ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технлогій

(повне найменування факультету)

Кафедра комп'ютерних систем та технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти « магістр»

«Дослідження масштабованих систем «розумний будинок»»

(тема кваліфікаційної роботи українською мовою)

«Research on scalable "smart home" systems»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 123 комп'ютерна інженерія
(код, назва спеціальності)

Освітня програма _____
(назва)

Коваленко Іван Олександрович
(прізвище, ім'я, по-батькові здобувача)

Керівник д.т.н., доц. Михайленко В.С. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент д.т.н, проф. Гунченко Ю.О.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ _5_ від _25_._12_. 20_23_ р.

Завідувач(ка) кафедри

_____ Гунченко Ю.О.
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/_____/_____
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ _____
(підпис) (прізвище, ім'я)

Одеса 2023

АНОТАЦІЯ

Кваліфікаційна робота присвячена темі дослідження масштабованих систем «розумного будинку». Метою роботи є поліпшення якості роботи масштабованої системи «розумний будинок» шляхом розробки методики оцінки якості протоколів передачі даних як комплексне програмне забезпечення системи що охоплює програмну та апаратну частину системи «розумний будинок». Об'єктом є дослідження протоколів передачі даних як частини комплексної системи «розумний будинок». Предметом дослідження є функціонування протоколів передачі даних у системі «розумний будинок».

У роботі розглядається розвиток моделей оцінки якості програмного забезпечення, що показує розвиток цієї сфери розробки програмного забезпечення. Описується комплексний багаторівневий підхід до проведення оцінки якості програмно-апаратних систем «розумного будинку». На основі отриманого підходу проводиться аналіз двох обраних протоколів передачі даних у системах «розумного будинку», а саме HTTP та MQTT. Проводяться експерименти для визначення кращого за швидкодією, використанням трафіку, затримці та енергоефективності.

Результатом проведеного дослідження стали такі результати:

- Протокол MQTT є на 22 % енергоефективнішим та на 15 % швидший за протокол HTTP у короткостроковій перспективі.
- MQTT використовує у 54.5 рази менше даних та є у 20 разів швидший при передачі даних які залежні від часу та мають буди якнайшвидше доставлені.
- Затримка протоколу MQTT у 24 рази менша за затримку у протоколі HTTP

Ключові слова: «розумний будинок», протоколи передачі даних, якість програмного забезпечення, енергоефективність, програмне забезпечення.

ABSTRACT

The qualification work is devoted to the research topic of scalable "smart house" systems. The purpose of the work is to improve the quality of the scaled "smart house" system by developing a methodology for evaluating the quality of data transmission protocols as a complex system software covering the software and hardware parts of the "smart house" system. The object is the study of data transfer protocols as part of a complex "smart home" system. The subject of the study is the functioning of data transfer protocols in the "smart house" system.

The work examines the development of software quality assessment models, which shows the development of this area of software development. A comprehensive multi-level approach to quality assessment of software and hardware systems of the "smart house" is described. Based on the obtained approach, the analysis of two selected data transfer protocols in "smart home" systems, namely HTTP and MQTT, is carried out. Experiments are conducted to determine the best in terms of speed, traffic utilization, latency, and energy efficiency.

The results of the research were as follows:

- MQTT is 22% more energy efficient and 15% faster than HTTP in the short term;
- MQTT uses 54.5 times less data and is 20 times faster when transferring data that is time dependent and must be delivered as soon as possible;
- the delay of the MQTT protocol is 24 times lower than the delay of the HTTP protocol

Key words: "smart house", data transfer protocols, software quality, energy efficiency, software.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

мАг – міліампер на годину

GUI – Graphical User Interface

IEEE – Institute of Electrical and Electronics Engineers

IoT – Internet of Things

IETF – Internet Engineering Task Force

LWT – Last Will and Testament

QoS – Quality of Service

SDLC – Software Development Lifecycle

ВСТУП

Комп’ютерні системи стали невід’ємною частиною життєзабезпечення сучасної системи. Сучасною тенденцією комфортного проживання людини є розгортання систем «розумного» будинку. Це дозволяє автоматизувати рутинні процеси по підтримці комфортного рівня параметрів житла таких як, температура, вологість, яскравість освітлення, рівень вуглекислого газу та навіть чистоту повітря. Це дуже важливі параметри життєзабезпечення людини. Тому якість систем має велику важливість для користувача. Наслідки від використання неякісного програмного забезпечення можуть бути катастрофічними починаючи від аварій, матеріальних втрат і небезпеки для людини і закінчуючи втратою іміджу компанії розробки.

Активна розробка та впровадження автоматизованих систем у житло людини проходила поступово, починаючи з 60-х років. Вважається що першим винаходом був прилад для регулювання яскравості світла, що дало перший поштовх до автоматизації будинку. Якісна система дозволяє достатньо добре економити ресурси на забезпечення комфорту у приміщеннях, додатково якість системи зумовлює популярність системи при виборі користувачами на етапі її купування та встановлення. Тому розробникам необхідно все більше уваги приділяти якості систем та їх компонентів. Багато виробників систем стараються зробити систему більш дешевою за рахунок закупки дешевих компонентів апаратного забезпечення, таких як датчики, пристрої обробки даних, контролери. Безумовно, такий підхід до побудови систем є гарною альтернативою дорогим оригінальним контролерам. Але якість комплектуючих іноді зі зменшенням вартості падає, що додатково є фактором для розробки неякісних систем. Додамо до неякісних компонентів дешеве та неякісне програмування і в підсумку маємо неякісну систему. Але за період розвитку програмної інженерії, як

самостійного напрямку в інженерній практиці, а потім і формування її як системної науки, одним з ключових було питання програмного забезпечення.

Під якістю системи розуміється здатність системи задовольняти встановленим або передбачуваним потребам. Тому важливим є забезпечення спільного розуміння між виробником і користувачами. Інженери повинні чітко розуміти зміст, вкладений в концепцію якості, характеристики і значення якості. Сформулювати чіткі вимоги до якості програмного та апаратного продукту, а потім їх правильно оцінити – одне з пріоритетних завдань забезпечення якості програмно-апаратного комплексу. Тому як системи «розумного» будинку являються програмно-апаратними, існує потреба у моделі та підходу до вимірювання якості цієї системи. Є необхідність у комплексній моделі та підходу до проведення вимірювання якості такої великої масштабованої системи, що включає в себе програмне та апаратне забезпечення.

Однією з важливих частин масштабованих систем «розумного» будинку є цілісність даних які передаються у системі. Дуже багато даних передається у таких системах, і якщо дані будуть якимось чином втрачені або передані у неправильному вигляді, це може спричинити помилки у керуванні системою, та призвести до аварії або травм користувача. Тому якість системи важлива частина процесу розробки.

Об’єктом є дослідження протоколів передачі даних як частини комплексної системи «розумний будинок».

Предметом дослідження є функціонування протоколів передачі даних у системі «розумний будинок»

Метою роботи є поліпшення якості роботи масштабованої системи «розумний будинок» шляхом розробки методики оцінки якості протоколів передачі даних як комплексне програмне забезпечення системи що охоплює програмну та апаратну частину системи «розумний будинок».

Для досягнення даної мети сформульовані наступні задачі:

- визначення сучасних моделей та підходів до оцінки якості програмного забезпечення на основі існуючих стандартів та запропонованих;
- аналіз існуючих протоколів передачі даних у системі «розумний будинок»;
- розробка комплексного підходу до вимірювання якості систем «розумний будинок»;
- приклад використання запропонованої моделі та методики оцінювання якості «розумний будинок» а саме реалізація компоненту моделі тестування системи передачі даних;
- приклад використання обраного протоколу передачі даних для системи «розумний будинок».

