року цей стандарт отримав визнання від Міжнародної організації зі стандартизації (ISO). Версія 3.1.1 протоколу MQTT була зареєстрована комітетом із IT ISO (JTC1) за номером ISO/IEC 20922 [15]. Протокол MQTT відзначається такими рисами:

- обмін повідомленнями реалізований з використанням принципу Pub-Sub;
- розмір заголовка повідомлення складає 2 байти, а пейлоад може варіюватися від 1 байта до 260 МБ;
- протокол надає можливість вибрати один з трьох рівнів безпеки, обслуговування.

MQTT працює за принципом видавець/підписник і управляється через центрального брокера [16]. Разючою особливістю такого принципу від клієнт-серверного підходу є те, що клієнти, які надсилають повідомлення (видавеці) та клієнти, які приймають повідомлення (підписники), як правило, розділені. Поділ можна організувати з трьох різних боків:

– простір, Pub та Sub не повинні бути зв'язаними напряму;
– час, Pub і Sub не повинні працювати в один і той самий час;
– синхронізація, операції на обох сторонах не повинні припинятися протягом публікації або отримання інформації.

Pub і Sub не передають один одному інформацію напряму, координування та керування передачею повідомлень від Pub до Sub і від Sub до Pub відповідно здійснює брокер. Розпаралелювання операцій на брокері є другою важливою особливістю принципу взаємодії Pub-Sub, архітектура MQTT зображена на рис. 2.7.

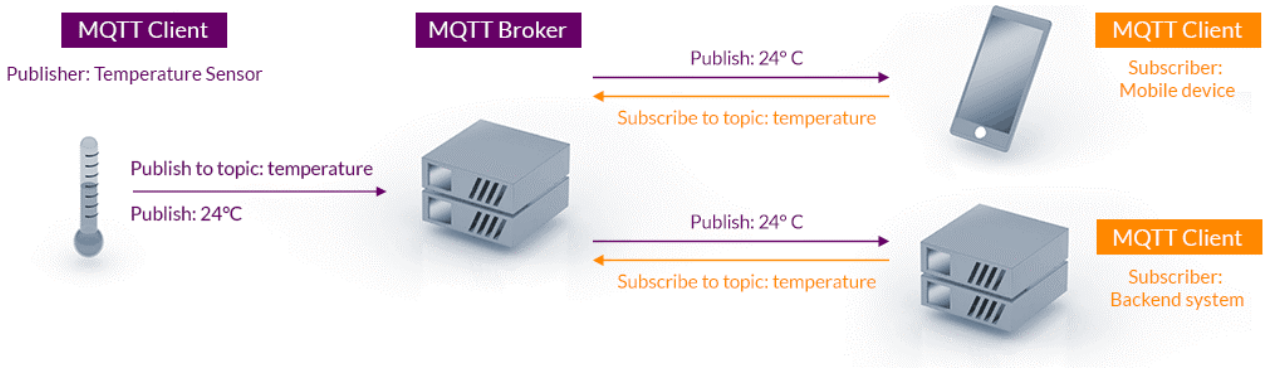

Рисунок 2.7 – Архітектура MQTT протоколу [16]

MQTT-клієнт – це пристрій, обладнаний мікроконтролером із підтримкою стеку TCP/IP. Бібліотеки MQTT для клієнтів доступні для широкого спектру мов програмування, таких як C, C++, C#, Go, iOS, Java, JavaScript, .NET [17].

Брокер є ключовим елементом системи «видавець-підписник», він відповідає за приймання всіх повідомлень, їхню фільтрацію, визначення того, кому цікаві ці повідомлення та в кінцевому підсумку, за пересилання

повідомлень всім клієнтам-підписникам. Серед серверних реалізацій брокера можна виділити декілька варіантів, таких як:

- IBM WebSphere MQ [18] ;
- Mosquitto [19], відкрите програмне забезпечення, яке забезпечує реалізацію протоколу MQTT;
- Eurotech Everywhere Device Cloud [20], хмарний сервіс, який надає рішення для обробки та обміну даними між пристроями IoT;
- eMQTTd [21], легко масштабований і високопродуктивний відкритий сервер MQTT. Здатний обслуговувати велику кількість з'єднань;
- HiveMQ [22], брокер, що забезпечує корпоративну безпеку та максимальну масштабованість для реалізації вимог в галузі MQTT.

Таким чином, дані з датчиків передаються від видавця до брокера, і тільки передплатники, які цікавляться конкретною темою, отримують відповідні повідомлення. Цей механізм дозволяє ефективно організовувати обмін інформацією в системах IoT, де багато пристроїв можуть бути зацікавлені в різних аспектах даних.

Тепер розглянемо загальний формат повідомлень протоколу MQTT, він зображений на рис. 2.8., він складається з:

- заголовок фіксованої довжини;
- заголовок змінної довжини (залежно від типу повідомлення);
- поля корисного навантаження змінної довжини.

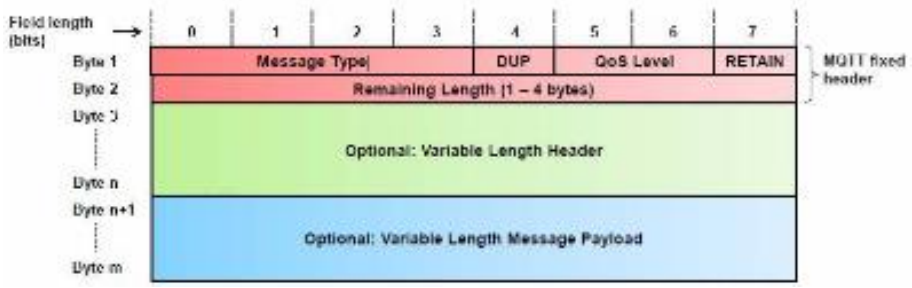

Рисунок 2.8 – Загальний формат повідомлення протоколу MQTT

Докладний опис заголовка фіксованої довжини надається через врахування ключових особливостей протоколу MQTT, оскільки саме через поля цього заголовка реалізуються його важливі аспекти. Перший байт заголовка містить чотири поля, із яких три представлені спеціальними прапорцями: DUP, QoS Level і RETAIN, а четверте вказує на тип повідомлення. Другий байт служить для вказівки залишкової довжини повідомлення, що складається з розміру заголовка змінної довжини і розміру корисного навантаження.

Прапорець QoS (Quality of Service) визначає рівень обслуговування в протоколі MQTT. Як вже зазначалося, основною особливістю цього протоколу є можливість використовувати різні рівні обслуговування, які визначаються значенням цього прапора. Це робить MQTT більш гнучким порівняно з протоколом CoAP (Constrained Application Protocol), де повідомлення може бути підтверджено або оброблено без підтвердження. Розглянемо різні рівні обслуговування QoS.

Коли прапорець QoS дорівнює 0 (At most once), то публікація повідомлення відбувається від видавця до брокера, а потім до підписника. Однак важливо відзначити, що повідомлення немає ніякого зворотнього зв’язку, тобто немає гарантії, що воно обов'язково дійде до підписника. Цей підхід застосовується у випадках, коли втрати даних не мають критичного значення, наприклад, у ситуаціях постійного моніторингу температури. Для наочності, на рис. 2.9 зображений сценарій QoS 0, його ще називають «надіслав та забув».

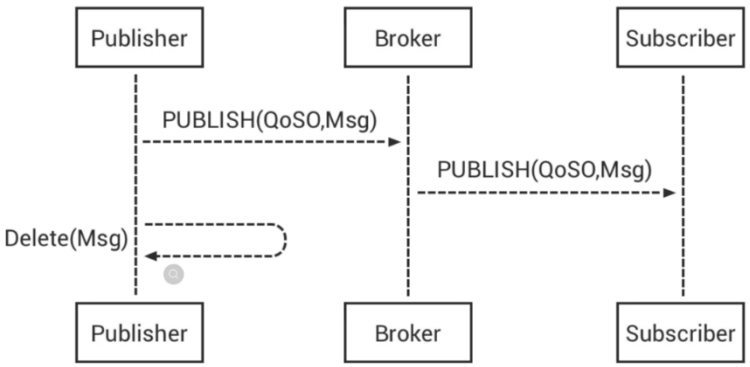

Рисунок 2.9 – Сценарій QoS 0 At most once

Коли QoS дорівнює 1, повідомлення може бути гарантовано опубліковано принаймні один раз. MQTT гарантує QoS 1 за допомогою простого механізму ACK. Публікатор публікує повідомлення і чекає на відповідь PUBACK-пакету одержувача. Якщо відповідь PUBACK не буде отримано протягом зазначеного часу, видавець встановить DUP повідомлення на 1 і виконає повторне надсилання повідомлення. Одержувач повинен відповісти на повідомлення PUBACK при отриманні повідомлення з QoS 1. Одержувач може приймати одне і те ж повідомлення кілька разів. Незалежно від прапора DUP, одержувач розглядатиме отримане повідомлення як нове і надсилатиме пакет PUBACK у відповідь. Сценарій цієї взаємодії представлено на рис. 2.10. Таким чином, передплатник має гарантію отримання цього повідомлення хоча б один раз.

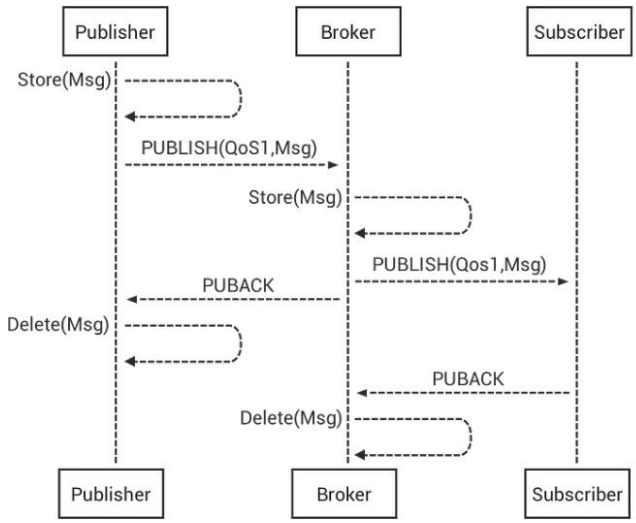

Рисунок 2.10 – Сценарій QoS 1 At least once

Коли прапорець QoS дорівнює 2, видавці та підписники гарантують, що повідомлення публікується лише один раз протягом двох сеансів. Це найвищий рівень обслуговування і втрата та дублювання повідомлень є неприйнятними.

Після публікації повідомлення з QoS 2 видавець зберігає опубліковане повідомлення і чекає на відповідь одержувача з повідомленням PUBREC. Після отримання повідомлення PUBREC видавець може безпечно видалити раніше опубліковане повідомлення, оскільки він вже знає, що одержувач успішно

отримав повідомлення. Видавець зберігає повідомлення PUBREC і відповідає повідомленням PUBREL, чекаючи на відповідь одержувача повідомленням PUBCOMP. Коли відправник отримує повідомлення PUBCOMP, він видаляє попередньо збережений стан, як показано на рис. 2.11.

Якщо під час передачі відбувається втрата пакета, відправник несе відповідальність за повторне надсилання попереднього повідомлення. Це справедливо незалежно від того, чи є відправник видавцем або брокером. Тому одержувач також повинен відповідати на кожне командне повідомлення.

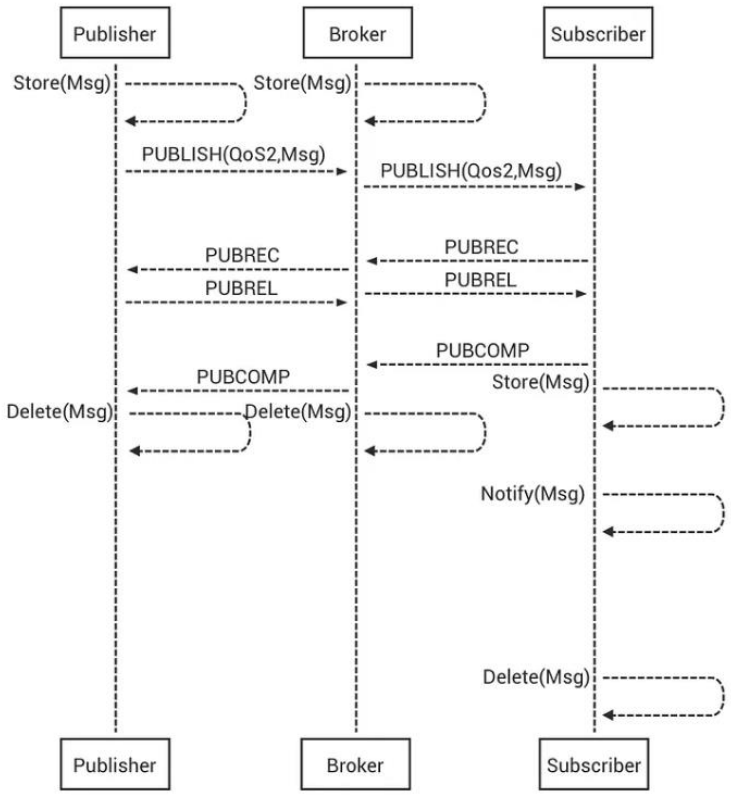

Рисунок 2.11 – Сценарій QoS 2 Publish only once

QoS публікації MQTT-повідомлень не є наскрізним, а є окремо встановленими як для видавця так і для підписника. Рівень QoS, на якому підписники отримують MQTT-повідомлення, в кінцевому підсумку залежить від QoS опублікованого повідомлення і QoS підписки на тему. Можемо звернутися до табл. 2.1, щоб зрозуміти яке повідомлення отримав підписник в різних ситуаціях [23].

Таблиця 2.1 – Відповідність між рівнем QoS в системі, в двойковому вигляді

QoS видавця	QoS підписника	QoS отриманого повідомлення
00	00	00
00	01	00
00	10	00
01	00	00
01	01	01
01	10	01
10	00	00
10	01	01
10	10	11

Підіб’ємо підсумок, мінімальний обсяг службової інформації, наявність класів обслуговування та ієрархічна структура тем є неоспоримими перевагами протоколу MQTT. Це підтверджується великим різноманіттям клієнтського та серверного програмного забезпечення, зокрема, відкритого програмного забезпечення. Архітектура Pub-Sub вносить вимоги до нового мережевого об'єкта – брокера, який забезпечує маршрутизацію користувацької інформації. Таким чином, спостерігається зсув парадигми від маршрутизації на транспортному рівні до маршрутизації на прикладному.

На основі вище викладеного огляду двох протоколів, для виконання роботи обираємо MQTT-протокол.

2.2 Загальна структурна схема SHS

Після вибору основного протоколу обміну повідомленнями в SHS, для подальшого просування в проектуванні, нам необхідно виділити основні, необхідні частини для роботи SHS. Для цього, користуючись отриманими даними, побудуємо дві загальні структурні схеми, одна на фізичному (рис. 2.12), інша на програмному рівні (рис. 2.13). Після аналізу цих двох загальних схем, можна буде сформулювати головні вимоги до тої чи іншої частини проектованої системи.

Умовно, можна виділити 4 групи учасників SHS:

– датчики;
– група взаємодії з навколишнім середовищем (реле, актуатори і т.д);
– локальний оброблювач інформації;
– центральний хаб;

Важливо відмітити, для зв’язку з користувачем, потрібен також маршрутизатор, який має бути з’єднаний з центральним хабом.

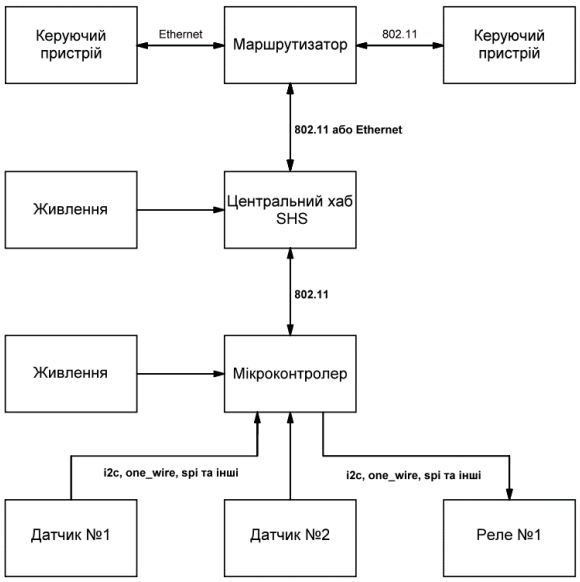

Рисунок 2.12 – Загальна структурна схема на фізичному рівні

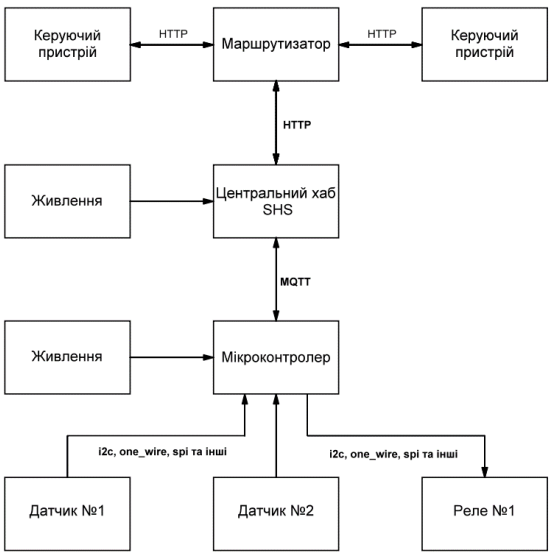

Рисунок 2.13 – Загальна структурна схема на програмному рівні

Локальний оброблювач інформації повинен мати достатню кількість входів та виходів для підключення до нього корисного навантаження. Для підключення цього корисного навантаження, необхідна підтримка популярних протоколів передачі інформації, таких як UART, I^2C, OneWire. Деякі компоненти з групи взаємодії, можуть потребувати широтно-імпульсної модуляції, тому її наявність, хоч в мінімальній кількості буде великою перевагою. Наявність вбудованого Wi-Fi модуля необов'язкове, так як такого роду пристрої можна підключити вище описаними протоколами передачі інформації. Споживаюча потужність повинна бути маленькою, для можливості живлення підсистеми від акумулятора, але потужність такого контролера повинна бути великою. Завдяки цим характеристикам, спроектована підсистема зможе нормально функціонувати в системі.

Центральний хаб, найвідповідальніша частина SHS, бо саме вона буде виступати в ролі брокера для MQTT протоколу, на ньому буде розгорнута система організації і автоматизації та на ньому ж буде розгорнутий користувацький інтерфейс. Ці умови накладають певні обмеження на вибір хабу, бо для цих завдань не підійдуть прості мікроконтролери, потрібно потужний процесор зі швидкою оперативною пам'яттю, об'ємом більше 512 МБ. Хаб повинен мати вбудований ethernet контролер та порт RJ45 для можливості підключення до маршрутизатора витою парою. Так як цей вузол найважливіший в SHS, необхідно забезпечити найбільшу пропускну здатність в цьому місці. Наявність Wi-Fi модуля необхідний для зв'язку з локальними оброблювачами інформації.