1 ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Поняття якості програмного забезпечення

Що представляється під поняттям якість і навіщо потрібне таке глибоке визначення?

Якість програмного забезпечення як поняття сформувалося внаслідок еволюції програмної індустрії та росту усвідомлення важливості забезпечення високої якості в розробці програмного забезпечення. На початках історії програмування у 1940-1950-х роках основною метою було створення програм, які працюють на визначених вхідних даних[1]. На той час важко було вимірювати якість програм оскільки багато із них використовувались у наукових або військових дослідженнях. Тому не існує багато інформації про використання тих чи інших методів та взагалі тестування програмного забезпечення.

З появою інтерактивних систем у 1960-1970-х роках стало важливим забезпечення користувачів зручним та ефективним інтерфейсом. Це привело до появи концепцій, таких як «зручність використання» та «ефективність». Саме у 1968 році на конференції НАТО відбулося видання «Криза програмування»[2] де висловлювалися обурення від поганої продуктивності та якості програм. Основними причинами кризи становили:

- зростання складності програм: з появою більших та складніших проєктів з’явилася потреба в управлінні розробками програм, оскільки традиційні методи не відповідали великим системам;
- великі витрати на розробку: проєкти значного розміру часто перевищували встановлені бюджети та терміни виконання, що призводило до фінансових проблем;

- низька продуктивність: Розробка програм вимагала значних зусиль і займала багато часу, що зменшувало ефективність інформаційних технологій і виробничих умовах;
- потреба в нових методологіях: Традиційні методи розробки не забезпечували високої якості та гнучкості, яка була потрібна для великих та складних проєктів.

На цьому етапі з’явилися різні підходи та ідеї, спрямовані на вирішення цих проблем. Ключовими результатами конференції були заснування області програмної інженерії як самостійної галузі та розробка нових методологій, які включали в себе моделі життєвого циклу, управління конфігурацією та інші інноваційні підходи до розробки програмного забезпечення.

Криза програмування стала каталізатором для змін у підходах до розробки програмного забезпечення, стимулювала дослідження та введення нових методів, які сприяли покращенню якості, продуктивності та ефективності у галузі програмної інженерії. У 1979 році Філ Кросбі визначив термін якість як «Відповідність вимогам користувача» та припускав що вимоги повинні бути настільки чітко визначені щоб вони не могли бути незрозумілі або інтерпретовані якось некоректно. Також Уотс Хемпфрі описав якість як «досягнення відмінного рівня придатності до використання» який брав до уваги вимоги та очікування кінцевих користувачів продукту, які очікують що продукт буде зручним для їх потреб. У свою чергу компанія IBM описала та ввела в обіг фразу «якість, керована ринком». У критерії Белріджа що визначає організаційну якість використовує фразу – «якість, що керується споживачем», який розглядає задоволення споживача як головний критерій якості[3]. Останні дві фрази достатньо схожі. Але в основному використовується визначення якості яке описане у системі менеджменту якості ISO 9001 – «ступінь відповідності властивих характеристик вимогам»[4].

Насправді поняття якість не настільки очевидне і просте як може здатися на перший погляд. Існує безліч інтерпретацій якості в залежності від інженерного продукті. І саме на початку проекту на необхідно визначити, обговорити та описати вимоги до програмного продукту. Якість як характеристика можуть бути відсутніми або задаватися «жорстко», все це є результатом обговорення та компромісів, що перегукується з терміном «прийнятна якість», як менш жорстокої точки зору на забезпечення якості, як досягнення досконалості.

У сучасному світі, у часи міжнародних стандартів існує декілька визначень поняття якості.

Найпоширенішими є:

- визначення ISO: міра відповідності продукту визначеним вимогам та його здатність задовольняти потреби користувачів та зацікавлених сторін[5];
- визначення IEEE: ступінь в якому продукт має необхідну комбінацію властивостей для задоволення потреб користувачів[6].
-

1.2 Сучасні стандарти щодо якості програмного забезпечення

На сьогоднішній день найсучаснішим стандартом є серія стандартів ISO 25000 SQuaRE (Systems and software Quality Requirements and Evaluation)[5] – уніфікована серія стандартів яка забезпечує специфікацію вимог до якості програмного продукту, дозволяють зменшити невизначеність при сумісній роботі організацій щодо розробки, впровадження та супроводу програмного забезпечення. Як приклад між замовниками, розробниками та незалежними оцінювачами.

На рисунку 1.1 показана структура серії стандартів SQuaRE.

Рисунок 1.1 – Структура серії стандартів ISO 25000 SQuaRE

Серія стандартів ISO/IEC 25000 включає кілька документів, які визначають модель якості програмного забезпечення та надають вимоги та рекомендації щодо оцінки цієї якості. У таблиці 1.1 наведений опис кожного стандарту з цієї серії.

Таблиця 1.1 – опис сімейств стандартів SQuaRE

Сімейство	Характеристика
«Керування якістю» 25000n	Вступний розділ у якому визначені основні поняття, терміни, визначають моделі які будуть використовуватися у подальших розділах стандартів серії SQuaRE. Представлені методичні матеріали та вимоги щодо функцій підтримки. Матеріал стосується управлінням вимогами до продукту, його специфікаціями та оцінкою.
«Моделі якості» 2501n	Встановлює модель якості програмного забезпечення. Він містить основні характеристики якості, що розглядаються при розробці, оцінці та виборі програмних продуктів, стосується таких характеристик як функціональність, надійність, зручність у використанні, безпека та інші.

Продовження таблиці 1.1

Сімейство	Характеристика
«Вимірювання якості» 2502n	Розділ включає в себе математично визначені показники якості та практичне керівництво щодо їх використання. Кожний з стандартів цього сімейства зосереджується на конкретному аспекті якості, такому як продуктивність, безпека, зручність у використанні, надаючи конкретні критеріх та методи для їх оцінки. Ці стандарти допомагають оцінити та забезпечити відповідність програмного забезпечення вимогам користувачів у плані різноманітних атрибутів якості.
«Вимоги до якості» 2503n	Сімейство включає стандарти у яких визначаються вимоги до якості програмного продукту на основі існуючих моделей якості, що допомагає визначити вимоги до якості продукту ще на початкових стадіях розробки.
«Оцінка якості» 2504n	Формулюють вимоги та методичні матеріали для оцінки програмного продукту, використовується як замовниками так і розробниками продукту. Представлена підтримка та рекомендації для документування показників вимірювання якості.
«Розділ розширення» 25050 - 25099	Визначають вимоги до якості готового програмного продукту та вимоги до формату для звітів з вимірювання якості комерційного продукту.

1.3 Історичний розвиток моделей якості програмних продуктів

Модель МакКола[1]

Першою моделюю якості програмного забезпечення яка стала широко відомою стала модель яка була запропонована у 1977 році Макколом та іншими авторами. У цій моделі характеристики розділялись на три групи:

- фактори, які описують програмне забезпечення з позицій користувача та заданих вимог;
- критерії, які описують програмне забезпечення з позицій розробника і задаються як цілі;
- метрики, які використовуються для кількісного опису та вимірювання якості.

Фактори якості, яких було 11, згруповані у три групи відповідно до роду роботи людей з програмним забезпечення. Ця структура зображується у вигляді трикутника Маккола. Критерії якості – числові рівні факторів, які поставлені як цілі при розробці. Оцінити або виміряти фактори якості достатньо важко, тому Маккол використав метрики якості, які дозволили легше оцінити та виміряти фактори якості. Оцінки можуть набувати значень від 0 до 10.

Метрики якості:

- зручність перевірки на відповідність стандартам;
- точність управління та обчислень;
- ступінь стандартності інтерфейсів;
- функціональна повнота;
- однорідність використовуваних правил проектування документацій;
- ступінь стандартності форматів даних;
- стійкість до помилок;

- ефективність роботи;
- розширюваність;
- широта сфери потенційного використання;
- апаратна незалежність;
- повнота протоколювання помилок та інших подій;
- модульність;
- зручність роботи;
- захищеність;
- самодокументованість;
- простота роботи;
- незалежність від програмної платформи;
- можливість порівняння проекту з вимогами;
- зручність налаштування.

Кожна метрика впливає відразу на декілька факторів якості. Числове вираження фактора являє собою лінійну комбінацію значень впливаючих на нього метрик. Коефіцієнти вираження визначаються по різному, в залежності від організацій, команд розробки, видів програмного забезпечення. На рисунку 1.2 представлений Трикутник Маккола.

Рисунок 1.2 – Трикутник Маккола

Модель Боема[7]

Друга з основоположних моделей якості являється модель якості Боема яка була запропонована у 1978 році. Модель Боема має недоліки сучасних моделей, які автоматично та якісно оцінюють якість програмного забезпечення. В цілому модель Боема намагається якісно визначити якість програмного забезпечення заданим набором показників та метрик. Модель якості Боема представляє характеристики програмного забезпечення у більшому масштабі, ніж модель Маккола. Модель Боема схожа на модель якості Маккола тим що вона також являється ієрархічною моделюю якості, структурованою навколо високорівневих, проміжних та примітивних характеристик, кожна з яких вносить свій вклад у рівень якості програмного забезпечення. У цій моделі визначено 19 атрибутів, що включають всі 11 фактори якості за Макколом. Проміжні атрибути розділяються на примітивні, які у свою чергу можуть бути оцінені на основі метрик.

На додаток до факторів Маккола атрибути якості за Боемом включаються такі:

– ясність;

- зручність внесення змін;
- документованість;
- здатність до відновлення функцій;
- зрозумілість;
- адекватність;
- функціональність;
- універсальність;
- економічна ефективність.

Модель FURPS/FURPS+[8]

Акронім FURPS, визначає наступні катерогії вимог до якості програмного забезпечення:

- functionality (Функціональність) – особливості, можливості, безпека;
- usability (Практичність) – людський фактор, ергономічність, документація користувача;
- reliability (Надійність) – частота відмов, відновлення інформації, прогнозованість;
- performance (Продуктивність) – час відгуку, пропускна спроможність, точність, доступність, використання ресурсів;
- supportability (Експлуатаційна придатність) – тестованість, розширюваність, адаптованість, супроводжуваність, сумісність, конфігурованість, обслуговування, вимоги до встановлення, локалізованість.

Символ «+» розширяє FURPS модель, додаючи до неї такі категорії:

- обмеження проекту – обмеження по ресурсам, вимоги до мов та засобів розробки, вимоги до апаратного забезпечення;

- інтерфейс – обмеження що накладаються на взаємодію з зовнішніми системами;
- вимоги до виконання;
- фізичні вимоги;
- вимоги до ліцензування.

FURPS як модель якості, запропонована Грейді та Hewlett Packard побудована схожим чином з моделями Маккола та Боема, але на відміну від них складається з двох шарів, перший визначає характеристики, а другий зв’язані з ними атрибути. Основною концепцією, яка лежить в основі FURPS моделі якості, є декомпозиція характеристик програмного забезпечення на дві категорії вимог, а саме, функціональні (F) та нефункціональні (URPS) вимоги. Ці виділені категорії можуть бути використані як у якості вимог до програмного продукту, так і у оцінці якості програмного продукту. Сьогодні модель FURPS+ широко використовується у розробці програмного забезпечення та при ідентифікації вимог до розроблюваємої системи доцільно використовувати FURPS+ моделі у якості універсального контрольного переліку характеристик програмного забезпечення.

Модель якості програмного забезпечення QMOOD[1]

Джагдіш Баназія та Карл Девіс запропонували ієрархічну модель якості для об’єктно-орієнтованого проектування. Яка розширяє методологію моделі якості Дромі та включає у себе чотири рівня:

- визначення показників якості проекту: набір атрибутів якості проекту, які використовуються у QMOOD для опису характеристик об’єктно-орієнтованих систем та включають: функціональність, ефективність, зрозумілість, розширюваність, можливість багаторазового використання та гнучкість;
- визначення об’єктно-орієнтованих властивостей продукту: властивості проекту можуть бути визначені у процесі дослідження

внутрішніх та зовнішніх структур, функціональності компонент проекту, атрибутів, методів та класів. Структурними та об’єктно-орієнтованими властивостями проекту є: розмір проекту, ієрархічна структура, інкапсуляція, зв’язаність, зміст проекту, успадкування, поліморфізм, обмін інформацією, складність;

- визначення об’єктно-орієнтованих метрик проекту: різноманітні об’єктно-орієнтовані метрики проекту;
- визначення об’єктно-орієнтованих властивостей проекту: Компоненти проекту біли визначені для визначення архітектури об’єктно-орієнтованого проекту. Ця модель визначає парадигму, а також вводить ряд нових об’єктно-орієнтованих метрик.

Модель якості програмного забезпечення SQuaRE[5]

В додаток до стандарту ISO 9126 був представлений стандарт ISO/IEC 14598, який регламентує способи оцінки характеристик. В цілому вони утворюють модель якості, яка стане відомою як SQuaRE (Software Quality Requirements and Evaluation)[5].

У цьому стандарті визначений загальний підхід до моделювання якості програмного забезпечення. Спочатку ідентифікується невеликий набір атрибутів якості самого вищого рівня абстракції, потім у напрямку зверху вниз розвиваються обрані атрибути на набори підлеглих атрибутів.

Модель виділяє наступні шість основних характеристик якості:

- функціональність. Функціональні вимоги традиційно складають основний предмет специфікації, моделювання, реалізації та атестації програмного забезпечення. Специфікації формулюються у вигляді тверджень у імперативній формі, які описують поведінку системи. Використання формальних тверджень дозволяє довести відхилення фактичної поведінки системи від необхідної практично до нуля. Досягнення цього досягається шляхом вираження

функціональних вимог у вигляді пропозицій відповідних формальних обчислень, так що верифікація сводиться до нуля;

- надійність. Показники надійності характеризують поведінку системи при виході за межі штатних значень параметрів функціонування по причині збою в оточені або в самій системі. При оцінці атрибутів надійності застосовуються методи теорії імовірності та математичної статистики. Вимоги до надійності особливо важливі при розробці критичних систем забезпечення безпеки життєдіяльності. Хоча використання формальних методів сприяє зниженню кількості внутрішніх помилок, забезпечення надійності в цілому потребує спеціальних підходів, які враховують специфіку різних типів систем;
- зручність. Відповідність системи вимогам до зручності надзвичайно важко піддаєтсья оцінці. Запропоновані підходи включають заміри витрат нормативних одиниць праці, які витрачають користувачі на оволодіння системою, а також проведення та аналіз експертних оцінок, в тому числі із застосуванням методів нечіткої логіки;
- ефективність. Атрибути ефективності традиційно відносяться до числа важливих кількісних показників якості програмних систем. Їх значення підлягають фіксації в експлуатаційній документації до програмних та апаратних виробів. Є високо розвинутий інструментарій для вимірювання цих значень. Розроблені також методики, які дозволяють прогнозувати інтегральні значення показників ефективності системи виходячи із значень цих показників для компонентів самої системи та її оточення. Вибору формальних методів забезпечення ефективності варто приділяти особливу увагу. При цьому варто зазначити що хоча є тісний взаємозв’язок між продуктивністю та ресурсоємністю, підходи до

забезпечення кожної з цих вимог в загальному випадку мають різну природу;

– супроводжуваність. Вимоги до супроводжуваності направлені на мінімізацію зусиль із супроводжування та модернізації системи, які витрачаються експлуатаційним персоналом. Для оцінки зусиль із супроводу продукту використовуються методики прогнозування витрат на виконання типових процедур супроводження. Вартість супроводу довго живучих систем може суттєво перевищувати вартість розробки. Супровід суттєво спрощується коли розробка проводилась із використанням формальних методів, оскільки мається вичерпний комплект технологічної документації та перевірочних тестів;
– переносимість. Переносимість системи характеризує ступінь свободи у виборі компонентів системного оточення, необхідних для її функціонування. Оцінка переносимості ускладнюється принципіальною незавершеною, динамічністю списку компонентів оточення, що обумовлене швидким прогресом у сфері інформаційних технологій. Системи які розроблені з використанням формальних методів, відрізняються великим рівнем переносимості. Якщо така система не підтримує якусь цільову технологічну платформу, то створення «клону» потребує істотно менших витрат, ніж заміна самої системи або платформи.