Припускається, що у багатьох користувачів вже є власний маршрутизатор, тому його включення в систему не передбачено. Однак, незважаючи на це, наявність маршрутизатора є обов'язковою для функціонування системи. Всі сучасні маршрутизатори будуть задовольняти нашу систему, але чим швидше – тим краще. Виключення маршрутизатора зі складу системи суттєво знижує кінцеву вартість продукту для кінцевого користувача.

2.3 Система організації, автоматизації Home Assistant SHS

Home Assistant – це вільне та відкрите програмне забезпечення для автоматизації домашнього середовища, розроблене для централізованого керування пристроями розумного будинку, з основним акцентом на локальному управлінні та конфіденційності. Ним можна керувати через веб-інтерфейс користувача, використовуючи додатки для Android та iOS або голосовими командами через підтримувані віртуальні асистенти, такі як Google Assistant або Amazon Alexa.

Після встановлення на пристрій програмного забезпечення, Home Assistant діє як центральна система керування домашньою автоматизацією, що зазвичай називають хабом для розумного будинку. Він спеціально призначений для керування пристроями, програмним забезпеченням, додатками та послугами технології зв'язку IoT. Він підтримує модульні інтеграційні компоненти, включаючи вбудовані компоненти для бездротових протоколів зв'язку, таких як Bluetooth, Zigbee і Z-Wave.

Home Assistant також підтримує керування відкритими та власними екосистемами, якщо вони надають публічний доступ через відкритий API або MQTT для інтеграцій сторонніх розробників через локальну мережу або інтернет. Це дуже важливо, бо для інклюзивних людей можна розробити будь-який унікальний пристрій, що буде корисним саме їй і за допомогою MQTT протоколу підключити його до Home Assistant.

Інформацію з усіх пристроїв та їх атрибути, які програмне забезпечення Home Assistant розпізнає, можна використовувати для керування пристроями в межах сценаріїв для запуску автоматизації, за допомогою графіків. Наприклад, для керування освітленням, кондиціонуванням повітря, системами розваг та побутовими приладами.

Home Assistant активно підтримується та може бути встановлений на різних платформах. До них належать одно-платні комп'ютери Hardkernel

ODROID, Raspberry Pi, Asus Tinkerboard, Intel NUC та інші, операційні системи Windows, macOS, Linux, а також NAS-системи. Підтримка Windows здійснюється за допомогою віртуальної машини Windows або встановлення Windows Subsystem for Linux.

На офіційно підтримуваних апаратних платформах, таких як одноплатні комп'ютери ODROID N2+ та Raspberry Pi 3/4, встановлення вимагає прошивання відповідного образу системи на карту microSD, eMMC або на інше локальне сховище, з якого система може завантажитися. Home Assistant можна використовувати як шлюз або міст для пристроїв, що використовують різні технології IoT, такі як Zigbee або Z-Wave, необхідне обладнання можна встановити на GPIO (Serial/I2C/SMBus), UART або за допомогою USB-портів. Крім того, він може прямо або опосередковано підключатися до локальних IoT-пристроїв, концентраторів/шлюзів/мостів або хмарних сервісів від багатьох різних постачальників, включаючи інші відкриті та закриті екосистеми розумного будинку.

Рисунок 2.14 – ODROID N2+ «Home Assistant Blue»

У грудні 2020 року було представлено модернізований комп'ютерний пристрій ODROID N2+ з комплектним програмним забезпеченням під назвою «Home Assistant Blue» як офіційно підтримувану загальну апаратну еталонну платформу, вона зображена на рис. 2.14. Він постачається з попередньо встановленою ОС Home Assistant на локальному накопичувачі eMMC, адаптером

живлення та спеціальним тематичним чохлом Home Assistant. Характеристики такого пристрою відмінно підходять для автоматизації великих будинків, бо мають в своєму розпорядженні 6-ядерний процесор Amlogic S922X з частотою роботи до 2.2 ГГц ARM-архітектури, 4 Гб оперативної пам'яті формату DDR4 та 128 Гб eMMC флеш накопичувач але без Wi-Fi та Bluetooth. Засновники Home Assistant дали зрозуміти, що випуск офіційного апаратного забезпечення не завадить підтримувати їх інші апаратні платформи, такі як серія Raspberry Pi.

Принципи автоматизації в Home Assistant базуються на концепції «автоматизаторів», які дозволяють визначати умови та викликати дії відповідно до цих умов. Основні складові цих автоматизаторів включають в себе тригери, умови, дії та групування.

Тригер визначає події або умови, які спричиняють запуск автоматизації. Тригер може бути спровокований різними подіями, такими як зміна стану пристрою, часовий інтервал, сонячне освітлення, натискання кнопки.

Умови визначають додаткові обмеження для виклику автоматизації. Наприклад, користувач може вказати, що автоматизація повинна виконуватися тільки при певних умовах, таких як час доби, наявність людей вдома, або стан інших пристроїв.

Дії визначають завдання, які виконуються при виклику автоматизації. Це може бути зміна стану пристроїв, відправлення повідомлення, виклик служби, запуск сценарію, або навіть виконання скрипту.

Home Assistant дозволяє групувати автоматизатори та використовувати шаблони для створення більш важких та гнучких правил автоматизації. Шаблони дозволяють вам створювати умови та дії, які залежать від змінних параметрів.

Підсистема автоматизації Home Assistant надає зручний та потужний інтерфейс для створення складних правил автоматизації. Можна використовувати графічний інтерфейс веб-керування або конфігурувати автоматизації безпосередньо у файли конфігурації YAML, що дозволяє отримати повний контроль над системою.

2.4 Огляд мов програмування

2.4.1 Python

Python - це виразна високорівнева мова програмування з проектуванням, що надає перевагу читабельності коду завдяки значним відступам. Мова включає динамічну типізацію та систему збору сміття, сприяє різним парадигмам програмування, включаючи об'єктно-орієнтовану, структуровану (зокрема, процедурну) та функціональну програмування. Розширений функціонал Python пояснюється його високоякісною стандартною бібліотекою, завдяки чому він часто отримує прізвище «мова з батарейками в комплекті».

Ця мова цілком підтримує об'єктно-орієнтоване та структуроване програмування, а також багато з її функцій спрямовані на функціональне та аспектно-орієнтоване програмування, включаючи метапрограмування та метаоб'єкти. Крім того, вона може підтримувати різні інші парадигми за допомогою розширень, таких як проектування за контрактом та логічне програмування.

Мова використовує динамічну типізацію та комбінує підрахунок посилань і збирач сміття для ефективного управління пам'яттю. Крім того, вона використовує динамічне розпізнавання імен (пізнє зв'язування), яке зв'язує імена методів та змінних під час виконання програми.

Дизайн мови також передбачає певну підтримку функціонального програмування у традиціях Ліспа. Стандартна бібліотека включає функції для фільтрації, відображення та зменшення, роботи зі списками, словниками, наборами та генераторами виразів. Крім того, вона включає два модулі, itertools та functools, які використовують функціональні інструменти з Haskell та Standard ML.

Замість вбудовання всієї своєї функціональності в ядро, Python був розроблений так, щоб його можна було легко розширювати за допомогою модулів. Ця компактна модульність зробила його популярним вибором для

додавання програмованих інтерфейсів до існуючих додатків. Гвідо ван Россум задумав невелику, основну мову з великою стандартною бібліотекою та легко розширюваним інтерпретатором, це виникло після його розчарування підходом у мові програмування ABC.

Синтаксис і семантика мови Python були розроблені так, щоб їх було легко читати, з візуально приємним форматуванням, часто використовуючи англійські ключові слова, а не розділові знаки. Приклад коду з аналогічною реалізацією наведено на рис. 2.15.

```
def quadrato(*numeri):
        c = [str(x**2) for x in numeri]
        print ' ; '.join(c)

def divisore(num, den):
        try:
                if den == 1:
                        return num
                else:
                        return float(num)/den
        except ZeroDivisionError:
                print 'Errore. Divisione per zero!'
```

Рисунок 2.15 – Приклад коду на Python

У Python для визначення блоків замість фігурних дужок або ключових слів використовується відступ пробілів. Збільшення відступу відбувається після певних операторів, а зменшення відступу означає кінець поточного блоку. Це призводить до того, що візуальна структура програми точно відображає її семантичну структуру. Ця концепція також відома як правило off-side. Хоча деякі інші мови можуть використовувати відступи, у більшості випадків вони не несуть смислового значення.

Оператор "=" в Python встановлює зв'язок імені з динамічно виділеним об'єктом. Змінні можуть посилатися на будь-який об'єкт у будь-який момент

часу. У Python ім'я змінної є узагальненим посиланням без фіксованого типу даних, але завжди посилається на об'єкт з певним типом. Це відомо як динамічна типізація, що відрізняється від мов із статичною типізацією, де змінні можуть містити значення лише певного типу.

Python розрізняє списки та кортежі. Списки записуються у вигляді [1, 2, 3] і можуть бути змінені, але не можуть бути використані як ключі у словниках (оскільки ключі словників у Python повинні бути незмінними). Кортежі, представлені як (1, 2, 3), є незмінними та можуть використовуватися як ключі в словниках, якщо всі елементи кортежу є незмінними.

Розпакування послідовності – це функція у Python, за допомогою якої обчислюються декілька виразів, щоб дозволити присвоєння змінній або властивості, доступній для запису. Це робиться подібно до формування літералів кортежу, вони розміщуються разом ліворуч від знаку рівності в операторі присвоювання. Це спрощує мову і забезпечує узгодженість процесів і процедур. Крім того, це гарантує, що мова є конкретною, прямою і вільною від зайвих наповнювачів. Оператор присвоювання вимагає наявності ітерабельного об'єкта праворуч від знаку рівності, який генерує таку ж кількість значень, що й надані вирази, які можна записати. При ітерації через них, він присвоює кожне згенероване значення відповідному виразу зліва.

У Python є оператор «форматування рядків» %, який працює подібно до форматування рядків printf у C. Наприклад, «pipe=%s water=%d» % («blap», 2) обчислюється як «pipe=blap water=2». У версіях Python 2.6 і 3 можна використовувати метод format() класу str, щоб доповнити існуючу функціональність. Наприклад, «pipe={0} water={1}».format(«blap», 2).

У Python є різні рядкові літерали, які розділяються одинарними або подвійними лапками. На відміну від оболонок Unix, Perl та мовах, що зазнали впливу Perl, одинарні та подвійні лапки працюють однаково. Вони обидві використовують зворотну косу риску (\) як символ екранування. Форматовані

рядкові літерали, також відомі як інтерполяція рядків, стали доступними у Python версії 3.6.

Літерали у потрійних лапках, які починаються і закінчуються трьома одинарними або подвійними лапками, можуть охоплювати декілька рядків і функціонувати подібно до документуючих рядків (докстрінгів) у інших мовах, таких як оболонки, Perl і Ruby.

Типізація. Типові обмеження не перевіряються під час компіляції, тому операції з об'єктами можуть завершитися невдачею, це означає, що об'єкт має невідповідний тип. Хоча Python має динамічну типізацію, вона є строго типізованою і забороняє невизначені операції (наприклад, додавання числа до рядка) замість того, щоб приймати значення та компілювати код.

Програмісти можуть визначати власні типи за допомогою класів у Python, які широко використовуються в об'єктно-орієнтованому програмуванні. Виклик класу створює нові екземпляри класів: наприклад, виклик PipeClass() або WaterClass(). Крім того, класи самі є екземплярами типу метакласу, що дозволяє здійснювати метапрограмування та рефлексію.

Мета полягає в тому, щоб забезпечити підтримку майже всіх конструкцій мови Python у муру. У ньому вже є підтримка більшості функцій та ідіом Python, і наразі він пропонує часткову підтримку нещодавно випущеного Python 3.12. Зауважемо, що включення підказок у вихідний код Python завжди є безпечним, оскільки вони є частиною специфікації мови і будуть підтримуватися всіма майбутніми версіями Python. Однак, підтримка деяких інструментів може відставати від нового синтаксису, наприклад, додавання у Python нових ключових слів, таких як «match». В останніх версіях муру додано таку підтримку, але вона все ще може не спрацьовувати у коді Python, що використовує найновіші можливості. Тим не менш, цей код буде цілком працездатним у досить пізніх версіях CPython.

Завдяки розширеній математичній бібліотеці Python та додатковій бібліотеці NumPy від сторонніх розробників, яка розширює і без того широкі

можливості мови, вона стала поширеною мовою наукових сценаріїв, що використовується для таких завдань, як обробка числових даних та маніпуляції з ними.

Більшість реалізацій Python, включаючи CPython, мають цикл REPL. Це дає можливість діяти їм як інтерпретатор командного рядка, в якому користувачі вводять оператори по одному і миттєво отримують результати. Python також надає початківцям інтегроване середовище розробки (IDE) під назвою IDLE.

Додаткові оболонки, такі як IDLE та IPython, покращують користувацький досвід, забезпечуючи покращене автозавершення, збереження стану сеансу та підсвічування синтаксису.

Окрім традиційних інтегрованих середовищ розробки для настільних комп'ютерів, існують IDE на основі веб-браузера, такі як SageMath, спеціально розроблені для розробки програм, пов'язаних з наукою та математикою. Існує також PythonAnywhere, браузерна IDE та хостингове середовище і Canopy IDE, комерційна IDE, яка надає пріоритет науковим обчисленням.

Еталонна реалізація. CPython – це стандартна реалізація Python, написана мовою C і відповідає стандарту C89. Python 3.11 використовує C11 з деякими можливостями C99. CPython має власні розширення C, але сторонні розширення не обмежуються застарілими версіями C. Вони можуть бути реалізовані за допомогою C11 або C++. Програма компілює код Python у проміжний байт-код, який потім виконується віртуальною машиною. CPython постачається з повною стандартною бібліотекою, написаною як на C, так і на рідній мові Python. Він широко доступний на багатьох платформах, включаючи Windows (починаючи з Python 3).

Вдосконалення мови узгоджується з розвитком еталонної реалізації CPython. Список розсилки python-dev слугує основним форумом для розробки мови. Спочатку конкретні питання обговорювалися на баг-трекері Roundup, який розміщувався у фонді. У 2022 році всі обговорення та проблеми були перенесені на GitHub. Розробка Python спочатку виконувалася на саморозміщеному

репозиторії вихідного коду під управлінням Mercurial до січня 2017 року, коли його розробка була перенесена на GitHub.

Дослідження показало, що скриптові мови, такі як Python, є більш ефективними, ніж традиційні мови, такі як C та Java, для вирішення проблем програмування, пов'язаних з пошуком у словнику та маніпулюванням рядками. Крім того, дослідження показало, що споживання пам'яті в Python часто перевершує Java і лише трохи поступається C або C++. [183]

Серед відомих організацій, які використовують Python – Google, Wikipedia, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify, а також деякі менші гравці, такі як ILM та ITA. Python є основною мовою програмування, яка використовується для створення соціальної мережі новин Reddit.

2.4.2 YAML

YAML – це проста для розуміння мова серіалізації даних. Її часто використовують для конфігураційних файлів і в додатках, де зберігаються або обмінюються даними. YAML і розширювана мова розмітки (XML) мають багато спільних комунікаційних застосувань. Однак YAML навмисно використовує мінімальний синтаксис, який відрізняється від стандартної узагальненої мови розмітки (SGML). Він вказує на вкладеність за допомогою відступів у стилі Python і більш стислого формату, який використовує [...] для списків і {...} для мап. Він не дозволяє використовувати символи табуляції для відступів, тому лише деякі JSON-файли відповідають критеріям дійсного YAML 1.2.

Є можливість створювати користувацькі типи даних, однак YAML за своєю суттю кодує скаляри (наприклад, рядки, цілі числа і числа з плаваючою комою), списки і асоціативні масиви, також відомі як мапи, словники або хеші. Ці типи даних походять з мови програмування Perl, хоча більшість мов програмування високого рівня мають схожі принципи.

Синтаксис зосереджений на двокрапках, що застосовуються для вираження пар ключ-значення, запозичених із заголовків електронної пошти, як

зазначено в RFC 822, тоді як роздільник документів --- запозичено з MIME (RFC 2046). Повторно використовуються екрануючі послідовності мови C, а обгортання пробілів для багаторядкових рядків запозичено з HTML. Вкладені списки та хеші можуть утворювати деревоподібну структуру як всередині списків, так і всередині хешів. Псевдоніми YAML, такі як ті, що використовуються в XML для SOAP, дозволяють представляти довільні графіки.

YAML підтримується багатьма мовами програмування, а редактори вихідного коду, такі як Vim, Emacs та різні інтегровані середовища розробки, пропонують такі функції, як підсвічування синтаксичних помилок та згортання вкладених структур, щоб полегшити редагування YAML. Крім того, з 2006 року розширення .yaml є рекомендованим розширенням для файлів YAML.

Список також можна вказати, взявши текст у квадратні дужки ([...]), де кожен елемент відокремлюється комою.

Елемент масиву – це пара ключ-значення, де ключ і значення відокремлюються двокрапкою і пробілом. Кожен елемент перераховується в окремому рядку. Для представлення рядків у стилі URL, без взяття їх у лапки, YAML вимагає, щоб після двокрапки ставився пробіл.

Перед ключем можна додати знак питання у вигляді «?ключ: значення», щоб дозволити ключу включати спеціальні символи, такі як тире або квадратні дужки, не беручи їх у лапки.

Асоціативний масив можна визначити, взявши текст у фігурні дужки ({...}), де ключі та значення відокремлюються двокрапкою, а записи - комами (пробіли не є обов'язковими для підтримки сумісності з JSON).

Рядки, особливий скалярний тип в YAML, зазвичай не потребує лапок, але може бути взятий в подвійні лапки («») або одинарні (').

Спеціальні символи у подвійних лапках можуть бути представлені за допомогою екранованих послідовностей у стилі C, що починаються зі зворотної косої риски (\). Згідно з документацією, єдиною вісімковою екранованою послідовністю, яка підтримується, є \0.

Єдиною екрануючою послідовністю, яка підтримується всередині одинарних лапок, є подвоєна одинарна лапка ('), яка позначає саму одинарну лапку, як наприклад, «don't't».

Блокові скаляри розмежовуються відступами, з додатковими модифікаторами для збереження (|) або згортання (>) нових рядків.

YAML відрізняється від інших мов серіалізації даних своїми структурами та типізацією даних. Ці структури дозволяють легко зберігати кілька документів в одному файлі, а також використовувати посилання для повторюваних вузлів і довільні вузли як ключі.

Якорі вузлів (за допомогою &) і посилання (за допомогою *) передбачені для ясності, компактності і для запобігання помилок при введенні даних. Посилання на якір працює для всіх типів даних.

На рис. 2.16, наведений приклад черги в секвенсорі інструментів з двома кроками, на які є посилання без повного опису.

```yaml
--- # Sequencer protocols for Laser eye surgery
- step:  &id001                  # defines anchor label &id001
    instrument:      Lasik 2000
    pulseEnergy:     5.4
    pulseDuration:   12
    repetition:      1000
    spotSize:        1mm

- step: &id002
    instrument:      Lasik 2000
    pulseEnergy:     5.0
    pulseDuration:   10
    repetition:      500
    spotSize:        2mm
- Instrument1: *id001   # refers to the first step (with anchor &id001)
- Instrument2: *id002   # refers to the second step
```

Рисунок 2.16 – Приклад коду

Явна типізація даних зазвичай не використовується в більшості документів YAML, оскільки YAML автоматично визначає прості типи. Типи даних можна розділити на три категорії: базові, визначені та визначені користувачем. Основні

типи – це ті, які, як існують у всіх синтаксичних аналізаторах (наприклад, типи з плаваючою комою, цілі числа, рядки, списки, карти тощо).