2 КОМПЛЕКСНА МОДЕЛЬ ТЕСТУВАННЯ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ

IoT – це термін, який був зовсім незнайомий звичайним людям у минулі часи. Однак у цю еру розумних технологій і розумніших систем IoT став дуже популярним. IoT – це нова сфера, яка може відігравати життєво важливу роль майже в усіх галузях і дисциплінах, включаючи сільське господарство, сектор охорони здоров'я, домашню автоматизацію, авіацію та транспорт, оборону, військові програми тощо[9].

IoT – це мережа фізичних пристроїв, транспортних засобів (також їх називають «підключеними пристроями» та «розумними пристроями»), будівель та інших об'єктів із вбудованою електронікою, програмним забезпеченням, датчиками, приводами та мережевим підключенням, які дозволяють цим об'єктам збирати та обмін даними. IoT можна використовувати, щоб зробити об'єкти чи речі розумнішими шляхом дистанційного визначення або контролю за ними. IoT містить речі, які мають унікальні ідентифікатори та підключені до Інтернету. IoT описує систему, де предмети у фізичному світі та датчики всередині цих елементів або які підключені до них підключаються до Інтернету через бездротові та дротові підключення до Інтернету. Ці датчики використовують різні типи локальних з'єднань, такі як радіочастотна ідентифікація (RFID), зв'язок ближнього поля (NFC), бездротова локальна мережа (Wi-Fi), Bluetooth і ZigBee. Датчики також мають можливість підключатися за допомогою систем, такі як системи широкого доступу до мережі такі як Глобальна система мобільного зв'язку (GSM), послуги загального пакетного радіозв'язку (GPRS), послуги мобільного зв'язку третього і четвертого покоління (3G, LTE). Типова архітектура IoT проілюстрована на рис. 2.1. Це мережевий набір пристроїв і вбудованих компонентів програмного забезпечення. Вбудована система може мікроконтролерами або платами, які мають певні можливості обробки.

Наприклад, Arduino, Raspberry Pi, Intel Galileo тощо. Інтернет-протокол версії 6 (IPv6) використовується для IP-адресації, тому що він може адресувати до мільйонів пристроїв. Існує область застосування, де центральні елементи або датчики розгорнуті для моніторингу та контролю. Вони сприймають відповідну інформацію та передають її до певної кіберфізичної системи, яка виконує обчислення та координацію.

Підключені компоненти передають дані за допомогою бездротових технологій. На основі інструкцій або зворотного зв'язку приводи виконують необхідні дії, які мають бути виконані в сценарії.

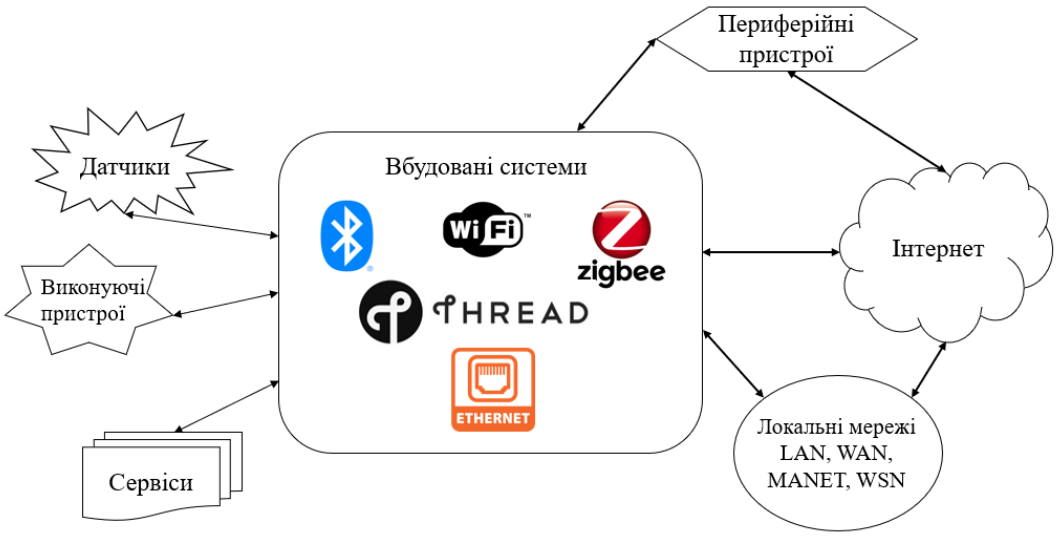

Рисунок 2.1 – Типова архітектура IoT

Типова архітектура IoT має наступні характеристики:

– підключення: усі пристрої, датчики та обладнання мають бути з'єднані один з одним через безпечну мережу;
– речі: речами в перспективі IoT можуть бути різноманітними пристрої, такими як транспондери з біочіпами на домашніх або сільськогосподарських тваринах, імплантати для моніторингу

серця, автомобілі з вбудованими датчиками. Речі можуть бути приводами, датчиками або навіть шлюзами. Ці пристрої або речі збирають необхідні дані за допомогою існуючих технологій, а потім автономно передають дані між іншими пристроями;

- дані: Дані є сполучним агентом в IoT, який ініціює дії та інтелект;
- зв’язок: кожен компонент або речі спілкуються один з одним для передачі даних або для ініціювання моніторингу та керування системою. Зв'язок зазвичай ініціюється за допомогою бездротових протоколів;
- інтелект: сенсори зазвичай роблять пристрої IoT інтелектуальними з деякими елементами обробки для аналізу даних і попередньої обробки;
- дія: це може бути або ручна дія, або дія, заснована на порівняннях явищ (наприклад, у рішеннях щодо зміни клімату) та автоматизація, зазвичай ініційована виконавчими механізмами;
- екосистема: середовище або домен IoT.

Тестування – це діяльність з контролю якості, яка передбачає виявлення дефектів і виправлення помилок. виправлення помилок[8]. Тестування є частиною процесу життєвого циклу розробки системи (SDLC) для валідації та перевірки роботи розробленого продукту або додатку. Тестування може виконуватися на різних етапах процесу розробки залежно від методології та інструментів, що використовуються, і зазвичай починається після етапу підтвердження вимог. Початкова фаза – на рівні пристрою, де основна увага приділяється тестуванню окремих компонентів, пристроїв і комунікаційних рівнів. Коли компоненти IoT пов'язані між собою, ми проводимо тестування, щоб знайти помилки взаємодії. Кінцева мета тестування системи – задовольнити зацікавлені сторони і забезпечити якість програми.