Більш складні типи даних, такі як двійкові дані, визначені в специфікації YAML, але підтримуються не всіма реалізаціями. YAML дозволяє локально розширювати визначення типів даних для підтримки користувацьких класів, структур або примітивів, таких як числа з плаваючою комою четвертої точності.

Хоча YAML може визначати тип даних сутності, існують ситуації, коли необхідне явне приведення типів даних. Найпоширеніший випадок - це коли рядок з одного слова нагадує число, логічне значення або тег, що вимагає розмежування за допомогою лапок або явного тегу типу даних (рис. 2.17)

```yaml
---
a: 123                  # an integer
b: "123"                # a string, disambiguated by quotes
c: 123.0                # a float
d: !!float 123          # also a float via explicit data type prefixed by (!!)
e: !!str 123            # a string, disambiguated by explicit type
f: !!str Yes            # a string via explicit type
g: Yes                  # a boolean True (yaml1.1), string "Yes" (yaml1.2)
h: Yes we have No bananas  # a string, "Yes" and "No" disambiguated by context.
```

Рисунок 2.17 – Приклад наведення різних типів даних

Features. Неієрархічні моделі даних. Тоді як JSON може представляти дані лише в ієрархічній моделі, де кожен дочірній вузол має єдиного батька, YAML пропонує просту реляційну схему. Це дозволяє посилатися на ідентичні дані з двох або більше точок дерева, а не вводити їх у цих точках надлишково. Це можна порівняти з об'єктом IDREF, створеним у XML. Синтаксичний аналізатор YAML розширює ці посилання у повністю заповнені структури даних після зчитування, заперечуючи необхідність використання реляційної моделі кодування у програмі, що використовує синтаксичний аналізатор. Це покращення може підвищити чіткість і зменшити кількість помилок при введенні даних у конфігураційні файли або протоколи обробки з багатьма параметрами,

які залишаються незмінними в послідовній серії записів, тоді як змінюються лише деякі. Наприклад, в інвойсі записи «ship-to» і «bill-to» майже завжди містять ідентичні дані.

Features. Розмежування з відступами. YAML покладається на контурний відступ для структури, що робить його дуже стійким до зіткнення розділювачів. Крім того, нечутливість YAML до лапок і дужок у скалярних значеннях дозволяє вбудовувати XML, JSON і YAML-документи у YAML-документ за допомогою блочного літерала (з використанням | або >) і належного відступу (2.18).

```
---
example: >
        HTML goes into YAML without modification
message: |

        <blockquote style="font: italic 1em serif">
        <p>"Three is always greater than two,
           even for large values of two"</p>
        <p>--Author Unknown</p>
        </blockquote>
date: 2007-06-01
```

Рисунок 2.18 – Розмежування з відступами

YAML базується на рядках, що дозволяє легко конвертувати неструктуровані дані з існуючих програм у формат YAML, зберігаючи при цьому зовнішній вигляд оригінального документа. Відсутність у YAML закриваючих тегів, дужок або лапок гарантує, що генерувати добре сформований YAML безпосередньо з нескладних програм буде дуже просто. Аналогічно, роздільники пробілів допомагають швидко фільтрувати YAML-файли за допомогою лінійно-орієнтованих команд, таких як grep, AWK, Perl, Ruby і Python.

Слід зазначити, що на відміну від мов розмітки, послідовні рядки YAML, як правило, самі по собі є добре сформованими документами YAML. Це спрощує написання синтаксичних аналізаторів, яким не потрібно обробляти весь документ, перш ніж витягувати певні записи з середини.

2.5 Вибір елементної бази

2.5.1 Датчик вологості та температури SHT30

SHT30A-DIS – це цифровий датчик вологості та температури, розроблений компанією Sensirion, яка заснована в Швейцарії. Він ґрунтується на передовому сенсорному чіпі CMOSens®, який є основою для нової платформи вимірювання вологості та температури. SHT30A-DIS, це покращена версія попереднього датчику SHT20, в порівнянні з попередником, розробники покращили надійність і точність показників вологості та температури [24].

Датчик надійно працює в межах рекомендованого, нормального діапазону, який відображений на рис. 2.19. Продовжений вплив умов, що виходять за межі нормального діапазону, особливо при вологості >80%RH, може тимчасово зсунути сигнал відносної вологості (+3%RH після 60 годин впливу). Після повернення в нормальний діапазон він повільно відновлює свій сигнал до початкового стану калібрування. Тривале перебування в екстремальних умовах може спричинити швидше старіння датчика та зменшити його точність[4].

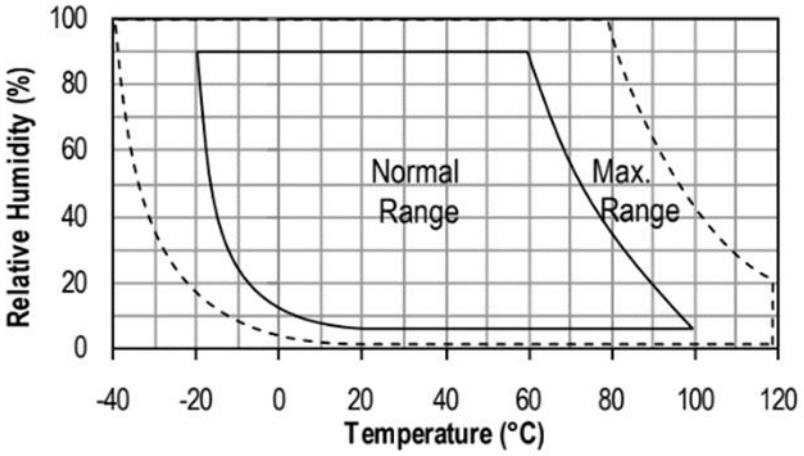

Рисунок 2.19 – Умови роботи датчика SHT30 [24]

Розглянемо точність вимірювання відносної вологості при різних температурах. Максимальний допуск для точності вимірювання відносної вологості, при 25°C представлено на рис. 2.20, А. Для інших температур максимальний допуск був оцінений в межах, зазначених на рис. 2.20, В.

Важливо відзначити, що наведені значення є максимальними допусками (без урахування гістерезису) порівняно з високоточним еталонним зразком, таким як метод визначення точки роси. Типові відхилення становлять ±2% відносної вологості, де максимальний допуск складає ±3% відносної вологості і приблизно половина максимального допуску для інших значень.

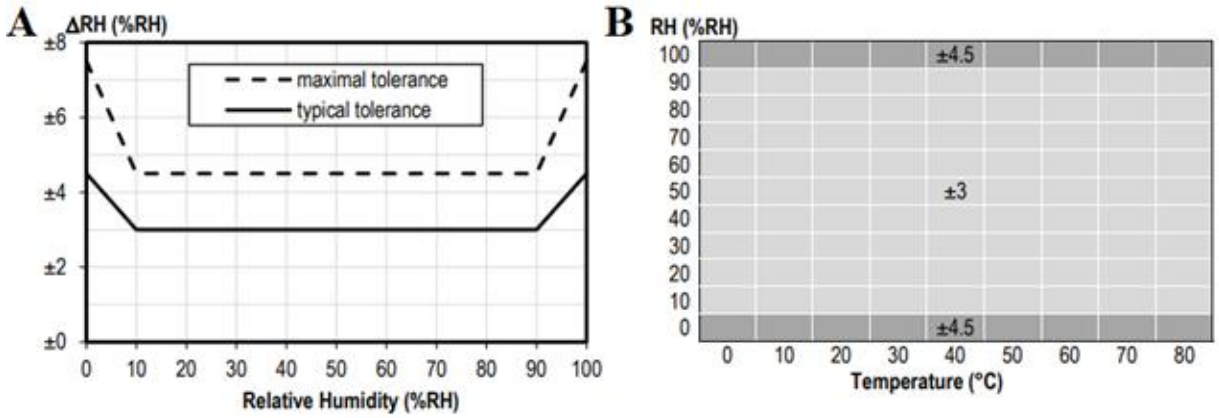

Рисунок 2.20 – Допуск відносної вологості при 25°C (A), Залежність допуску RH від T (B), для SHT30A [24]

SHT30 також відзначається низьким споживанням енергії (3.3В), що робить його ідеальним вибором для використання в пристроях, де реалізується живлення відм мережі та акумулятора. Це може бути важливим фактором для систем розумного будинку, де енергоефективність та тривалість роботи пристроїв грають важливу роль.

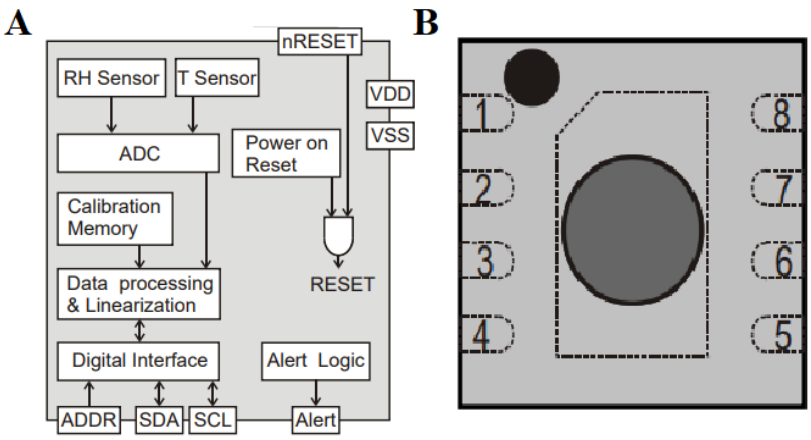

Рисунок 2.21 – Функціональна схема (A), вигляд датчику SHT30A (B) [24]

На рис. 2.21, А представлена функціональна блок-схема SHT30A. Сигнали вологості та температури з датчиків калібруються на заводі, лінеаризуються і компенсуються в залежності від температури та напруги живлення. Зовнішній вигляд датчика представлений на рис. 2.21, В, а всі піни датчика та їх призначення перераховані в табл. 2.2 [3].

Таблиця 2.2 – Піни SHT30A [3]

№ Піну	Назва	Функція
1	SDA	Передача інформації від/до МК; вхід/вихід.
2	ADDR	PIN-код адреси; введення;
3	ALERT	Вказує на стан тривоги; вихід; потрібно залишити плаваючим, якщо він не використовується
4	SCL	Тактуючий сигнал для прийомо-передачі; Вхід/Вихід
5	VDD	Напруга живлення; Вхід
6	nRESET	Контакт скидання, активний низький рівень; Вхід; якщо не використовується, рекомендується залишити плаваючим;
7	R	Немає електричної функції; для підключення до VSS
8	VSS	Земля

Джерело живлення повинно бути підтягнуто до землі за допомогою керамічного конденсатора ємністю близько 100 нФ, який слід розташувати якомога ближче до датчика.

SHT30 підтримує кілька різних протоколів зв'язку, що робить його сумісним з різними системами розумного будинку. Один з найпоширеніших протоколів, який підтримується SHT30, - це I2C, який забезпечує простий і ефективний спосіб зчитування даних з датчика. Крім того, SHT30 також підтримує протокол зв'язку SMBus, що робить його сумісним з широким спектром пристроїв та контролерів для систем розумного будинку.

2.5.2 Датчик пропану MQ4

Датчик виявлення пропану MQ-4 є напівпровідниковим пристроєм, принцип його дії базується на зміні опору тонкого шару діоксиду олова (SnO2) при контакті з молекулами природного. Чутливий елемент датчика має в своїй будові керамічну трубку, яку покрито AlO3 та на яку нанесено чутливий шар діоксиду олова [25].

Для взаємодії діоксиду олова з природним газом, його потрібно нагріти до робочої температури, для цього в середині керамічної трубки розташована ніхромова спіраль.

Вимірювання відбувається за рахунок зміни опору всередині датчика, компаратор на схемі порівнює його з пороговим значенням, яке задає потенціометр, в разі перевищення – датчик передасть на мікроконтролер логічну одиницю.

На рис. 2.22 показані типові характеристики чутливості MQ-4 для поширених газів.

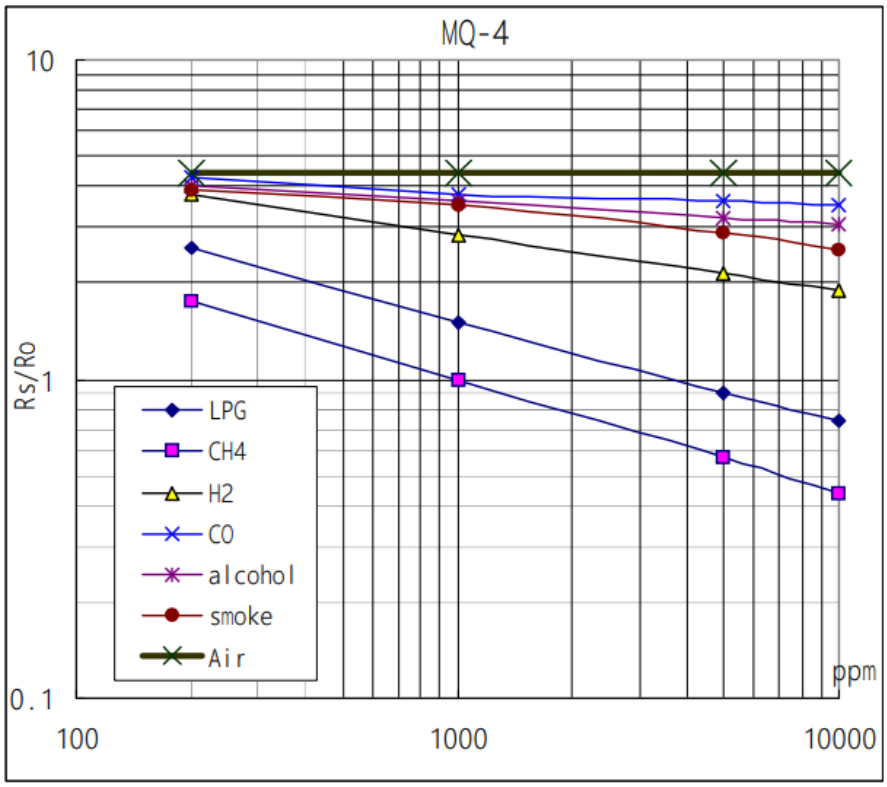

Рисунок 2.22 – Характеристика чутливості датчика MQ-4 [25]

2.5.3 Датчик відкривання вікон KY-025

Контроль стану вікон відбувається за допомогою модуля KY-025, в якому встановлений геркон. Принципова схема модуля зображена на рис. 2.23.

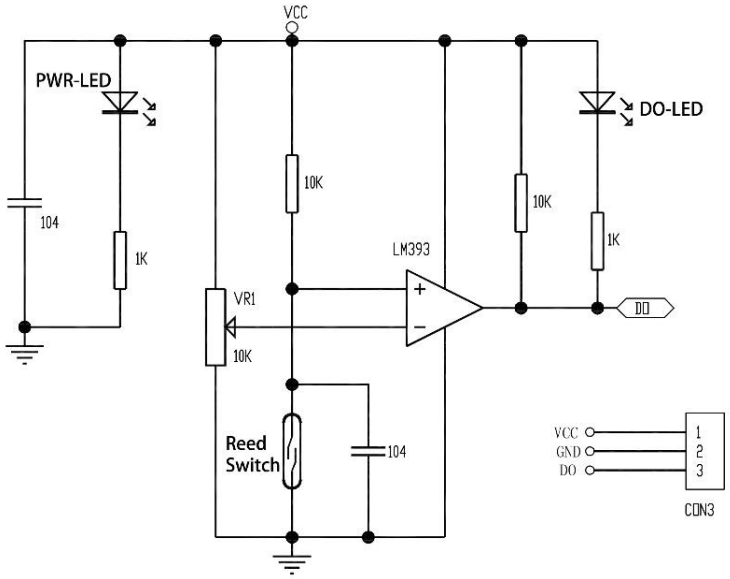

Рисунок 2.23 – Принципова схема KY-025

Модуль KY-025, згідно принципової схеми складається з геркона, потенціометра, компаратора LM393, світлодіода та обв'язки. Коли магніт знаходиться близько до модуля, контакти геркона замикаються і модуль виводить низький рівень сигналу, коли ж магніту немає, геркон розмикається і вивід модуля переходить на високий рівень. Модуля є енергоефективним, його робоча напруга лежить в діапазоні від 3.3 до 5В. Важливо зазначити, що відстань між герконом та магнітом повинна бути в межах 1,5 см, оскільки за межами цього діапазону втрачається чутливість або не спрацьовує тригер. Додатково, можна використовувати потенціометр для регулювання чутливості модуля.

2.5.4 Датчик протікання

Датчик протікання дає змогу визначити появу крапель вологи і вчасно відреагувати на це закривши клапан постачання води. Структурна схема датчика зображена на рис. 2.24.

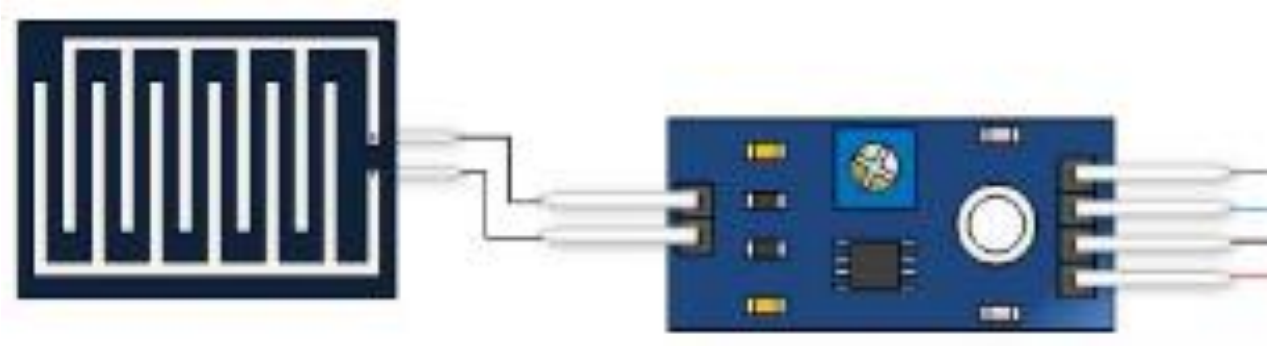

Рисунок 2.24 – Структурна схема датчика протікання.

Згідно структурної схеми, датчик складається з двох частин, плата з компаратором, аналогічна як в датчику відкривання вікна, тому характеристика з енергоефективністю зберігається, та площадки з струмопровідними доріжками, як розташовані на відстані близько одного сантиметра один від одного.

Принцип роботи заключається у вимірі опору на струмопровідних доріжках. При потраплянні вологи, через неї буде протікати малий струм, який буде фіксуватися входом компаратора. На інший вхід компаратора потрапляє пороговий сигнал, через потенціометр формується висота цього порогу. Це допомагає запобігти хибних тривог, наприклад конденсація пару або потрапляння невеликої кількості крапель води.