Тестування системи IoT також можна назвати процесом виконання валідації та верифікації системи або додатку на основі Інтернету речей, які відповідають бізнес-орієнтованим і технічно-орієнтованим функціональним вимогам, якими керувалися на етапі проектування та розробки. Валідація та верифікація – це процес забезпечення того, щоб програмне забезпечення або система відповідала вимогам, зазначеним у специфікації системних вимог (System Requirement Specification - SRS), і що вона виконує свої функції за призначенням. функціональність. Це можна розглядати як методологію забезпечення якості продукту. Терміни можна визначити наступним чином:

- верифікація - процес забезпечення того, що розроблений продукт створено належним чином;
- валідація - процес забезпечення того, чи розроблений продукт відповідає очікуванням.

Пропонується гібридна модель шляхом інтеграції моделей тестування різних апаратних засобів, програмного забезпечення та протоколів зв’язку.

2.1 Запропоновані рішення

Був проведений огляд літератури, щоб визначити, які методології тестування використовуються в системах на основі Інтернету речей. У цій галузі виконано дуже мало робіт, хоча IoT є новою сферою інформаційних технологій.

Маріниссен та ін. обговорили існуючі тенденції Інтернету речей і проблеми, пов’язані з тестуванням додатків на основі Інтернету речей. Вони розглядали завдання тестування з різних точок зору[10]. Точка зору проектування складається з тестових завдань щодо датчиків, підключення по ZigBee або Wi-Fi та керування живленням. Вони також обговорили

необхідність тестування виробництва IoT, а також необхідність тестування безпеки в додатках на основі IoT. Їх висновки стверджують, що розробка якісного IoT-продукту з мінімальними витратами є основним завданням для розробників IoT.

Рітз та ін. запропонували новий підхід для тестування додатків на основі IoT, побудованих на методології вставки коду[11]. Він отриманий з використанням семантичного опису сервісу на основі IoT. Запропонована ними структура архітектури складається з механізму проектування тестів і механізму виконання тестів, а також із середовища ізольованого програмного середовища, яке має можливість імітувати мережеві аспекти системи. Введення коду здійснюється вручну. Однак цій методології не вистачає перевірки апаратного забезпечення та фізичних елементів системи на основі IoT.

Ліл та ін. запропонували тестову модель IoT для тестування технологій на основі RFID та IoT, що використовуються в бразильських інтелектуальних системах автомобільного транспорту[12]. Вони розробили тестовий стенд і структуру для оцінки продуктивності та функціональних можливостей систем на основі RFID, які використовуються в інтелектуальному автомобільному транспорті, але ця методологія розглядає лише тестові випадки в їхній області.

Розенкранц та ін. запропонували структуру тестування IoT, яка підтримує методи безперервної інтеграції апаратного та програмного забезпечення[13]. Запропонована модель використовує тестові кластери, щоб кожен міг розгорнути тестові платформи в цій системі. Ця стратегія розподіленого тестування забезпечує спільний доступ до окремих систем на базі IoT або до повністю розширених тестових стендів IoT. Він створює тестові приклади для різних операційних платформ. Вони стверджують, що їх архітектуру можна використовувати для тестування сумісності мережі за

допомогою переадресації зв’язку від однієї системи IoT, що тестується, до іншої з прозорістю розташування.

З цього аналізу можна зробити висновок, що не існує загальної моделі тестування IoT для тестування додатків на основі IoT. Більшість існуючих систем не тестують інтегровану систему IoT[14]. У цій роботі інтегруються найкращі функції попередніх робіт та імпортуються деякі методології тестування IoT, які використовуються різними працівниками галузі, щоб забезпечити продуктивність і ефективність систем IoT.

2.2 Запропонована модель

IoT відрізняється від інших веб-додатків і додатків вбудованої системи наступними характеристиками:

- інтегрована система з обладнанням, програмним забезпеченням, датчиками, роз’ємами та шлюзами;
- аналітика потоку даних у реальному часі зі складною обробкою подій;
- підтримка обсягу та швидкості даних;
- візуалізація великих даних;
- хмарні послуги та обчислення;
- окремі взаємодіючі протоколи зв'язку.

Через вищезазначені характеристики архітекторам IoT нелегко оцінити продуктивність і ефективність систем на основі IoT. Нижче наведено численні виклики тестування IoT[15]:

- динамічне середовище: ми не можемо слідувати принципам тестування програмного забезпечення, що працює у визначеному середовищі, хоча воно має справу з інтегрованою роботою різних

датчиків і пристроїв, керованих деяким прикладним програмним забезпеченням;

- складність: хоча система IoT працює на кількох пристроях і комунікаційних протоколах, існує великий набір варіантів використання, які ускладнюють тестування;
- масштабованість: це непросте завдання створити масштабоване тестове середовище, оскільки більшість систем на базі IoT є динамічними, і в будь-який час виникне необхідність розширити межі та можливості системи;
- надійність: через наявність пристроїв для обробки та взаємозв'язку неможливо точно виміряти надійність системи IoT;
- безпека: підсистемами та компонентами можуть керувати треті сторони, що може призвести до проблем конфіденційності та безпеки;
- якість і точність апаратного забезпечення: хоча система IoT використовує багато апаратного забезпечення, якість і ефективність кожного компонента, який ми використовуємо, повинні пройти ретельний контроль якості. Точність відіграє життєво важливу роль у більшості систем IoT.

Враховуючи вищезазначені проблеми, ми пропонуємо похідну тестову модель IoT, як показано на рис. 2.2. Запропонована тестова модель складається з чотирьох рівнів тестування. Кожен рівень виконує набір процедур верифікації та валідації для забезпечення функціональності та можливостей системи на основі IoT[16].

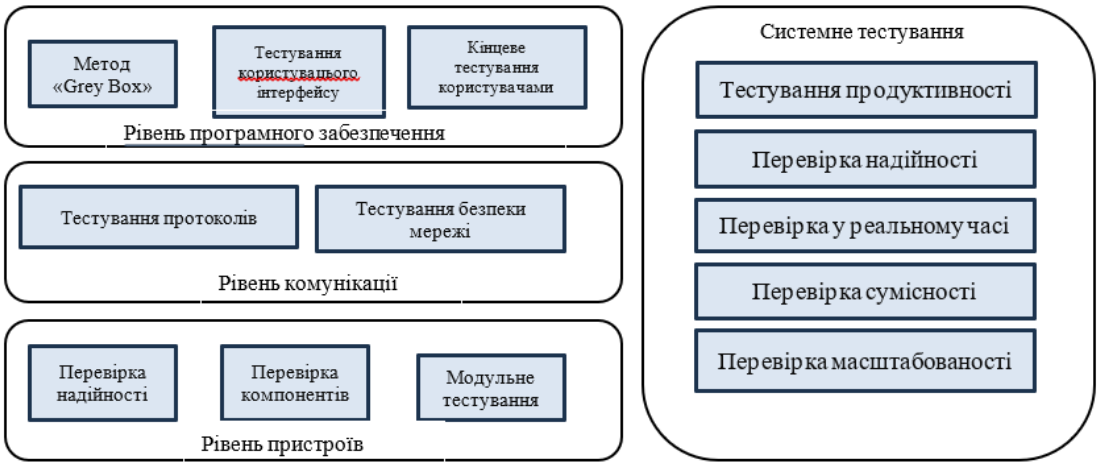

Рисунок 2.2 – запропонована тестова модель

2.2.1 Рівень пристроїв

Цей рівень вважається фізичним рівнем системи IoT. Фізичний рівень IoT складається з окремих компонентів, таких як датчики, пристрої моніторингу, такі як камера спостереження та центральний процесор (Raspberry Pi, Arduino та Intel Galileo). Система на основі IoT може використовувати або не використовувати датчики на основі домену програми. Датчики, які зазвичай використовуються, включають датчик температури, датчик наближення, датчик тиску, PIR (пасивний інфрачервоний) датчик руху, датчик вологості тощо. Розробники повинні переконатися, що компонент, який використовується на фізичному рівні, працює ідеально. Набір тестів, які необхідно провести на цьому рівні, включає тестування компонентів, модульне тестування та тестування надійності.

– тестування компонентів;

Різні класи компонентів, що використовуються в системі IoT, можуть бути виготовлені на сторонніх підприємствах або заводах. Розробник повинен переконатися, що кожен компонент точно та чітко виконує свої

функції. Тестування точності, перевірка кіл, функціональне тестування тощо слід проводити на цьому рівні для кожного компонента, який використовується в системі IoT.

– модульне тестування;

Для розробки фізичної частини системи IoT підключаються різні компоненти та пристрої. Кожен розроблений фізичний модуль проходить індивідуальну перевірку його функціональності.

– перевірка надійності;

Надійність описує здатність системи або компонента виконувати заплановані функції за заздалегідь визначених умов протягом певного періоду часу[18]. Вона також визначається як ймовірність того, що система готова виконувати свої завдання.

Значення надійності варіюються від 0 до 1. Надійність апаратного компонента можна або періодично перевіряти, або випадково оцінювати. Рівень відмов апаратних компонентів зазвичай відповідає кривій ванни, як показано на рисунку 2.3

Рисунок 2.3 – Надійність пристроїв

2.2.2 Рівень комунікацій

Дискретні пристрої зв’язуються з хмарним сервісом за допомогою мобільного або веб-додатку. Комунікаційна модель IoT може бути моделлю запит-відповідь, моделлю публікації-підписки, моделлю push-pull або модель ексклюзивної пари. Різні протоколи зв’язку, які зазвичай використовуються в типовій системі IoT, показані на рис. 2.4. Вона складається з чотирьох рівнів: канального рівня, рівня мережі або Інтернету, транспортного рівня та рівня додатків. Види тестування, які необхідно виконати на рівні зв’язку IoT, запропоновані нижче.

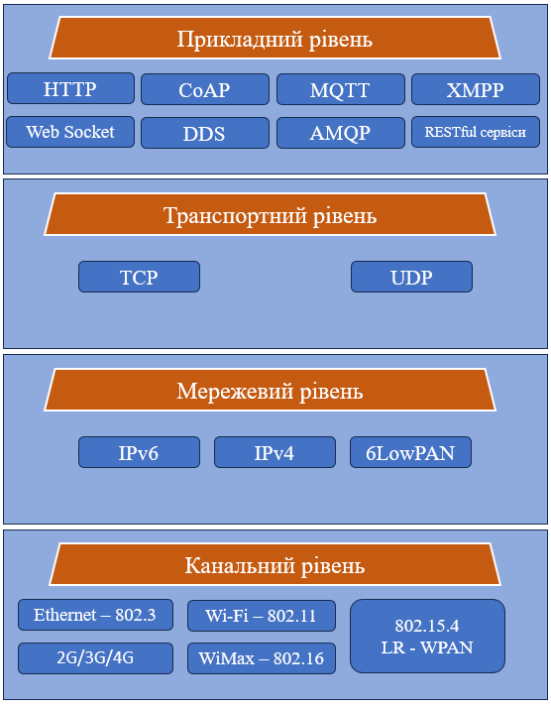

Рисунок 2.4 – протоколи IoT систем

– тестування протоколу та сумісності;

Незважаючи на те, що система використовує широкий спектр протоколів, розробник зобов’язаний гарантувати, що всі пристрої та

компоненти використовують сумісні протоколи зв’язку. Це тестування є обов’язковим, тому що ми використовуємо пристрої, які працюють від локальної мережі (BAN) до глобальної мережі (WAN). Для цього можна використовувати аналізатор протоколів і симулятор. Аналізатор протоколів забезпечує належне виконання декодування разом із аналізом викликів і сеансів. Симулятор імітує численні сутності та елементи мережі. Зазвичай тестування протоколу та сумісності виконується за допомогою DUT (Device Under Test) для інших пристроїв, таких як маршрутизатори та комутатори, шляхом налаштування протоколу всередині нього. Тестування відповідності, тестування мережевих функцій і негативне тестування також можуть виконуватися на цьому рівні.

– тестування безпеки та конфіденційності;

Хоча в розподіленому середовищі використовуються різні сумісні протоколи зв’язку, система на основі IoT дуже вразлива до загроз безпеці. Конфіденційність даних також має бути збережена в деяких областях застосування, як-от військові, моніторинг здоров’я тощо. У цьому сценарії розробник зобов’язаний забезпечити аспекти безпеки шифрування та дешифрування даних, конфіденційності та захисту даних, ідентифікації пристрою та довіри між різними хмарними та мобільними службами.

– тестування продуктивності;

Виконується для аналізу кількісної та якісної продуктивності розгорнутої системи на основі Інтернету речей у реальному часі з обмеженнями мережі. Додатково тестування продуктивності може проводитись за допомогою генерації тестових даних, це дозволить достатньо навантажити систему та якісно оцінити швидкодію[17]. Ця методологія включає тестування системи на основі IoT з поєднанням різних топологій, типу мережі, розміру мережі та інших умов середовища.

2.2.3 Рівень програмного забезпечення

Більшість систем, заснованих на IoT, складається з мобільних додатків або веб-додатків для віддаленого моніторингу та керування пристроями чи речами. Ці програми можуть працювати на різних платформах операційної системи IoT, як-от Windows 10 IoT Core, Raspbian, Snappy Ubuntu Core, PINET, RISC OS тощо. Але розробники завжди надають перевагу розробці програм на базі Android, ніж іншим альтернативам, щоб система IoT могла керувати за допомогою мобільних телефонів кінцевих користувачів. Рівень прикладного програмного забезпечення виконує загальні принципи тестування програмного забезпечення, як описано нижче.

– тестування «сірого ящика»;

Це гібридна форма тестування програмного забезпечення, яка об’єднує найкращі функції тестування як чорного ящика, так і тестування білого ящика. Він оцінює як поведінкові, так і функціональні особливості системи. Він гарантує, що розроблена система Інтернету речей виконує заплановані функції та побудована відповідно до стандартних вказівок. Для виявлення помилок інтерфейсу на цьому рівні також можна використовувати різні методи інтеграційного тестування. У разі постійної інтеграції модулів можна дотримуватися процедур регресійного тестування.

– тестування інтерфейсу;

Графічний інтерфейс користувача (GUI) має багато можливостей для додатків на основі Інтернету речей, тому як кінцевий користувач використовує деякі розумні пристрої для віддаленого керування об’єктами та речами. Графічний інтерфейс користувача має бути розроблений таким чином, щоб він був зручним і однозначним. При розробці інтерфейсу користувача слід дотримуватися стандартних позначень і значків. Його можна оцінити, надавши бета-версію кінцевому користувачеві, щоб можна

було зібрати відгуки кінцевих користувачів. На основі відгуків користувачів дизайн графічного інтерфейсу можна вдосконалити та зробити його більш прийнятним.

– тестування кінцевими користувачами;

Після розробки повної версії програми та її інтеграції з системою на основі Інтернету речей можна виконати тестування програми кінцевим користувачем. Це можна вважати своєрідним системним тестуванням.

Його можна включити в альфа-, бета-тестування та приймальне тестування. Альфа-тестування зазвичай виконується розробниками, щоб перевірити функціональність системи. Бета-тестування виконується випадковою групою кінцевих користувачів, щоб переконатися, що програмне забезпечення відповідає очікуванням користувачів. Приймальне тестування або тестування кінцевого користувача виконується цільовим користувачем, щоб визначити, прийняти чи відхилити цю систему на основі IoT.