2.5.5 Реле світла

Твердотільні реле – це сучасний аналог електромеханічного реле [3], призначений для керування електричним навантаженням, без частин, які рухаються та зношуються. Його передвісник - електромеханічне реле, яке використовує котушки, пружини та механічні п’ятаки контактів для перемикання джерела живлення. Твердотільні реле немає жодної рухомої частинки, на рис. 2.25 зображена його структурна схема. Натомість вони використовують електричні та оптичні властивості твердотільних напівпровідників.

Таке реле забезпечує ідеальну електричну ізоляцію між його вхідними та вихідними контактами. Вихід твердотільного реле функціонує як звичайний вимикач, оскільки він має дуже високий, практично нескінченний імпеданс у режимі розімкнення та дуже низький опір у режимі проведення. Твердотільні реле можуть бути розроблені для перемикання як змінного, так і постійного струму, використовуючи SCR, TRIAC або комутаційний транзисторний вихід, замість традиційних механічних контактів, які розімкнені у стані спокою.

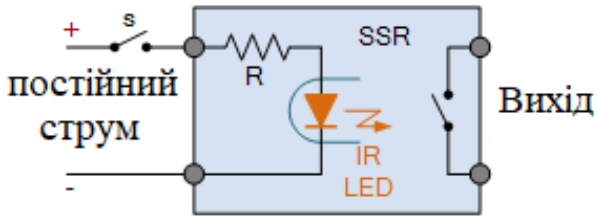

Рисунок 2.25 – Схематичне відображення твердотільного реле

Основна переваги твердотільних реле полягає в повній відсутності рухомих частин, які зношуються. Це означає, що вони не стикаються з проблемами відскоку контактів та здатні здійснювати перемикання набагато швидше та довговічніше, ніж механічні реле.

Ключовий компонент твердотільного реле можна назвати оптоізолятор, що включає один інфрачервоне джерело світла з одного боку та фоточутливий пристрій іншого. Цей оптоізолятор служить для ефективної ізоляції вхідного та вихідного областей пристрою.

Світлодіодне джерело світла підключене до секції вхідного приводу твердотільного реле створює оптичний зв'язок через проміжок до сусіднього фоточутливого транзистора. При проходженні струму через світлодіод він світиться та його світло фокусується через проміжок на фототранзистор. Завдяки застосуванню твердотільного реле, можна досягти енергетичної ефективності використовуючи напругу 3.3В. В структурній схемі ми маємо ще клапан рідини та клапан газу, але приділяти їм увагу на цілий розділ немає необхідності, бо їх принцип роботи аналогічний роботі реле.

2.5.6 Платформа NodeMCU

Модуль ESP12E розроблений командою Ai-thinker та виготовлений в Китаї компанією Espressif Systems. В основі цього модуля лежить мікроконтролер ESP8266, обладнаний інтегрованим 32-розрядним мікропроцесором Tensilica L106. Окрім вбудованого Wi-Fi, цей мікроконтролер відрізняється відсутністю флеш-пам'яті на кристалі; програми користувача виконуються зовнішньою флеш-пам'яттю інтерфейсом SPI [26].

NodeMCU – це платформа, яка об'єднує всю необхідну обв'язку для модуля ESP12E, вона спеціально розроблена для застосування в IoT. Вона включає в себе регулятор напруги, пам'ять та USB-конвертер CH340 для запису програмного забезпечення через порт програмування micro-USB. NodeMCU дуже популярний і використовується широко, тому для нього існує велика кількість документації, оскільки проект є відкритим [4].

Кожен із 17 GPIO можна налаштувати з внутрішнім стягуванням, підтягуванням або встановити високий опір. При ініціалізації GPIO як входу, дані зберігаються у відповідних програмних регістрах. Також є можливість конфігурувати вхід для виявлення переривань по фронту або рівню – підсумовуючи, GPIO є дво-направленими, неінвертуючими та три-становими [3].

У модулі ESP12-E використовується вбудований 32-розрядний процесор архітектури RISC AVR Tensilica L106, який забезпечує низьке енергоспоживання та досягає максимальної тактової частоти в 159,9 МГц. Використання операційної системи реального часу та стеку Wi-Fi дозволяє ефективно використовувати 79,9% обчислювальної потужності кристалу [27].

Модуль вже готовий для програмування, прошивки та розробки користувацьких додатків. Центральний процесор включає наступні інтерфейси:

- інтерфейси RAM/ROM;
- інтерфейс оперативної пам'яті;
- інтерфейс AHB.

ESP12-E має Wi-Fi інтегроване в кристалі, а також контролер пам'яті та блоки пам'яті, включаючи SRAM і ROM. Центральний процесор отримує доступ до цих блоків пам'яті за допомогою інтерфейсів iBus, dBus і AHB. Всі блоки пам'яті можна звертатися за запитом, і контролер пам'яті формує конвеєрну чергу відповідно до часу виконання, коли ці запити надходять від процесора. Функціональна блок-діаграма ESP12-E зображена на рис. 2.26.

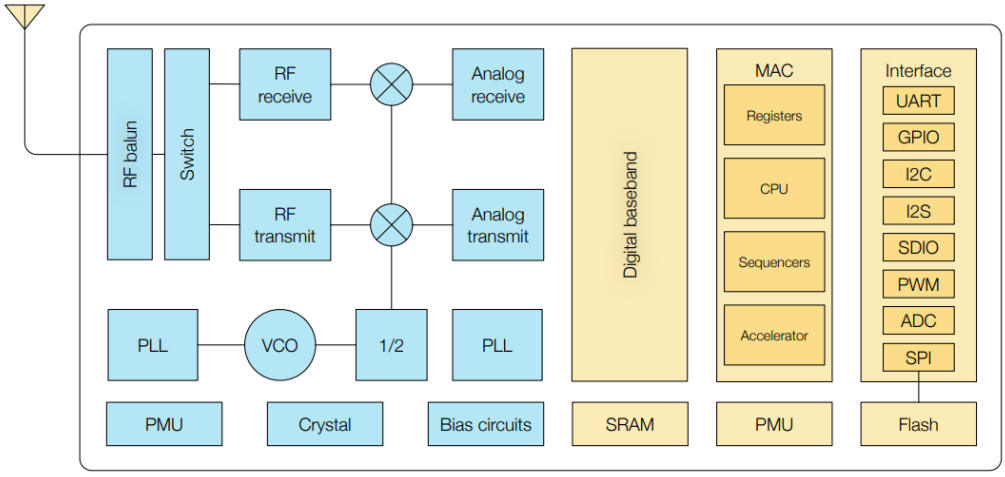

Рисунок 2.26 – Функціональна блок-діаграма ESP12-E [27]

Відповідно до нашої поточної версії SDK, доступний для користувачів простір SRAM призначається:

- розмір ОЗП – 50 КБ. Коли ESP8266 працює в стаціонарному режимі та під'єднується до роутера або хабу, максимальний об'єм пам'яті, доступний у розділі Heap + Data, становить приблизно 50 КБ;
- на кристалі немає ПЗУ. Це означає, що користувацька програма повинна зберігатися у зовнішньому SPI Flash, цю проблему вирішує використання саме платформи Node-MCU.

ESP8266 реалізує TCP/IP і повний протокол WLAN MAC 802.11 b/g/n, табл. 2.3. Він підтримує операції Basic Service Set, STA і SoftAP в рамках функції розподіленого керування. Керування живленням обробляється з мінімальною взаємодією з хостом, щоб мінімізувати період активної роботи. [4]

Таблиця 2.3 – Параметри Wi-Fi на ESP-12E [4]

Категорія	Параметр
Протокол	802.11 b/g/n (HT20)
Діапазон частот	2.4 GHz ~ 2.5 GHz (2400 MHz ~ 2483.5 MHz)
Потужність передавача	802.11 b: +20 dBm
	802.11 g: +17 dBm
	802.11 n: +14 dBm
Чутливість приймача	802.11 b: –91 dbm (11 Mbps)
	802.11 g: –75 dbm (54 Mbps)
	802.11 n: –72 dbm (MCS7)
Шифрування	WEP/TKIP/AES
Мережеві потоколи	IPv4, TCP/UDP/HTTP

Платформа NodeMCU до класу енергоефективних, має в своїй будові регулятор напруги 3.3В, перерахуємо способи живлення:

– подавати 5-18 В через контакт Vin;

– 5В через USB-раз’єм чи контакт VUSB;

– 3,3В через контакт 3V.

Зобразимо на рис. 2.27 розпіновку котактів платформи NodeMCU та принципову схему модуля NodeMCU на рис. 3.12.

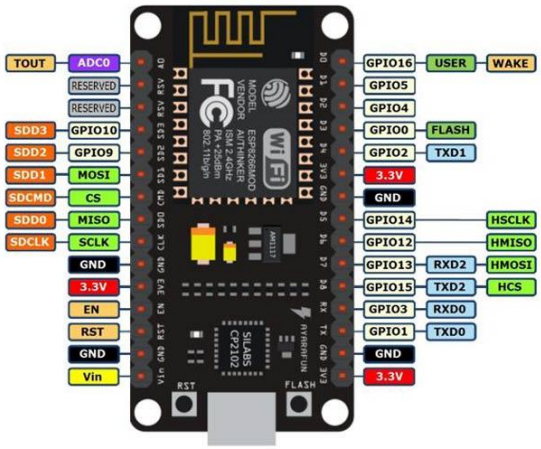

Рисунок 2.27 – Розпіновка платформи NodeMCU

2.5.7 Центральний HUB SHS Raspberry Pi 3 B

Протягом останніх 8 років, все більше популярності набуває маленький комп'ютер Raspberry Pi, динамік продажів якого набирає не абияких обертів. Всього таких комп’ютерів продано близько 19 мільйонів одиниць, з них 9 мільйонів лише за два роки [28]. Компанія завдячує такому успіху моделі Raspberry Pi 3 Model B+, яка стала дуже популярною серед користувачів, її зовнішній вигляд та інтерфейси наведено на рис. 2.28.

Дуже компактний комп’ютер Raspberry Pi 3 B+ має в своїй будові розпаяну оперативну пам’ять формату DDR2-SODIMM, ARM-процесор, eMMC Flash і допоміжні лінійні стабілізатори. Цей набір характеристик дозволяє розробнику використовувати потужність компонентів, стек обладнання та програмного забезпечення Raspberry Pi у власних проектах з компактним форм-фактором.

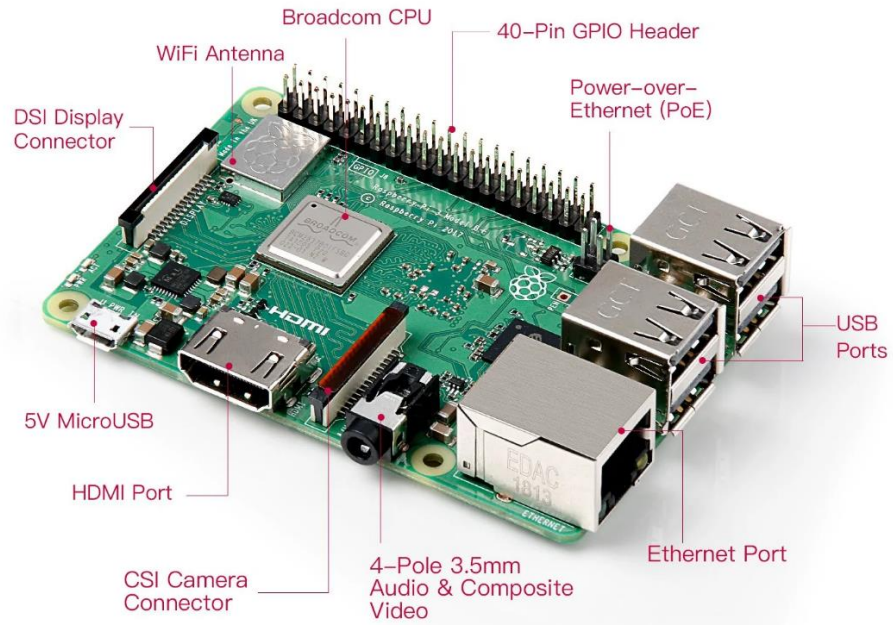

Рисунок 2.28 – Основні інтерфейси Raspberry Pi 3 B+ [28]

Raspberry Pi 3 B+, чия структурна схема представлена на рис. 2.29, обладнаний процесором BCM2837, 1 ГБ розпаяної оперативної пам'яті формату LPDDR2, але без вбудованого Flash-накопичува. Це зроблено не для економії, а для більшої модернізації користувачем власної системи, бо плата має контакти

інтерфейсу SD/eMMC, доступні для користувача, щоб підключити власний пристрій SD/eMMC або ж USB-інтерфейс куди можна підключати жорсткі диски ємністю більше 1 ТБ. Важливо відзначити, що в новіших ревізіях Raspberry Pi 3 B+ ставиться новіший процесор BCM2837 замість BCM2835, який використовувався в попередніх ревізіях Raspberry Pi.

Головною різницею покращенням xx37, стала можливість використання більшого об’єму оперативної пам'яті (до 1 ГБ), ЦП було оновлено з одноядерного до чотирьох-ядерного Cortex A53 із виділеним 512 КБ кеш-пам'яті L2. BCM2837 включає дво-ядерний графічний співпроцесор Video Core IV® Multimedia, який підтримує Open GL ES 2.0, апаратне прискорення Open VG та 1080p30 H.264 декодування [29].

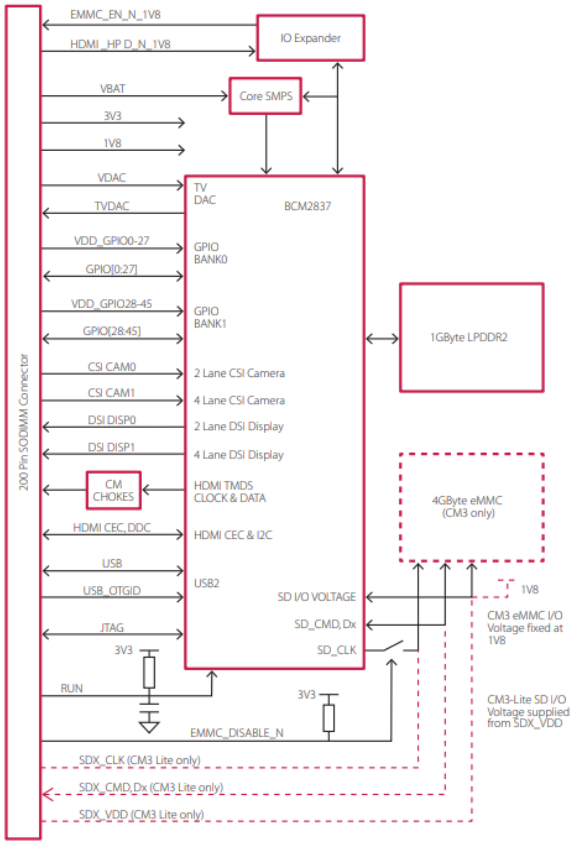

Рисунок 2.29 – Структурна схема Raspberry Pi 3 B+ [29]

Cortex-A53 – процесор середнього класу з низьким енергоспоживанням, який реалізує архітектуру ARMv8-A. Процесор Cortex-A53 має від одного до

чотирьох ядер, кожне з яких має систему пам'яті L1 і один спільний кеш L2. На рис. 2.30 показано приклад конфігурації Cortex-A53 MPCore з чотирма ядрами та інтерфейсом ACE або CHI.

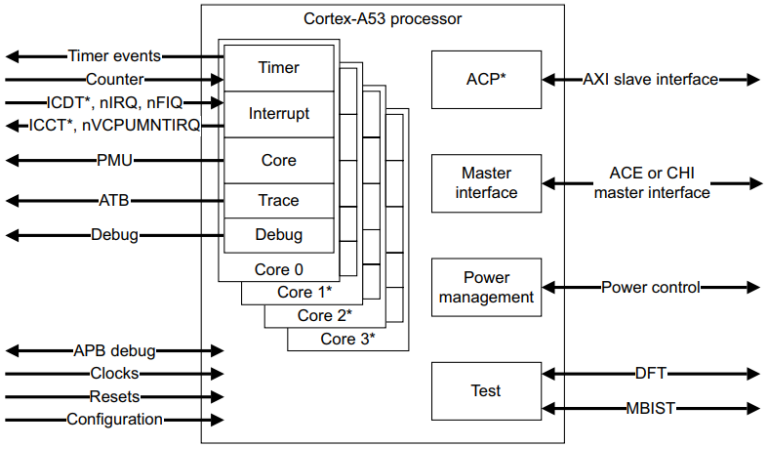

Рисунок 2.30 – Конфігурація процесора Cortex-A53

Додаткове розширення шифрування Cortex-A53 MPCore підтримує криптографію ARMv8. Це додає нові інструкції A64, A32 і T32 в розширений SIMD.

В оновленій версії Pi 3 B+, помітно покращено вбудований контролер мережі. Тепер, замість контролера 10/100 Мбіт, чіпсет Pi обладнано контролером 10/100/1000 Gigabit LAN. Як показує порівняльна характеристика мережі для різних версій плат Raspberry, наведена на рис. 2.31, модель B+ забезпечує значно більшу пропускну здатність.

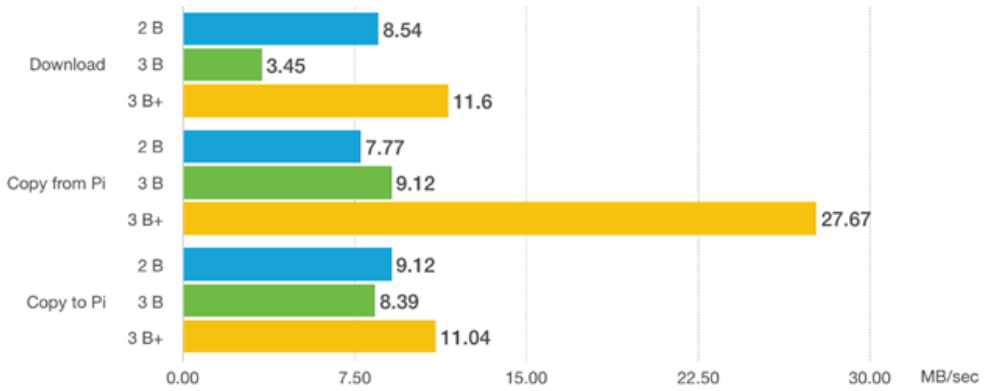

Рисунок 2.31 – Швидкість LAN різних версій плат Raspberry

Продуктивність Wi-Fi в моделі 3B+ перевершує модель 3B, що видно з порівняльної характеристики на рис. 2.32. Додатково, на відміну від вимогливих налаштувань Wi-Fi в моделі 3B, де доводилося мати справу з дивними помилками, налаштування для 3 B+ зводилось до редагування всередині утиліти raspi-config з командного рядка.

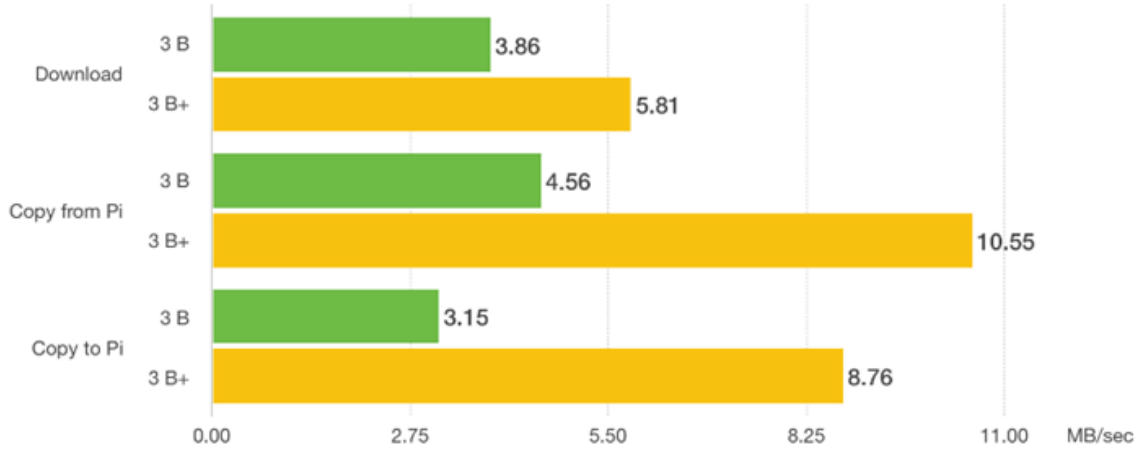

Рисунок 2.32 – Швидкість Wi-Fi різних версій плат Raspberry

Кожне покоління Raspberry має вищу частоту процесору ніж в попередника, а також нові системи, які потребують постійного живлення просто для роботи в режимі очікування і 3 B+ не є винятком. Тести з енергоспоживанням різних версій Raspberry наведені на рис. 2.33.

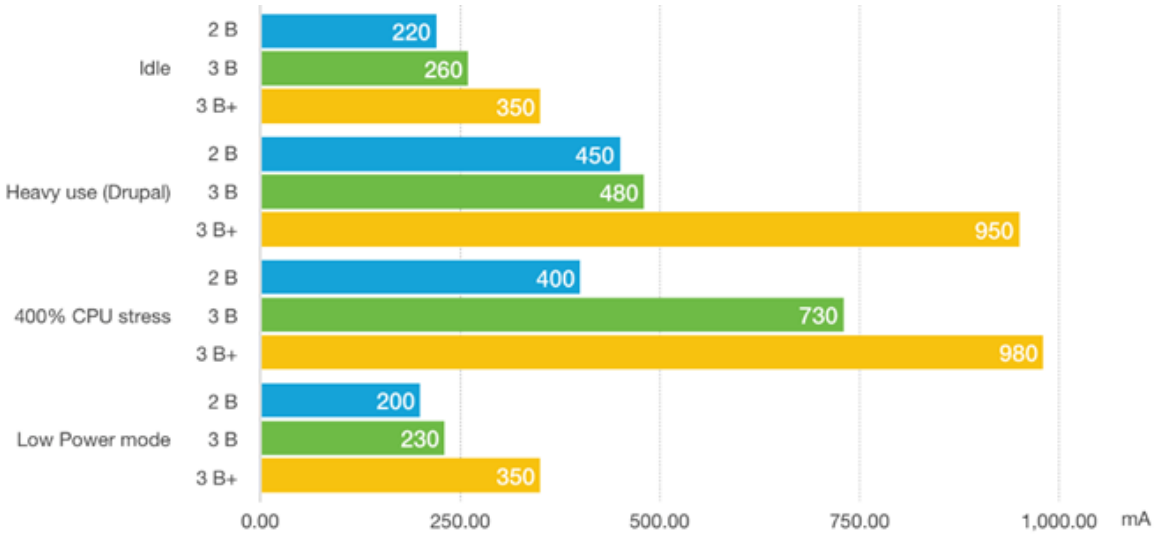

Рисунок 2.33 – Порівняльна характеристика енергоспоживання Raspberry

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ SHS

3.1 Розробка структурної схеми SHS

Відповідно до технічного завдання, необхідно розробити систему, що відповідає базовим потребам середнього користувача з інвалідністю. Основні вимоги до цієї системи включають зручність, швидкість та доступність.

Система має бути орієнтована на використання особами з обмеженими можливостями, надаючи їм можливість легко та ефективно користуватися всіма її перевагами. Основні параметри SHS включають:

- простота використання;
- легке додавання нових, готових пристроїв;
- безпровідний режим роботи;
- локальний режим роботи;
- автономність всієї системи;
- створення бекапу;
- компактність;

SHS повинна забезпечувати виконання наступних корисних функцій, та їх похідних:

- керування освітленням в 3-х кімнатах, з відображенням стану;
- відслідковування стану 2-х вікон в квартирі;
- віддалене керування вхідним замком;
- керуванням газового чи електричного котла;
- відслідковування температури та вологості в 3-х кімнатах квартири;
- відслідковування протікання рідин та газів в 2-х кімнатах;
- застосування аварійних заходів, при протіканні рідини або газів;

На основі перелічених вище функцій та параметрів, складемо структурну схему «Інформаційної технології управління розумним будинком» та зобразимо її на рис. 3.1.

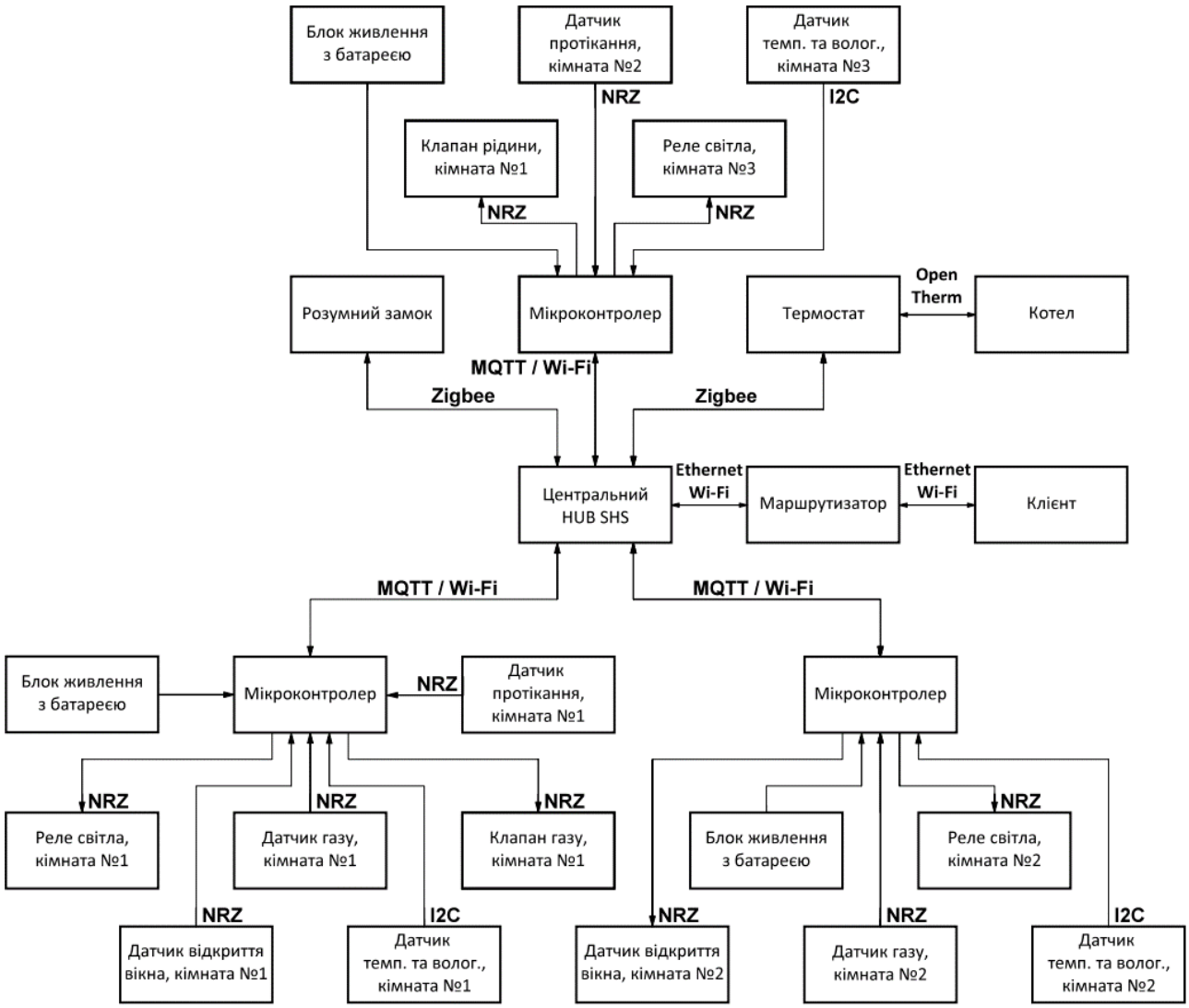

Рисунок 3.1 – Структурна SHS

Структурна схема побудована на основі поширеного планування український однокімнатних квартир.

Кімната №1 – це кухня, де розташована газова плита та котел, якщо викид газу буде з квартири власника, то тут буде найвища концентрація і тут її буде легко виявити. Газова труба з лічильником зазвичай також проходить через кухню, тому розташування тут автоматичного клапану – раціональне рішення. Датчик протікання рідини, тут необхідний для установки під котел, пральну машину, посудомийка або ж раковину, де він буде фіксувати протікання та передавати на центральний хаб сигнал тривоги.

Кімната №2 – це основна кімната, де користувач проводить свій основний час. Наявність тут датчика газу, відіграє запобіжну роль в системі, він остаточно закриває сценарій отруєння користувача газом під час сну та у випадку, коли газ потрапив до квартири від сусідів. Датчики температури та вологості, дозволяють користувачеві відслідковувати параметри мікроклімату та реагувати на них в ручну або ж за допомогою автоматики.

Кімната №3 – це ванна кімната, де може бути розташована душова кабіна, ванна, пральна машина, бойлер і т.д. Тут також повинен бути встановлений датчик протікання, він допоможе вберегти користувача від великих збитків. Зазвичай, вхідна труба водопостачання потрапляє в квартиру саме у ванній кімнаті (або ж власники квартир просто встановлюють тут ванну кімнату, для зменшення витрат на встановлення зайвих труб), тому тут і буде встановлений клапан аварійного перекриття водопостачання. Важливо відмітити, на нашій структурній схемі зображений лише один клапан рідини, зроблено це з огляду на нові квартири, де немає центрального опалення і до квартири підведена лише холодна вода. Для квартир з централізованим опаленням, рекомендується встановити такий клапан на дві вхідні труби.

Блок термостату винесений з меж впливу локальних збирачів інформації, бо він зазвичай встановлюється в прохідних кімнатах. Багато сучасних котлів підтримують протокол Open Therm, тому з підключенням термостату не повинно виникнути великих проблем. В ситуації, коли котел має лише ручне керування, можна реалізувати керування на основі SHS, датчика температури, та двох-канального реле.

Температура буде братися з датчика кімнати, де необхідно підтримувати температуру. При зменшенні температури за задане параметр «X», двоканальне реле замикається і котел починає нагрівати приміщення, при перевищенні температури «Y», відповідно вимикає реле і котел. Петлю гістерезису можна налаштовувати програмно, але такий метод керування краще не використовувати як правило, бо він доволі архаїчний.

3.2 Розробка блоку керування «Кімната №1»

Згідно структурної схеми на рис. 3.2 та плануванню, «Кімната №1» - це кухня, на якій нам потрібно реалізувати відслідковування температури, вологості, концентрації метану та у разі перевищення небезпечної концентрації перекривати клапан ввідної газової труби, фіксувати витік рідини під місцями потенційного витоку, такі як раковина, пральна машина, посудомийна машина або інше.

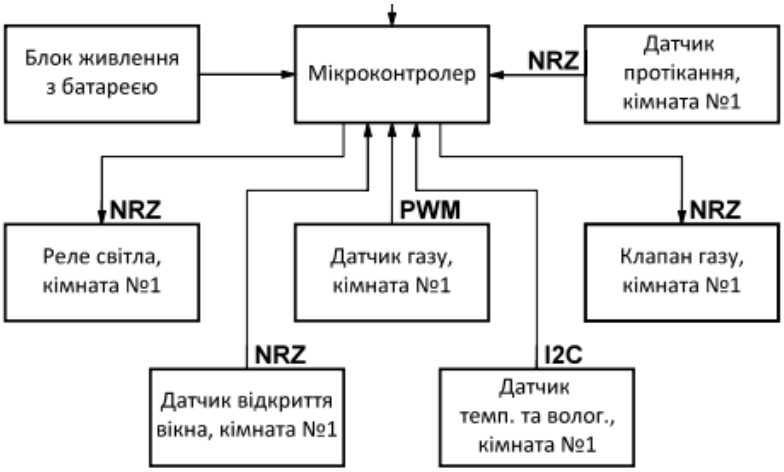

Рисунок 3.2 – Структурна схема блока керування «Кімната №1»

Для коректного вирішення проблеми з природним газом, нам необхідно визначити, яка концентрація метану є небезпечна для людини. Згідно «Північно-Східного міжрегіонального управління Державної служби з питань праці», метан може викликати задуху при 20% концентрації, але при цьому вибухонебезпечна концентрація побутового газу складає від 5 до 15%, тобто гранично допустима концентрація дорівнює 300 мг/м^3. Нам потрібно налаштувати наш датчик, щоб він спрацьовував на концентрації метану 250 мг/м^3.

Датчик MQ-4, який ми застосовуємо в нашому проекті вимірює концентрацію в одиницях виміру ppm, наведемо формулу (3.1) для переводу в мг/м^3.

$$ppm = \frac{{}^{\text{мг}}/_{\text{м}^3}}{1{,}245} \tag{3.1}$$

Підставивши у формулу відповідне значення граничної концентрації, отримуємо ≈ 200 ppm. Сам датчик має в своїй конструкції компаратор, який при перевищенні розрахованого порогу матиме на своєму виході логічну одиницю, але перед цим поріг необхідно налаштувати, це потрібно для використання цифрового входу мікроконтролера, замість піна з аналогово-цифровим перетворювачем. Для цього необхідно підключити датчик до цифрового входу та одночасно до аналогового входу, відслідковуючи поточний рівень метану в моніторі порту, відрегулюємо поріг спрацювання потенціометром. Використаємо простий код, наведений нижче:

```
const int AO_Pin=0;
const int DO_Pin=8;
const int LED_Pin=13;
int threshold;
int AO_Out;
 // Set up
void setup() {
Serial.begin(115200);
pinMode(DO_Pin, INPUT);
pinMode(LED_Pin, OUTPUT);
}
// Main loop
void loop()
{
AO_Out= analogRead(AO_Pin);
threshold = digitalRead(DO_Pin);
Serial.print("Threshold: ");
Serial.print(threshold);
Serial.print(", ");
Serial.print("Methane Conentration: ");
Serial.println(AO_Out);
if (threshold == HIGH){
digitalWrite(LED_Pin, HIGH);
}
else{
digitalWrite(LED_Pin, LOW
}
delay(1000);
}
```

Код програми наведено в «Додатку А», опишемо принцип його роботи. Мікроконтролер підключається до хабу, який розгорнутий на потужностях Raspberry Pi 3 B з програмним забезпеченням «Home Assitant», через Wi-Fi. При увімкненні NodeMCU, спочатку проходить ініціалізація змінних, бібліотек та налаштування робочих пінів на вхід та вихід, а потім робочий код працює в безкінечному циклі.

Датчик відкриття вікна на модулі KY-025, являє собою геркон та неодимовий магніт. При закритому вікні, він передає на цифровий вхід мікроконтролера значення «false», при відкритті повертає «true», опитування датчика проводитися 1 раз в секунду. На мікроконтролері інформація обробляється і відправка його стану по MQTT у відповідний топік буде передаватися лише при зміні стану.

Датчик протікання, який являє собою струмопровідну пластину та компаратор, повертає «true» при зафіксованому протіканні, в нормальному ж стані датчик повертає «false» опитування датчика проводитися 1 раз в секунду. На мікроконтролері інформація обробляється і відправка його стану по MQTT у відповідний топік буде передаватися лише при зміні стану, адже електронний клапан який перекриває водопостачання знаходиться під керуванням іншого блоку.

Датчик SHT30 відслідковує температуру та вологість в кімнаті, спілкування з мікроконтролер відбувається за протоколом I^2C. Опитування датчика проводиться 1 раз секунду. На мікроконтролері інформація обробляється і відправка даних для відображенні в хабі, по MQTT у відповідний топік, буде передаватися по розрахунковим даним раз в 262,5 секунд.

Реле світла, яка повинно бути вмонтоване в розрив фазі, керується лише по протоколу MQTT. При надходженні керуючого сигналу, вмикання світла відбувається командою «true», а вимикання командою «false».

Датчик газу MQ-4, після його калібрування та налаштування можна підключати до цифрового порту мікроконтролера. При нормальній концентрації

метану (<250ppm), він подає сигнал «false» на порт мікроконтролера, але при збільшенні концентрації вище допустимого порогу, він передасть сигнал «true». Цей датчик знаходиться з електронним клапаном газу в одному рівні підпорядкування, тому інформація з цього датчика не відправляється на центральний хаб

Електронний клапан газу, в початковому своєму стані – відкритий, тобто при надходженні на нього сигналу «false», ніякої дії виконано не буде. При надходженні на нього сигналу «true», клапан закриється і перекриє газ. Дозволяючий сигнал на цю дію може прийти з локального датчика MQ-4, який знаходиться з клапаном в одній локальній системі та з центрального хабу, з датчика який розташований в блоці керування «Кімната №2».

Опитування датчиків 1 раз в секунду та відправка даних лише при їх зміні, зроблено з ціллю економії електроенергії (особливо важливо при роботі від батареї) та зменшити навантаження на канал передачі.

3.3 Розробка блоку керування «Кімната №2»

Згідно структурної схеми на рис. 3.3 та плануванню, то «Кімната №2» - це жила кімната, в якій користувач проводить більшу частину часу.

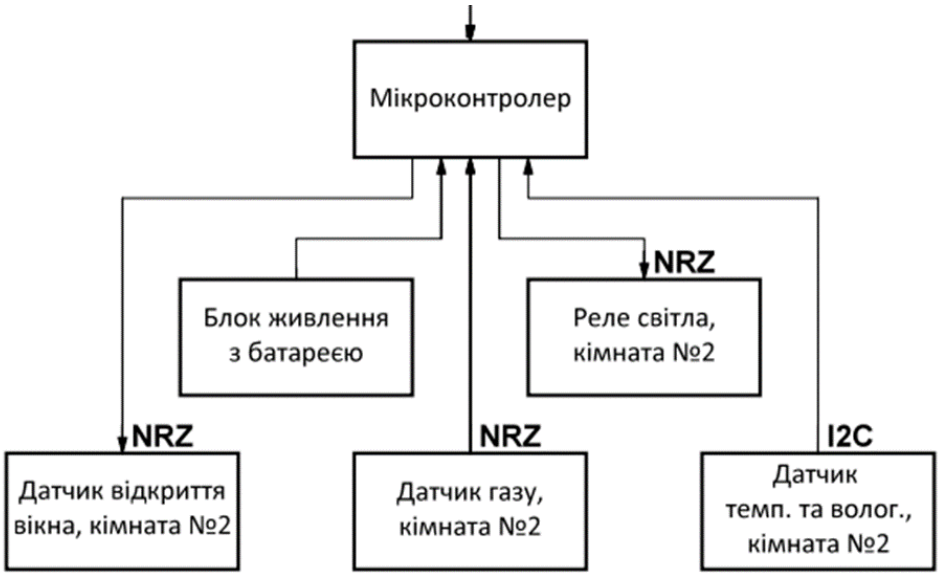

Рисунок 3.3 – Структурна схема блока керування «Кімната №2»

Зі структурної схеми видно, що вона подібна до схеми блоку керування «Кімната №1», але має розбіжності в принципі роботи, код програми наведено в «Додатку Б». Блок керування має датчик природного газу, який також потрібно відкалібрувати за методою описаною в розділі 3.3, до одного порогу спрацювання.