2.2.4 Системний рівень

Тестування рівня системи в основному зосереджується на нефункціональних вимогах і продуктивності системи на основі IoT. Продуктивність – це масштабованість, надійність, доступність, портативність тощо. Системне тестування IoT включає наступні процедури тестування.

– тестування продуктивності;

Тестування продуктивності IoT здебільшого залежить від мережевих факторів, таких як затримка, пропускна здатність, втрата пакетів, обробка одночасних користувачів тощо. Тестові приклади слід створювати для типових і нетипових випадків використання, щоб впоратися з винятковими

випадками використання. Воно може бути доповнено тестуванням пікового навантаження.

– перевірка надійності;

Як вже було сказано, тестування надійності на рівні апаратних пристроїв і датчиків гарантує, що вся система на основі Інтернету речей працює точно протягом певного періоду часу без збоїв.

– тестування в реальному часі;

Тестування центрального процесору і віртуалізоване тестування не зафіксують усі помилки. Хоча системи IoT мають динамічну поведінку, існує ймовірність виникнення нового жанру помилок під час роботи в реальних життєвих ситуаціях. Тому перед розгортанням розробники повинні провести тестування в середовищі реального часу з реалістичними тестовими даними та умовами системи. Це дуже важливо у випадку додатків на основі Інтернету речей, які використовуються в охороні здоров'я, і програм віддаленого моніторингу, які використовуються в чутливих регіонах або місцях, де людина не може безпосередньо втручатися та контролювати.

– тестування сумісності;

Розроблена система IoT повинна мати можливість працювати в усіх стандартних умовах роботи. Хоча система IoT складається з різних пристроїв, які використовують різноманітні програмні та апаратні платформи, виникає потреба у виконанні тестування портативності.

– тестування масштабованості;

Завдяки динамічній поведінці систем IoT будь-коли діапазон або регіон застосування IoT можна розширити до більшої кількості пристроїв або речей. У цьому випадку вся система IoT повинна мати можливість розширюватися без збоїв. Це має бути забезпечено шляхом виконання процедур тестування

масштабованості, доповнених тестуванням навантаження, тестуванням обсягу, тестуванням на проникнення тощо.

Таким чином, наша запропонована модель розглядає всі розділи системи на основі IoT шляхом оцінки апаратного, програмного, комунікаційного та системного рівнів.

3 ІСНУЮЧІ ПРОТОКОЛИ ПЕРЕДАЧІ ДАНИХ У СИСТЕМАХ ІНТЕРНЕТУ РЕЧЕЙ

На рівні додатків представлені два популярних протокола передачі даних, а саме:

- HTTP;
- MQTT;

3.1 Протоколи передачі даних у системі «Розумний будинок»

Ці протоколи можуть та широко використовуються у системах інтернету речей, тому існує безліч статтей, мануалів, документації у яких детально описані всі можливості протоколів. Розглянемо кожний з них.

- MQTT[19],[20] — один із найстаріших комунікаційних протоколів M2M, який був представлений у 1999 році. Він був розроблений Енді Стенфорд-Кларком з IBM і Арленом Ніппером з Arcom Control Systems Ltd (Eurotech). Це протокол обміну повідомленнями для публікації/підписки, розроблений для легких комунікацій M2M у обмежених мережах [2]. Клієнт MQTT публікує повідомлення брокеру MQTT, на які підписані інші клієнти або які можуть бути збережені для майбутньої підписки (рис 3.1). Кожне повідомлення

публікується за адресою, відомою як тема [3]. Клієнти можуть підписатися на кілька тем і отримувати кожне повідомлення, опубліковане в кожній темі. MQTT є двійковим протоколом і зазвичай вимагає фіксованого заголовка з 2 байтів з невеликим корисним навантаженням повідомлення до максимального розміру 256 МБ [4]. Він використовує TCP як транспортний протокол і TLS/SSL для безпеки. Таким чином, спілкування між клієнтом і брокером орієнтоване на встановлення зв'язків. Ще однією чудовою особливістю MQTT є три рівні якості обслуговування (QoS) для надійної доставки повідомлень [2]. MQTT найбільше підходить для великих мереж невеликих пристроїв, які потрібно відстежувати або контролювати з внутрішнього сервера в Інтернеті. Він не призначений ні для передачі між пристроями, ні для багатоадресної передачі даних багатьом приймачам [3].

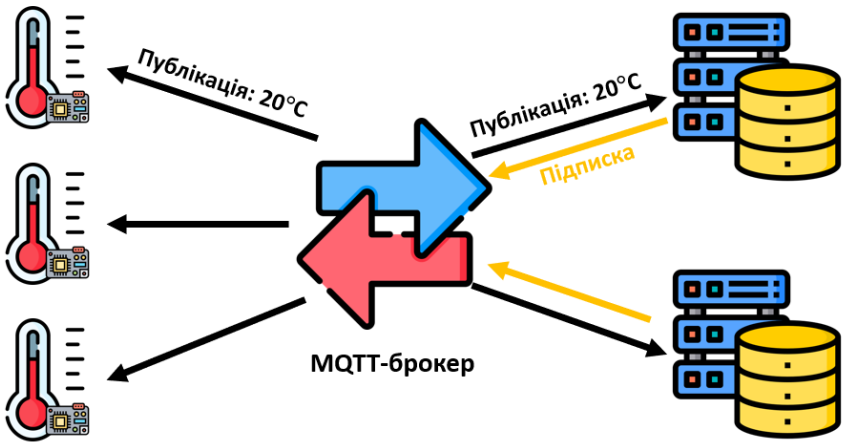

Рисунок 3.1 - схема роботи протоколу MQTT

– HTTP[21],[22] — це переважно протокол обміну веб-повідомленнями, спочатку розроблений Тімом Бернерсом-Лі. Пізніше він був спільно розроблений IETF і W3C і вперше опублікований як стандартний протокол у 1997 році. HTTP підтримує RESTfull вебархітектуру запит/відповідь (рис. 3.3). HTTP використовує універсальний ідентифікатор ресурсу (URI) замість

тем. Сервер надсилає дані через URI, а клієнт отримує дані через певний URI. HTTP — це текстовий протокол, який не визначає розмір заголовків і корисних повідомлень, а залежить від веб-сервера або технології програмування. HTTP використовує TCP як транспортний протокол за замовчуванням і TLS/SSL для безпеки. Таким чином, зв'язок між клієнтом і сервером є орієнтованим на з'єднання. Він явно не визначає QoS і вимагає додаткової підтримки для цього.

Рисунок 3.3 - Схема роботи протоколу HTTP

Для більш детального ознайомлення з характеристиками протоколів у таблиці 2.1 були зібрані основні характеристиками обраних протоколів.