Датчик відкриття вікна, реле для керування освітленням, датчик температури та вологості працюють за алгоритмом, який викладений в розділі 3.3. Датчик природного газу, виконує роль страхуючого, бо знаходиться в кімнаті, де немає джерел витоку газу, але тут велику кількість часу (навіть вночі) знаходиться клієнт. Тому для його безпеки цей датчик відслідковує рівень метану в жилому приміщенні. В разі фіксації ним перевищення заданого порогу концентрації метану, він через MQTT протокол, через центральний хаб передасть керуючий сигнал на електронний клапан, який знаходиться під керуванням блоку «Кімната №1»

3.4 Розробка блоку керування «Кімната №3»

Згідно структурної схеми на рис. 3.4 та плануванню, то «Кімната №3» - це ванна кімната, в якій користувач, в основному, взаємодіє з водою.

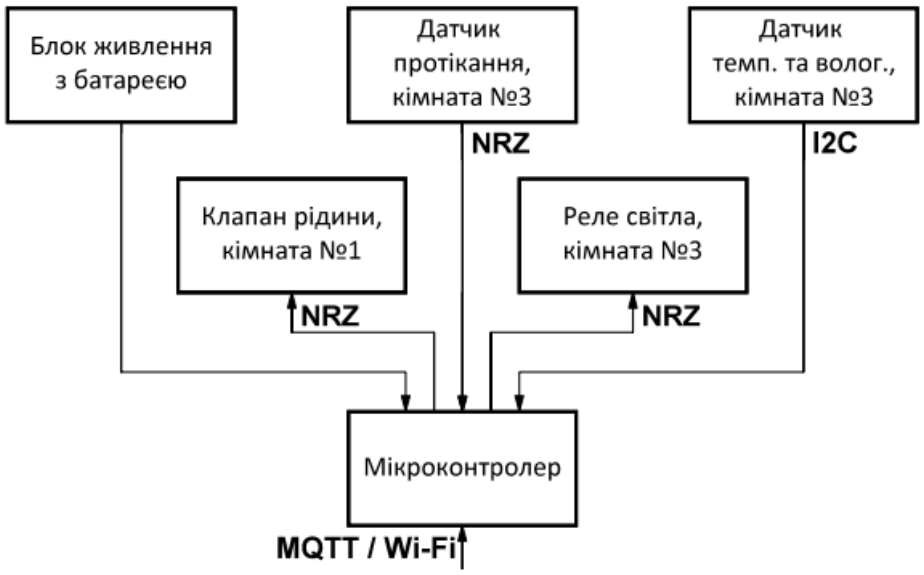

Рисунок 3.4 – Структурна схема блока керування «Кімната №3»

Зі структурної схеми видно, що вона також подібна до схеми блоку керування «Кімната №1» та «Кімната №2» за винятком двох елементів схеми, код програми наведено в «Додатку В».

Реле для керування освітленням, датчик температури та вологості працюють за алгоритмом, який викладений в розділі 3.3.

Датчик протікання, який являє собою струмопровідну пластину та компаратор, повертає «true» при зафіксованому протіканні, в нормальному ж стані датчик повертає «false», опитування датчика проводитися 1 раз в секунду. В цій же мережі знаходиться електронний клапан, яким повинен закриватися від показників локального датчика протікання, тому далі по MQTT інформація на хаб не переходить. Як описувалося в розділі 3.4, електронний клапан керується також показниками датчика вологи, який розташований в блоці керування «Кімната №1».

У всіх структурних схемах є блок «Блок живлення з батареєю», але він ніде не описаний. Це тому, що він ніяк програмно з платформами не взаємодіє.

Модуль WeMos Battery Shield, який першим своїм входом підключений до літієвого акумулятора ємністю 3200 мАг, з напругою від 3.3 до 4.2 В, іншим входом підключений до основного блоку живлення через microUSB. Його зовнішній вигляд зображено на рис. 3.5.

Рисунок 3.5 – Модуль WeMos Battery Shield

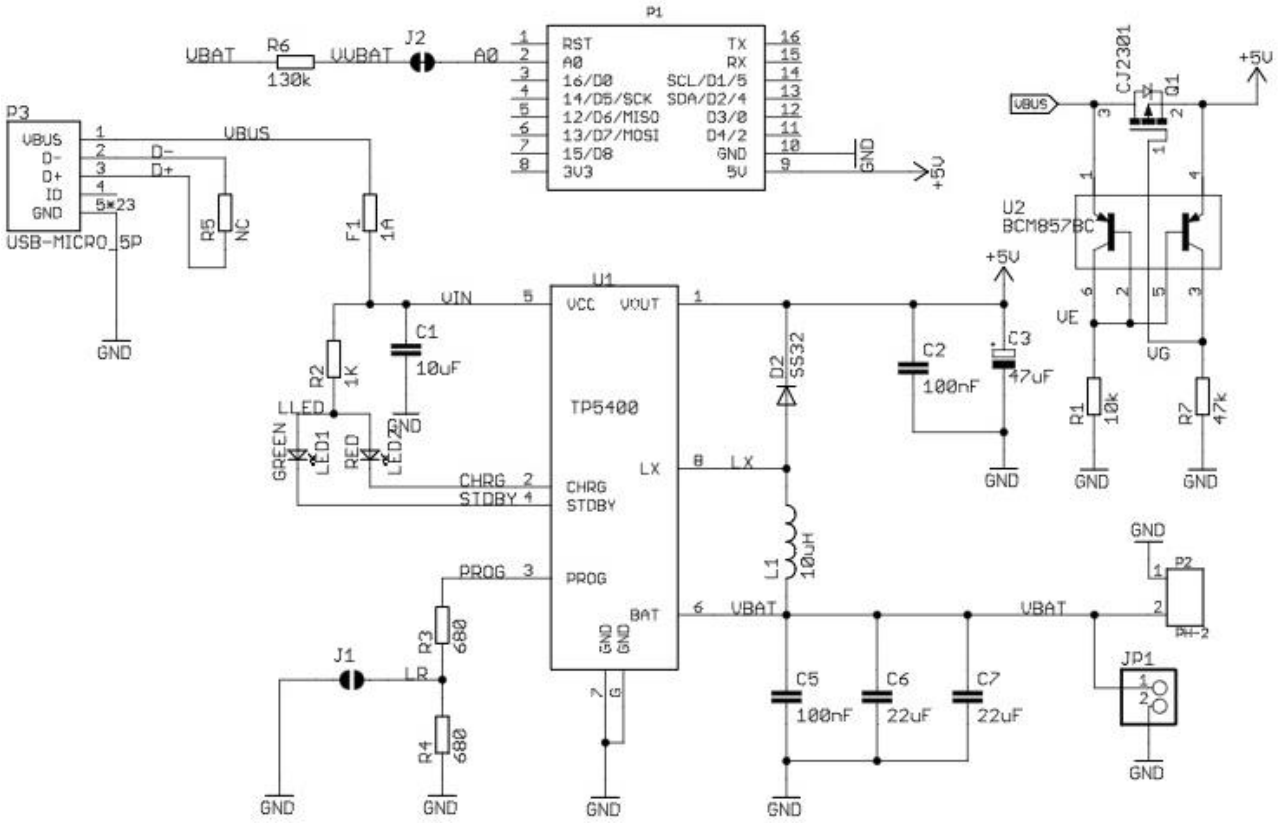

Рисунок 3.6 – Принципова схема WeMos Battery Shield,

У разі наявності основного джерела живлення через роз'єм microUSB, всі підключені пристрої отримуватимуть живлення від нього, а літієвий акумулятор буде заряджатися до напруги 4.2 В. При відсутності основного джерела живлення через роз'єм microUSB пристрої отримуватимуть живлення від акумулятора за допомогою підвищуючого DC-DC перетворювача TP5410 на 5 В. Вихід цього перетворювача підключений до виводу 5V. Якщо відсутня основна напруга живлення через роз'єм microUSB і напруга літієвого акумулятора знизиться нижче 2.7 В, підвищуючий DC-DC перетворювач вимкнеться, і пристрої не будуть живитися.

За замовчуванням літієвий акумулятор заряджається струмом 0.5 А, однак якщо замкнути перемичку J1 на зворотному боці плати, струм заряду збільшиться до 1 А. Принципова схема модуля зображена на рис. 3.6, червоний світлодіод на платі модуля інформує про те, що літієвий акумулятор заряджається, а зелений - про те, що він повністю заряджений.

3.5 Розробка центрального хабу SHS

Центральний хаб SHS – це Raspberry Pi 3 B+ з встановленим туди програмним забезпеченням Home Assistant OS. Варіантів установки існує всього чотири, але саме Home Assistant OS дозволяє використовувати всі функції операційної системи без жодних обмежень [30]. Доречі, такий варіант використання є самим популярним, що можна наочно побачити з інфографіки розробників на рис. 3.7.

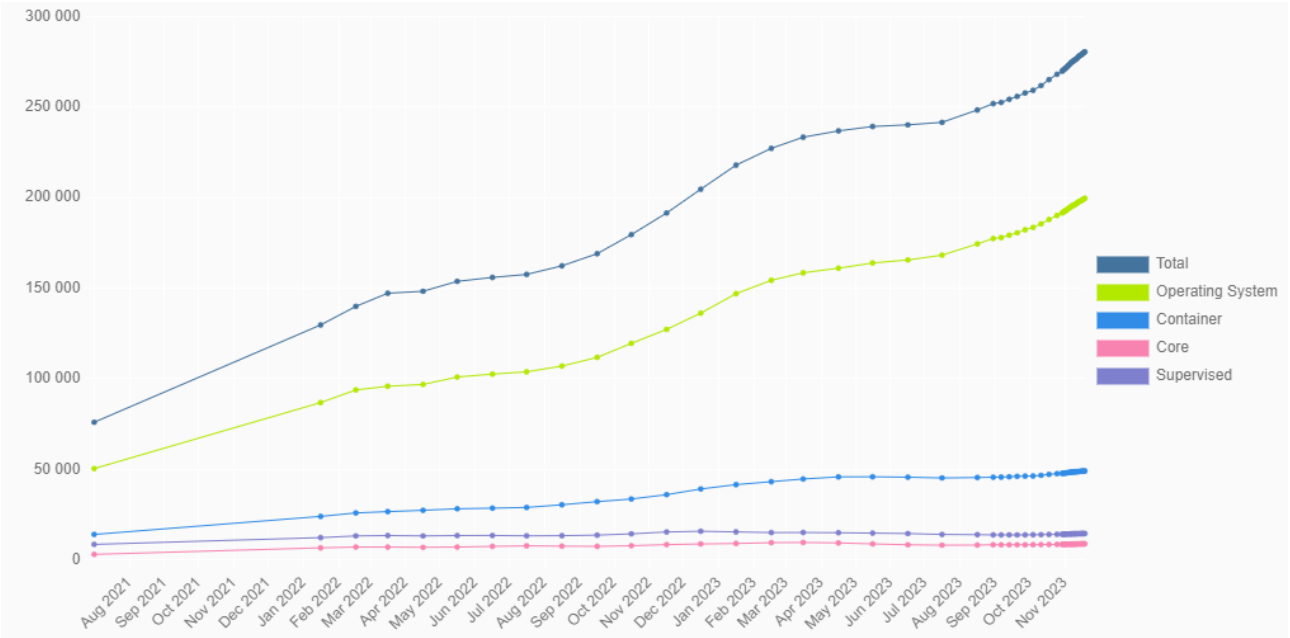

Рисунок 3.7 – кількість встановленнь Home Assistant OS [30]

Для установки операційної системи, нам знадобиться робочий комп'ютер, microSD карта, ethernet кабель та набір спеціалізованих програм.

Перш за все, необхідно завантажити останню версію Home Assistant OS для платформи Raspberry. Це особлива збірка, яка оптимізована під апаратне забезпечення цієї платформи. Завантажується операційна система в zip-архіві, для розпаковки можна використати стандартні засоби Windows 10.

Другим кроком, необхідно підготувати SD-карту, за допомогою спеціалізованої програми Rufus 3.14, до встановлення на неї завантаженої та розпакованої операційної системи. Цей продукт призначений для взаємодії з

усіма типами flash-накопичувачів, дозволяє виконувати форматування сховища в іншу файлову систему з користувацькою конфігурацією розміру кластера та дає можливість обрати схему розділу.

Третім кроком, потрібно здійснити запис операційної системи на підготовлений flash-накопичувач. Для цього будемо застосовувати програму balenaEtcher, процес роботи з якою зображений на рис. 3.8. Головне, знайти свою SD-карту в списку та обрати його, замість якогось з жорстких дисків.

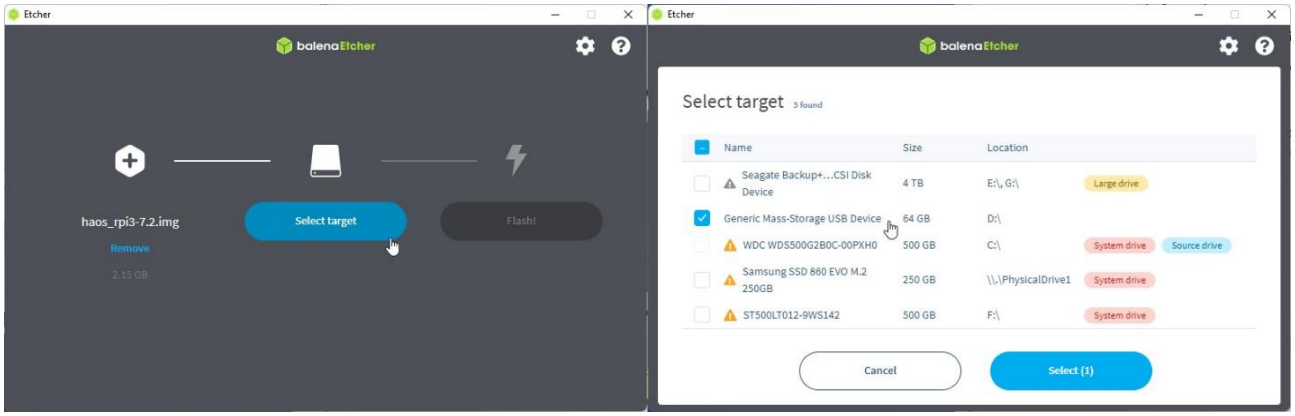

Рисунок 3.8 – Запис образу системи на SD-карту

Четвертим кроком, буде запуск системи, для цього встановлємо flash-накопичувач у вільний порт USB, з'єднуємо маршрутизатор та Raspberry Pi за допомогою ethernet кабелю, і нарешті підключаємо живлення. Home Assistant OS – це Linux-подібна система, тому при першому вмиканні необхідно дати час (близько 5-10 хвилин) на повну ініціалізацію пристрою. Для відслідковування стану завантаження, можна використовувати користувацький інтерфейс маршрутизатору, в ньому повинен з'явитись новий пристрій з призначеною IP-адресою.

На цьому встановлення операційної системи Home Assistant OS завершено, тепер необхідно налаштувати її з огляду на запити клієнта. Для переходу до налаштувань, потрібно ввести в браузері пристрою, який знаходиться в одній локальній мережі, IP-адресу хабу та порт через «:», на якому вже розгорнутий сервер з графічним інтерфейсом.

Якщо все зроблено правильно, то при переході по локальній IP-адресі, в моєму випадку це було 192.168.88.252:8123, буде зустрічати початковий екран, як на рис. 3.9.

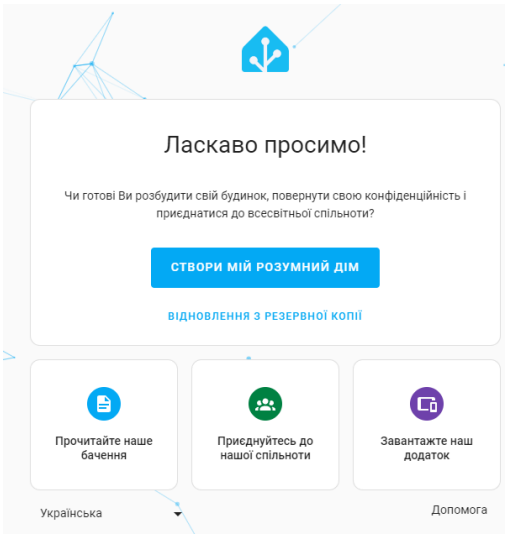

Рисунок 3.9 – Початковий екран при налаштуванні Home Assistant

Тепер потрібно створити обліковий запис користувача, який пізніше стане адміністратором будинку, з можливістю додавання нових пристроїв, редагуванням старих та покращення користувацького інтерфейсу. Процес створення зображений на рис. 3.10.

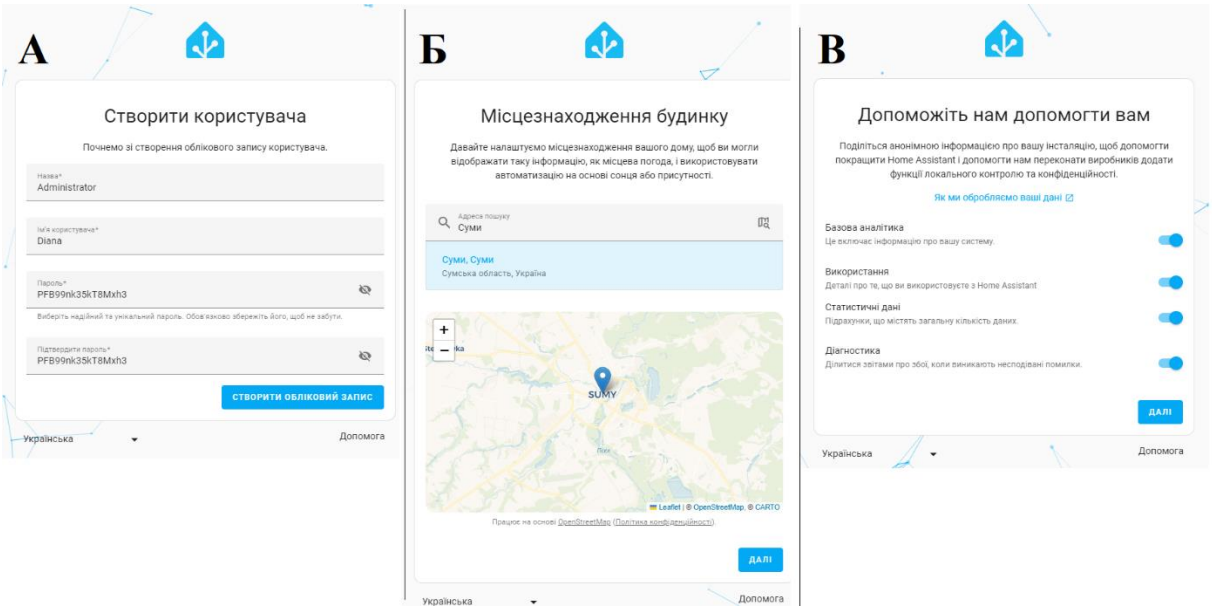

Рисунок 3.10 – Процес створення нового користувача

Після успішного створення логіну, паролю та початкових налаштувань, потрібно зробити з цього простого акаунту – адміністраторський. Це потрібно для захисту початкового стану системи, щоб звичайний користувач випадково не зміг нічого пошкодити. Для цього, потрібно перейти в налаштування акаунту та зробити активним перемикач «Розширений режим», як показано на рис. 3.11.

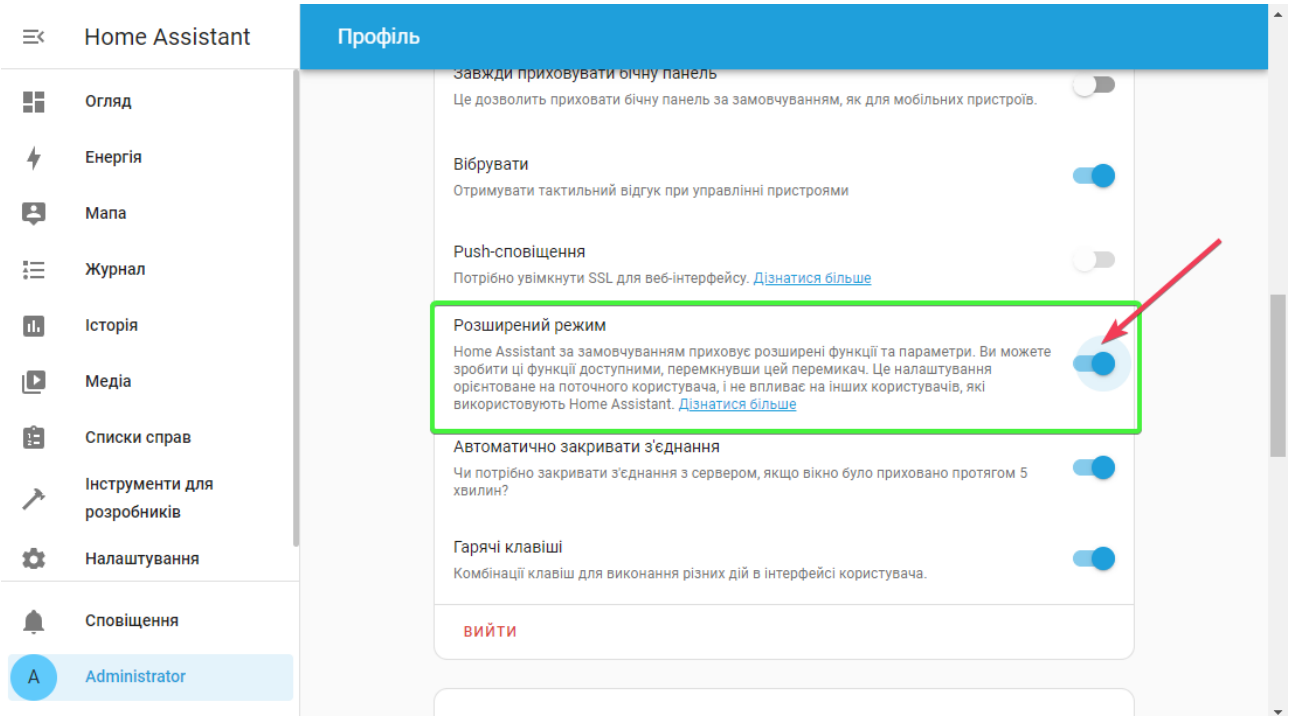

Рисунок 3.11 – Увімкнення розширених налаштування для адміністратора

Згідно нашого технічного завдання, центральний хаб повинен мати можливість працювати з протоколом MQTT, з'єднуватися з фабричними виробами, такі як вхідний замок і термостат по спеціальному протоколу ZigBee та мати зручний інтерфейс.

Для додавання можливості роботи з MQTT протоколом, потрібно розгорнути MQTT-брокера, після чого створити відповідні топіки, куди будуть підключатися підписники, тобто блоки керуваннями різними кімнатами. Щоб зробити це, потрібно лише перейти в меню «Налаштування» (рис. 3.12 А), знайти підменю «Доповнення» (рис. 3. 12 Б), знайти в магазині MQTT-брокер (рис. 3.12 В) та встановити його до себе в систему (рис. 3.12 Г).

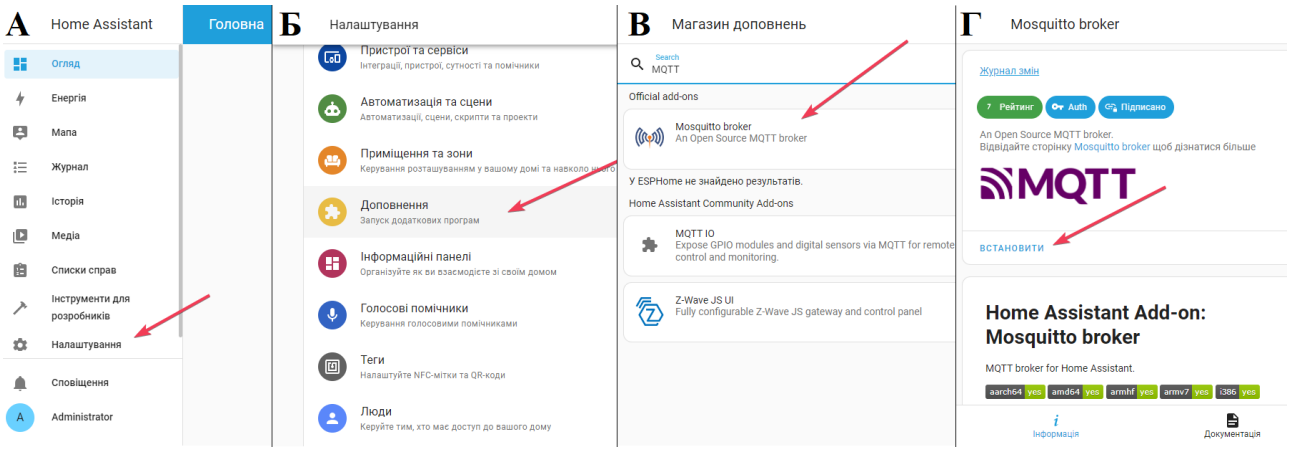

Рисунок 3.12 – Встановлення MQTT-брокеру

Для додавання можливості підключати пристрої по ZigBee протоколу, потрібно також встановити додатковий модуль, але просто з меню «Магазин Доповнень» це зробити не вдасться. Для цього необхідно скопіювати посилання з сторінки ZigBee протоколу на сайту GitHub [31], перейти в «Магазин доповнень», натиснути на піктограму «три крапки» (рис. 3.13 А), вибрати «Репозиторій» (рис. 3.13 Б), вставити туди скопійоване посилання, додати його і лише тоді ZigBee з’явиться в переліку додатків (рис. 3.13 В) і його можна буде встановити. Після цих дій, залишається лише зробити інтерфейс та налагодити роботу MQTT протоколу.

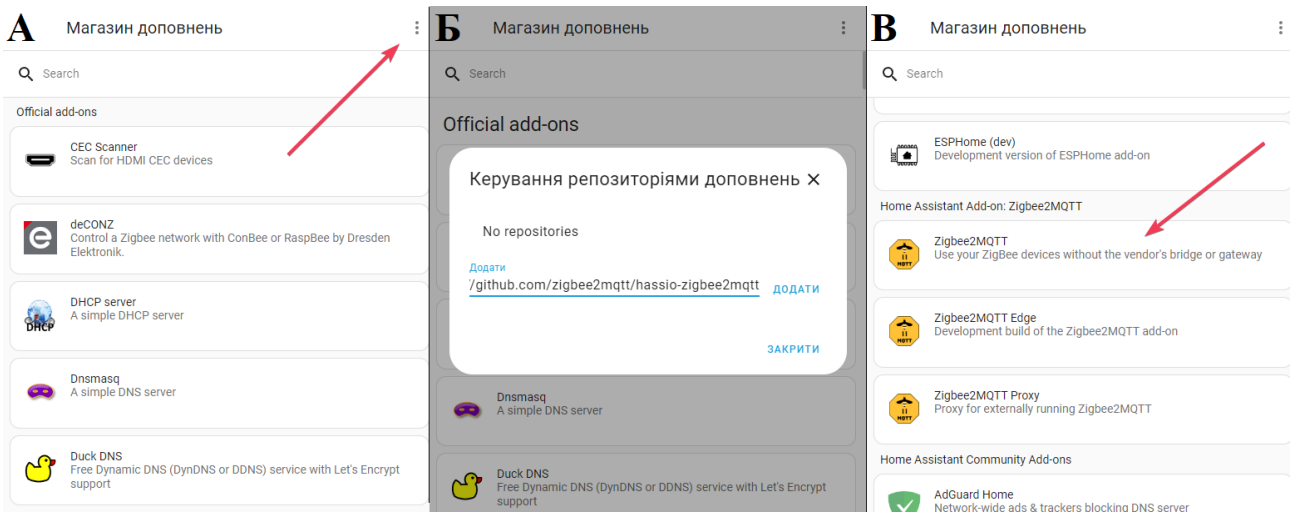

Рисунок 3.13 – Імплементація стороннього доповнення

Для налаштування системи знадобиться ще одна програма – MQTT explorer, це інструмент для візуального дослідження та тестування брокера MQTT. Це допоміжний інструмент, який надає зручний інтерфейс для взаємодії з брокером MQTT, перегляду топіків, відправлення та отримання повідомлень. Спочатку, за допомогою встановленого MQTT-брокера створемо топік home/assistant/room1/leak для відслідковування стану вікна, процес створення зображений на рис. 3.14.

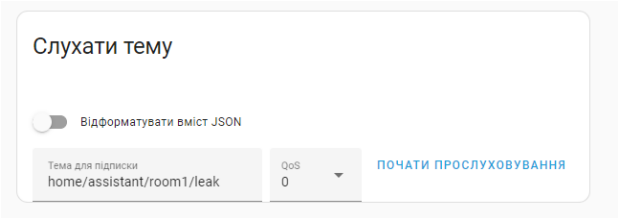

Рисунок 3.14 – Створення та тестування топіку протікання

Тепер за допомогою програми MQTT explorer, яка встановлена на комп’ютері в одній локальній мережі з хабом та вже підключена до брокера за допомогою ip-адреси, логіна та пароля, ми моделюємо стан, що наш датчик протікання який розташований на кухні спрацював і відправив сигнал «true».

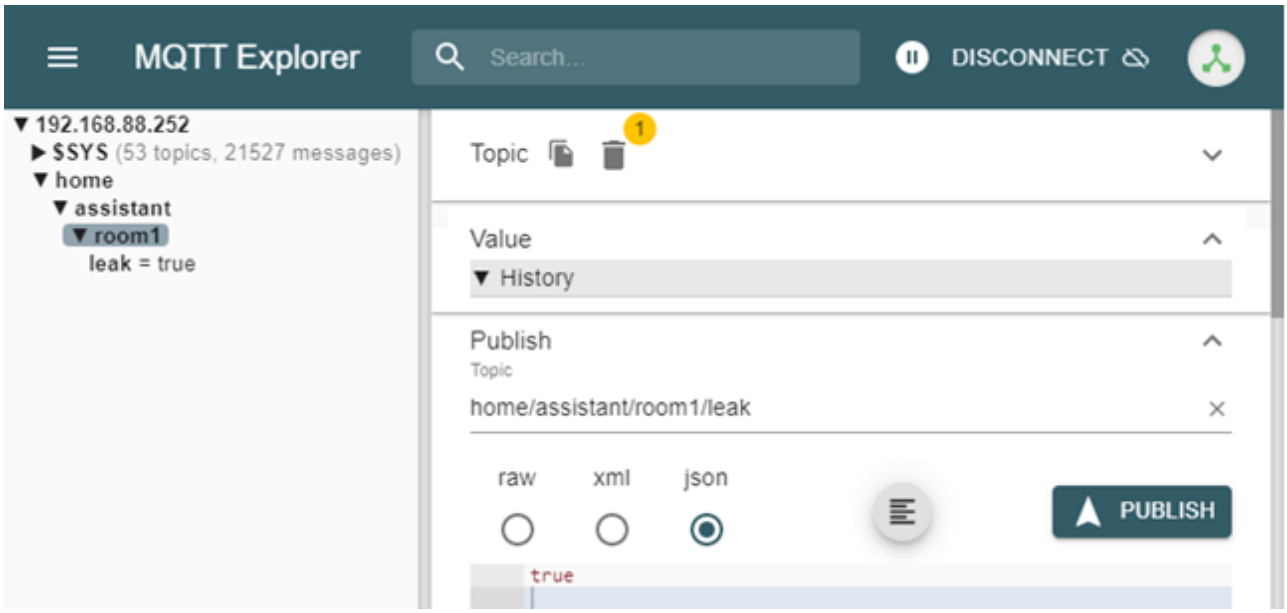

Рисунок 3.15 – Тестування топіку протікання

Налаштування програми MQTT explorer зображені на рис. 3.15, при публікації повідомлення «true» мікроконтролером із запитом на перезапис даних в топіку home/assistant/room1/leak, ми отримали позитивний результат тестування, наша приймальна сторона змінила свій вміст, як показано на рис. 3.16. Це працює і в зворотній бік – при зміні даних на самому брокері, дані передаються на підписника, це потрібно для реалізації вмикання і вимикання світла та інших дій.

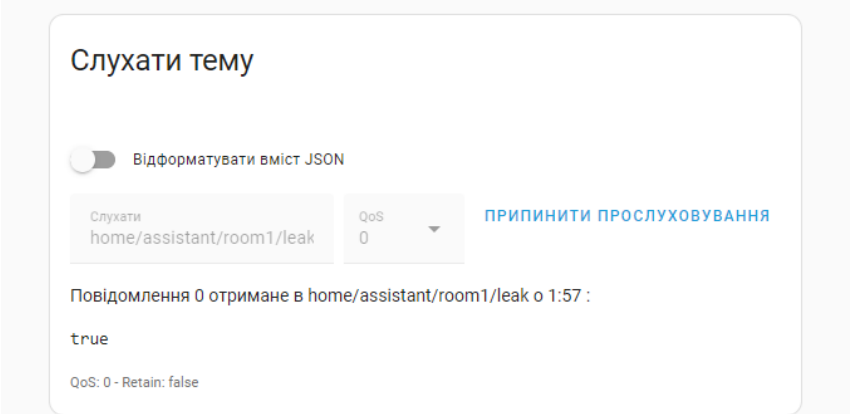

Рисунок 3.16 – Результат тестування каналу

Коли коректну роботу MQTT-брокера перевірено, аналогічним методом створюємо топіки для всіх датчиків в системі та зчитаємо дані. Перевірити результат ножна за допомогою програми MQTT-explorer, для наочності наведено його на рис. 3.17.

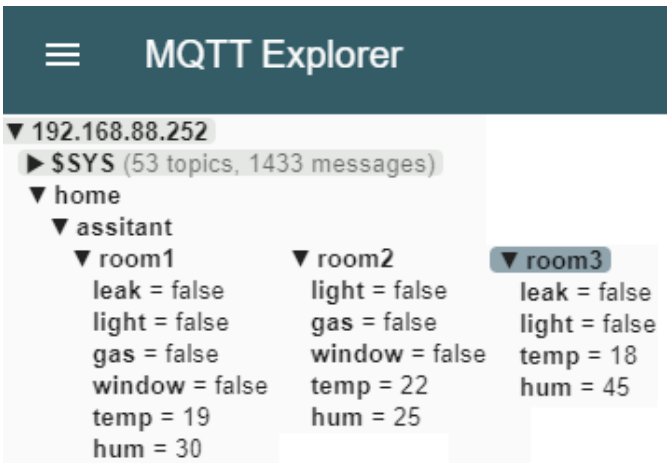

Рисунок 3.17 – Створені топіки для квартири

Тепер потрібно реалізувати взаємодію користувача з MQTT-протоколом, для цього створимо на головній панелі три вимикачі, які будуть керувати освітленням у всіх трьох кімнатах. Створення об’єктів для користувача відбувається за допомогою мови розмітки YAML – це зрозумілий людині читабельний формат обміну даними. Він широко використовується для конфігураційних файлів та структурованого представлення даних, таких як конфігураційні файли для різноманітних програм та платформ, включаючи Home Assistant. На рис. 3.18, зображений графічний інтерфейс, створених нами вимикачів за допомогою YAML, це три кнопки об’єднаних в один блок на 3 стовпці, в «Додатку Г».

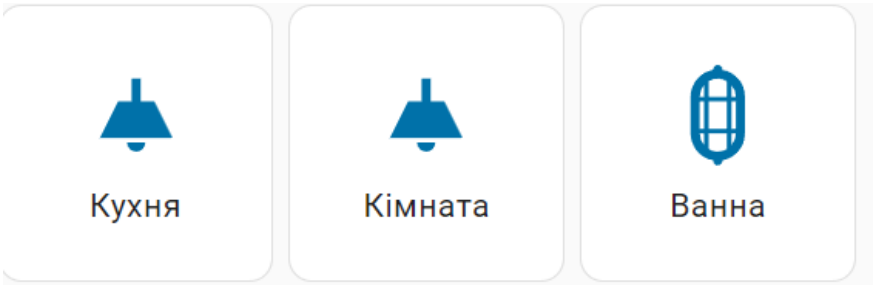

Рисунок 3.18 – Вимикачі світла

Розглянемо частину конфігурації, взаємодії, на прикладі кнопки «Кухня», для головного екрану Lovelace. Картка має наступні параметри:

- type: grid. Вказує на те, що це – картка сітки, яка дозволяє організувати інтерфейс у вигляді сітки;
- cards. Специфікує, які елементи будуть розміщені в цій картці;
- type: button. Визначає, що елемент на цій картці – це кнопка;
- tap_action та hold_action: Вказують на те, які дії повинні викликатися при торканні або утриманні кнопки. У нашому випадку, вони викликають сервіс mqtt.publish, що означає надсилання повідомлення до MQTT брокера;
- Name. Назва кнопки. У нас це "Кухня";

– Icon. Іконка, яка відображатиметься на кнопці. У нас обрана піктограма лампи у вигляді трапеції, її назва у системі – «mdi:ceiling-light»;
– show_state, show_name, show_icon. Вказують на те, чи відображати стан, ім'я та іконку на кнопці;

Отже, при торканні або утриманні цієї кнопки буде викликано сервіс mqtt.publish, що надсилатиме повідомлення до топіка home/assistant/room1/light зі значеннями "true" або "false", в залежності від того, чи це торкання чи утримання кнопки.

За допомогою YAML, продовжуємо наповнювати головний екран елементами керування, фінальний результат зображений на рис. 3.19

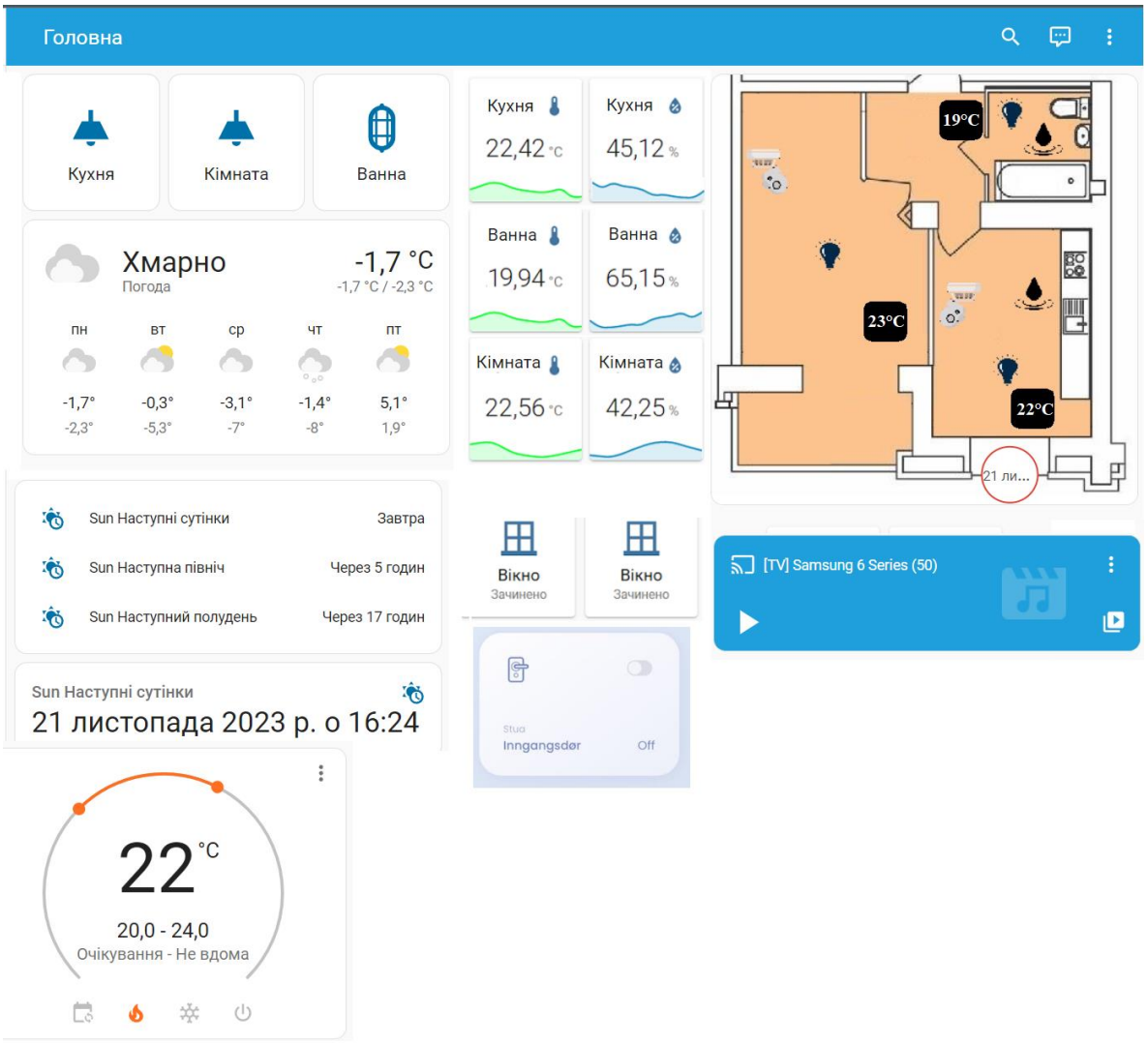

Рисунок 3.19 – Фінальний інтерфейс

ВИСНОВКИ

У кваліфікаційній роботі магістра розроблений прототип SHS для інклюзивних людей, на базі одноплатного комп'ютера Raspberry Pi 3 B+. Він дозволить ознайомити людей з сучасними можливостями автоматизації власної оселі за допомогою IoT.

Проектована система забезпечує базові потреби інклюзивного користувача, конкретно :

- керування освітленням в трьох кімнатах;
- відслідковування температури та вологості на кухні;
- відслідковування температури та вологості у ванній кімнаті;
- відслідковування температури та вологості у жилій кімнаті;
- відслідковування рівня метану в двох кімнатах;
- відслідковування протікання в двох кімнатах;
- автоматичним перекриттям газу в разі аварійної ситуації;
- автоматичним перекриттям води в разі аварійної ситуації;
- керування термостату;
- керування розумних замком.

Також, наявність можливості додавання великої кількості фабричних виробів вже відомих виробників. Пристрій відповідає технічному завданню.

У першому розділі, було розглянуті системи конкуренти які повністю або частково суміжні з напрямком проектування, їх переваги та недоліки. Розглянуті окремі прилади, які можуть виконувати роль «розумних речей» без інтеграції в глобальну систему. Наведена класифікація людей з інвалідністю, за законом України, що дало можливість краще зрозуміти проблематику та чітко сформувати постановку завдання.

У другому розділі було розглянуто загальний принцип роботи SHS різних виробників. Проаналізовано два найпоширеніших протоколи передачі інформації, які застосовуються в таких цілях – CoAP та MQTT. На основі

отриманої інформації сформована загальна структурна схема SHS, яка допомогла в подальшому в розробці локальної структурної схеми. Проаналізована система керування Home Assistant.

У третьому розділі, була розроблена структурна схема під конкретне технічне завдання, на основі якої був здійснений вибір елементної бази SHS. Після чого розроблено програмне забезпечення для 3-х локальних центрів «збирачів інформації» та головного хабу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Apple. URL: https://www.apple.com/home-app/ (дата звернення 15.11.2023).
2. Офіційний сайт Samsung. URL: https://www.samsung.com/ua/apps/smartthings/ (дата звернення 16.11.2023).
3. Соціальний захист населення України, 2022.
4. Smart House система для людей з обмеженими можливостями. Любченко Діана 6/12/2022 Sumy State University (Кафедра електроніки і комп'ютерної техніки).
5. Luke H.D. The origins of the sampling theorem. IEEE Comunication Magazine, 1999, no. 37(4), pp. 106-108.
6. Офіційний сайт digital-лабораторія інформаційних технологій URL: https://kr-labs.com.ua/blog/model-osi/ (дата звернення 15.11.2023).
7. Сайт з науково-популярними статтями URL:https://d0znpp.medium.com/coap-protocol-definition-architecture-752729043bb9 (дата звернення 15.11.2023).
8. RFC 7252 – Proposed Standard. 2014.
9. PATCH and FETCH CoAP // RFC 8132. 2017.
10. DTLS // RFC 6347 – Proposed Standard. 2012.
11. CoAP // RFC 7252 – Proposed Standard. 2014.
12. RFC 6298 – Proposed Standard. 2011.
13. Офіційний сайт IBM. URL: http://www.ibm.com/podcasts/software/ (дата звернення 18.11.2023).
14. OASIS Standard – MQTT Version 5.0.
15. ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v5.0.
16. Сайт з науково-популярними статтями URL: https://medium.com/@maydin/how-to-develop-iot-applications-for-android-using-mqtt-part-1-8018ec97d02c (дата звернення 18.11.2023).

17. Сторінка бібліотеки MQTT для мови програмування wiring на гітхабі. URL https://github.com/mqtt/mqtt.github.io/wiki/ (дата звернення 18.11.2023).
18. Сторінка продукту websphere-mq від компанії IBM URL: http://www-03.ibm.com/software/products/ru/websphere-mq (дата звернення 18.11.2023).
19. Сторінка MQTT брокеру URL: https://mosquitto.org (дата звернення 18.11.2023).
20. Сторінка хмарного сервісу, який надає послуги mqtt брокера URL: http://www.eurotech.com/en/products/software+services/everyware+device; (дата звернення 18.11.2023).
21. Сторінка MQTT брокеру URL: http://emqtt.io (дата звернення 18.11.2023).
22. Сторінка хмарного сервісу, який надає послуги mqtt брокера URL: http://www.hivemq.com (дата звернення 18.11.2023).
23. Научно-популярний ресурс, по mqtt протоколу URL: https://www.emqx.io/blog/introduction-to-mqtt-qos (дата звернення 19.11.2023).
24. Datasheet SHT3xA-DIS 2019;
25. Datasheet MQ-4;
26. Datasheet ESP-12E, 2015;
27. Datasheet ESP8266EX, 2020;
28. Сторінка виробника одноплатного комп'ютера Raspberry Pi URL: https://www.raspberrypi.com/ (дата звернення 19.11.2023).
29. Datasheet RASPBERRY P i 3B+ COMPUTE MODULE 3.
30. Сторінка завантаження програмного забезпечення HomeAssitant URL: https://www.home-assistant.io/installation/ (дата звернення 20.11.2023).
31. Сторінка бібліотеки zigbee для імплементації протоколу zigbee в HOAS URL: https://github.com/zigbee2mqtt/hassio-zigbee2mqtt (дата звернення 18.11.2023).

ДОДАТОК А

```
#include <Wire.h>
#include <Adafruit_SHT31.h>
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

const char *ssid = "SSID мережі клієнта";
const char *password = "Пароль мережі клієнта";
const char *mqtt_server = "IP_адрес_брокера";
const int mqtt_port = 1883;
const char *mqtt_user = "Користувач";
const char *mqtt_password = "Пароль";
const char *device_name = "Control_Room_1";
const int windowSensorPin = 16;
const int leakSensorPin = 0;
const int gasSensorPin = 2;
const int relayLightPin = 14;
const int relayGasPin = 12;

unsigned long previousMillisWindow = 0;
unsigned long previousMillisLeak = 0;
unsigned long previousMillisSHT30 = 0;
unsigned long previousMillisGas = 0;

const long interval = 1000;
const long sensor_interval = 262500;

bool windowStatus = false;
bool leakStatus = false;
float temperature = 0.0;
float humidity = 0.0;
bool gasStatus = false;
bool lightStatus = false;

WiFiClient espClient;
PubSubClient client(espClient);

Adafruit_SHT31 sht;

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

void reconnect() {
  while (!client.connected()) {
    if (client.connect(device_name, mqtt_user, mqtt_password)) {
      client.subscribe("home/assistant/room1/Light");
      client.subscribe("home/assistant/room2/Gas");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println("try again in 5 seconds");
      delay(5000);
    }
  }
}
```

```
void setup() {
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);

  sht.begin(0x44);
  pinMode(windowSensorPin, INPUT);
  pinMode(leakSensorPin, INPUT);
  pinMode(gasSensorPin, INPUT);
  pinMode(relayLightPin, OUTPUT);
  pinMode(relayGasPin, OUTPUT);
}

void checkWindowStatus() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisWindow >= interval) {
    previousMillisWindow = currentMillis;

    bool currentWindowStatus = digitalRead(windowSensorPin);
    if (currentWindowStatus != windowStatus) {
      windowStatus = currentWindowStatus;
      client.publish("home/assistant/room1/windows", windowStatus ?
"true" : "false");
    }
  }
}

void checkLeakStatus() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisLeak >= interval) {
    previousMillisLeak = currentMillis;

    bool currentLeakStatus = digitalRead(leakSensorPin);
    if (currentLeakStatus != leakStatus) {
      leakStatus = currentLeakStatus;
      client.publish("home/assistant/room1/leak", leakStatus ? "true" :
"false");
    }
  }
}

void checkSHT30() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisSHT30 >= sensor_interval) {
    previousMillisSHT30 = currentMillis;
    float currentTemperature = sht.readTemperature();
    float currentHumidity = sht.readHumidity();
    if (currentTemperature != temperature || currentHumidity != humidity)
{
      temperature = currentTemperature;
      humidity = currentHumidity;
      client.publish("home/assistant/room1/Temp", temperature);
      client.publish("home/assistant/room1/Hum", humidity);
    }
  }
}
```

```
void checkGasSensor() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisGas >= interval) {
    previousMillisGas = currentMillis;
    bool currentGasStatus = digitalRead(gasSensorPin);
    if (currentGasStatus != gasStatus) {
      gasStatus = currentGasStatus;
      client.publish("home/assistant/room2/Gas", gasStatus ? "true" :
"false");
    }
  }
}

void controlLightRelay() {
  if (client.available()) {
    String lightCommand = client.readStringUntil('\n');
    lightStatus = (lightCommand == "true");
    digitalWrite(relayLightPin, lightStatus);
  }
}

void controlGasRelay() {
  if (client.available()) {
    String gasCommand = client.readStringUntil('\n');
    bool gasRelayStatus = (gasCommand == "true");
    digitalWrite(relayGasPin, gasRelayStatus);
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  checkWindowStatus();
  checkLeakStatus();
  checkSHT30();
  checkGasSensor();
  controlLightRelay();
  controlGasRelay();
}
```

ДОДАТОК Б

```
#include <Wire.h>
#include <Adafruit_SHT31.h>
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

const char *ssid = "SSID мережі клієнта";
const char *password = "Пароль мережі клієнта";
const char *mqtt_server = "IP_адрес_брокера";
const int mqtt_port = 1883;
const char *mqtt_user = "Користувач";
const char *mqtt_password = "Пароль";
const char *device_name = "Control_Room_2";
const int windowSensorPin = 16;
const int relayLightPin = 0;
const int gasSensorPin = 2;

unsigned long previousMillisWindow = 0;
unsigned long previousMillisSHT30 = 0;
unsigned long previousMillisGas = 0;

const long interval = 1000;
const long sensor_interval = 262500;

bool windowStatus = false;
float temperature = 0.0;
float humidity = 0.0;
bool gasStatus = false;

WiFiClient espClient;
PubSubClient client(espClient);

Adafruit_SHT31 sht;

void setup_wifi() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

void reconnect() {
  while (!client.connected()) {
    if (client.connect(device_name, mqtt_user, mqtt_password)) {
      client.subscribe("home/assistant/room2/Light");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

```
void setup() {
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);

  sht.begin(0x44);
  pinMode(windowSensorPin, INPUT);
  pinMode(relayLightPin, OUTPUT);
}

void checkWindowStatus() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisWindow >= interval) {
    previousMillisWindow = currentMillis;

    bool currentWindowStatus = digitalRead(windowSensorPin);
    if (currentWindowStatus != windowStatus) {
      windowStatus = currentWindowStatus;
      client.publish("home/assistant/room2/windows", windowStatus ?
"open" : "closed");
    }
  }
}

void checkSHT30() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisSHT30 >= sensor_interval) {
    previousMillisSHT30 = currentMillis;

    float currentTemperature = sht.readTemperature();
    float currentHumidity = sht.readHumidity();

    if (currentTemperature != temperature || currentHumidity != humidity)
{
      temperature = currentTemperature;
      humidity = currentHumidity;

      String payload = String(temperature) + "," + String(humidity);
      client.publish("home/assistant/room2/TempHum", payload.c_str());
    }
  }
}

void checkGasSensor() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisGas >= interval) {
    previousMillisGas = currentMillis;
  }
}

void controlLightRelay() {
  if (client.available()) {
    String lightCommand = client.readStringUntil('\n');
```

```
    bool lightStatus = (lightCommand == "true");
    digitalWrite(relayLightPin, lightStatus);
  }
}

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

  checkWindowStatus();
  checkSHT30();
  checkGasSensor();
  controlLightRelay();
}
```

ДОДАТОК В

```
#include <Wire.h>
#include <Adafruit_SHT30.h>
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

const char *ssid = "SSID мережі клієнта";
const char *password = "Пароль мережі клієнта";
const char *mqtt_server = "IP_адрес_брокера";
const int mqtt_port = 1883;
const char *mqtt_user = "Користувач";
const char *mqtt_password = "Пароль";
const char *device_name = "Control_Room_3";
const int leakSensorPin = 16;
const int relayLightPin = 0;
const int relayWaterPin = 2;

unsigned long previousMillisSHT30 = 0;
unsigned long previousMillisLeak = 0;

const long interval = 1000;
const long sensor_interval = 262500;

float temperature = 0.0;
float humidity = 0.0;

WiFiClient espClient;
PubSubClient client(espClient);

Adafruit_SHT30 sht;


void setup_wifi() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);


void reconnect() {
  while (!client.connected()) {
    if (client.connect(device_name, mqtt_user, mqtt_password)) {
      client.subscribe("home/assistant/room3/Light");
      client.subscribe("home/assistant/room1/leak");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  setup_wifi();
```

```
  client.setServer(mqtt_server, mqtt_port);
  sht.begin();
  pinMode(leakSensorPin, INPUT);
  pinMode(relayLightPin, OUTPUT);
  pinMode(relayWaterPin, OUTPUT);
}

void checkSHT30() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisSHT30 >= sensor_interval) {
    previousMillisSHT30 = currentMillis;
    float currentTemperature = sht.readTemperature();
    float currentHumidity = sht.readHumidity();
    if (currentTemperature != temperature || currentHumidity != humidity)
{
      temperature = currentTemperature;
      humidity = currentHumidity;
      String payload = String(temperature) + "," + String(humidity);
      client.publish("home/assistant/room3/TempHum", payload.c_str());
    }
  }
}

void checkLeakSensor() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillisLeak >= interval) {
    previousMillisLeak = currentMillis;
    bool leakStatus = digitalRead(leakSensorPin);
    client.publish("home/assistant/room1/leak", leakStatus ? "true" :
"false");
    digitalWrite(relayWaterPin, !leakStatus); //   }
}

void controlLightRelay() {
  if (client.available()) {
    String lightCommand = client.readStringUntil('\n');
    bool lightStatus = (lightCommand == "true");
    digitalWrite(relayLightPin, lightStatus);
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  checkSHT30();
  checkLeakSensor();
  controlLightRelay();
}
```

ДОДАТОК Г

```
square: true
type: grid
cards:
  - show_name: true
    show_icon: true
    type: button
    tap_action:
      action: call-service
      service: mqtt.publish
      target: {}
      data:
        qos: 0
        retain: false
        topic: home/assistant/room1/light
        payload: 'true'
    hold_action:
      action: call-service
      service: mqtt.publish
      target: {}
      data:
        qos: 0
        retain: false
        topic: home/assistant/room1/light
        payload: 'false'
    name: Кухня
    icon: mdi:ceiling-light
    show_state: true
  - show_name: true
    show_icon: true
    type: button
    tap_action:
      action: call-service
      service: mqtt.publish
      target: {}
      data:
        topic: home/assistant/room2/light
        payload: 'true'
    hold_action:
      action: call-service
      service: mqtt.publish
      target: {}
      data:
        qos: 0
        topic: home/assistant/room2/light
        payload: 'false'
    name: Кімната
    icon: mdi:ceiling-light
  - show_name: true
    show_icon: true
```

```
    type: button
    tap_action:
      action: call-service
      service: mqtt.publish
      target: {}
      data:
        topic: home/assistant/room3/light
        payload: 'true'
    name: Ванна
    show_state: false
    hold_action:
      action: call-service
      service: mqtt.publish
      target: {}
      data:
        topic: home/assistant/room3/light
        payload: 'false'
    icon: mdi:bulkhead-light
columns: 3
```