Таблиця 3.1 – Характеристики протоколів передачі даних

Критерії	MQTT	HTTP
1. Рік	1999 рік	1997 рік
2. Архітектура	Клієнт/Брокер	Клієнт/Сервер
3. Абстракція	Публікація/Підписка	Запит/Відповідь
4. Розмір заголовку	2 байт	Невизначено
5. Розмір повідомлення	Малий і невизначений (максимальний розмір до 256 Мб)	Великий і невизначений(залежить від типу серверу та технологій програмування)

Продовження таблиці 3.1

Критерії	MQTT	HTTP
6. Семантика/Методи	Підключити, Відключити, Опублікувати, Підписатись, Відписатись, Закрити	GET, POST, HEAD, PUT, PATCH, OPTIONS, CONNECT, DELETE
7. Підтримка кешу та проксі	Частково	Так
8. Якість обслуговування/Надійність	QoS 0 – максимум один раз(відправив та забув) QoS 1 – принаймні один раз QoS 2 - рівно один раз	Обмежено (методи транспортного протоколу TCP)
9. Стандартизація	OASIS, Eclipse Foundation	IETF, W3C
10.Транспортний протокол	TCP	TCP
11.Безпека	TLS/SSL	TLS/SSL
12. Порт за замовчуванням	1883/8883 (TLS/SSL)	80/443 (TLS/SSL)
13.Формат кодування	Двійковий	Текст
14.Модель ліцензування	Open Source	Безкоштовно
15.Підтримка огранізацій	IBM, Facebook, Eurotech, Cisco, Red Hat, Software AG, Amazon Web Services (AWS)	Global Web Protocol Standard

3.2 Відносний аналіз протоколів MQTT, HTTP

Для кращого розуміння параметрів та вибору найкраще підходящого протоколу передачі повідомлень був проведений відносний аналіз протоколів. У цьому розділі представлений поглиблений і відносний аналіз цих трьох протоколів обміну повідомленнями для систем інтернету речей. Був проведений аналіз критеріїв, для того щоб визначити відповідні переваги та обмеження кожного протоколу обміну повідомленнями. Ці протоколи обміну дуже обширні, та відрізняються один від одного, оскільки розвивалися через різні процеси та потреби. Це відносне порівняння базується на лінгвістичному діапазоні «нижчий» і «вищий», щоб надати швидке та шире уявлення про кожен протокол щодо іншого протоколу. Ця оцінка базується на основі статичних компонентах і деяких емпіричних даних з літератури. Тим не менш, він не враховує динамічні умови мережі та накладні витрати, які виникають під час повторної передачі пакетів, що також може змінити результати порівняння.

3.2.1 Розмір повідомлення проти накладних витрат

Розглянемо процес встановлення з’єднання та відправки/отримання повідомлень. Протокол MQTT основується на з’єднаннях клієнта с сервером брокером. Кожен клієнт має унікальний ідентифікатор, за допомогою якого брокер може регулювати правила підключення. Додатково може використовуватися ідентифікація за допомогою імені користувача та паролю. Також може використовуватися шифрована передача поверх TLS/SSL. З’єднання за протоколом MQTT починається з того, що клієнт відправляє повідомлення CONNECT брокеру. Ініціювати сеанс може тільки клієнт,

жоден клієнт не може безпосередньо з’єднуватися з іншим клієнтом. На повідомлення CONNECT брокер завжди відправляє відповідь CONNACK і код статусу. Після встановлення з’єднання, воно починає працювати. Клієнт може отримувати та відправляти повідомлення без виконання процедури ініціювання з’єднання. Якщо треба закрити сеанс, клієнт повинен відправити повідомлення DISCONNECT, після цього сеанс закривається. Схема роботи обміну повідомленнями показана на рисунку 3.4.

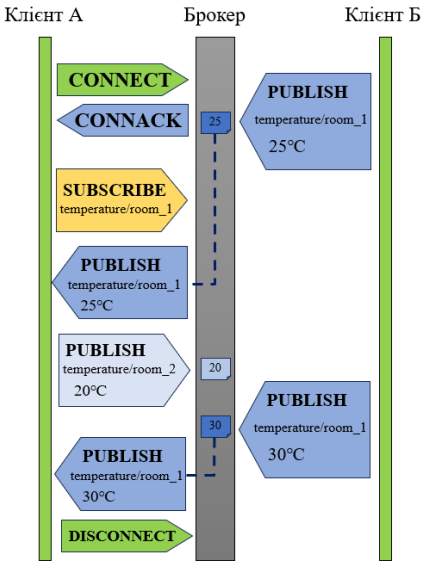

Рисунок 3.4 – Схема роботи протоколу MQTT

Для протоколу HTTP схема роботи відрізняється, незважаючи на те що обидва протоколи засновані на TCP-з’єданнях, для кожної передачі повідомлення через HTTP необхідно заново встановлювати TCP-з’єдняння. Це сповільнює передачу маленьких повідомлень, які є доволі частими у системах «розумного будинку». Зазвичай можливо використовувати протокол HTTP під час передачі даних між системами, де є необхідність передачі великого об’єму даних за один раз. Схема роботи протоколу HTTP показана на рисунку 3.5.

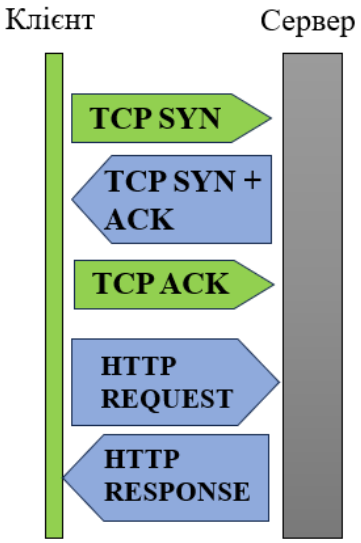

Рисунок 3.5 – Схема роботи протоколу HTTP

Для вимірювання кількості байтів які необхідні для встановлення зв’язку, передачі даних, та закриття з’єднання було відправлено повідомлення у форматі JSON.

```
{"test":1234}
```

Результати проведення вимірювання показані у таблицях 3.2, 3.3.

Таблиця 3.2 – результати вимірювання кількості байтів для захищених сесій

Захищена сесія	Вихідні байти	Вхідні байти	Кількість пакетів
HTTPS	1734	4186	20
MQTT через SSL (Ethernet)	1274	4159	20
MQTT через SSL (WiFi)	1186	4075	18

Таблиця 3.3 – результати вимірювання кількості байтів для незахищених сесій

TCP сесія	Вихідні байти	Вхідні байти	Кількість пакетів
HTTP	675	431	10
MQTT (Ethernet)	615	352	11
MQTT (WiFi)	601	342	11

За результатами вимірювань можна зробити висновок що протокол MQTT не сильно легший під час відправки одного повідомлення, різниця становить 10%. Показники можливо покращити якщо не використовувати команду DISCONNECT при використанні протоколу MQTT.

Розглянемо реальний сценарій використання протоколів. Сценарій представляє собою хаб, який збирає дані з вимірювальних пристроїв. Завдання передати дані до додатку використовуючи кожний протокол, та виміряти кількість байтів, пакетів, та час за який проходить передача повідомлень. Вимірювання проводились для 100 повідомлень з використанням наступних сценаріїв:

а) клієнт MQTT підтримує з’єднання та публікує кожне повідомлення окремо;
б) HTTP keep-alive з’єднання із POST запитом для кожного повідомлення
в) одиночний HTTP POST запит з пачкою повідомлень;

Сценарії «а» та «б» мать перевагу над сценарієм «в» в тому що дані будуть доставлені якнайшвидше, та вони будуть актуальними. Сценарій «в» не підходить для передачі даних між вимірювальними пристроями та хабом, але може бути використаний при зберіганні даних у віддаленому сховищі, або при передачі даних між великими системами. Тому сценарій «в» представлений для наочності. Результати вимірювань представлені у таблиці 3.4.

Таблиця 3.4 – результати вимірювання кількості байтів, пакетів, та кількості витраченого часу для передачі

1000 повідомлень	Передані байти	Кількість пакетів	Час, секунди
MQTT з використанням SSL 1 повідомлення на публікацію, єдина сесія, QoS = 1	283743	265	5,911
HTTPS 1 POST запит для кожного повідомлення, keep-alive з'єднання	15474263	12079	115,669
HTTPS 1 POST запит з 1000 повідомленнями	20515	27	0,307

Результати показують що протокол MQTT у 20 разів швидший та використовує у 54,5 разів менше даних при передачі даних які залежні від часу та мають бути якнайшвидше доставлені.

На рисунку 3.6 показано графік порівняння цих протоколів обміну повідомленнями на основі їх розміру повідомлення та накладних витрат на повідомлення. Під накладними витратами розуміється час витрачений на відправку повідомлень.

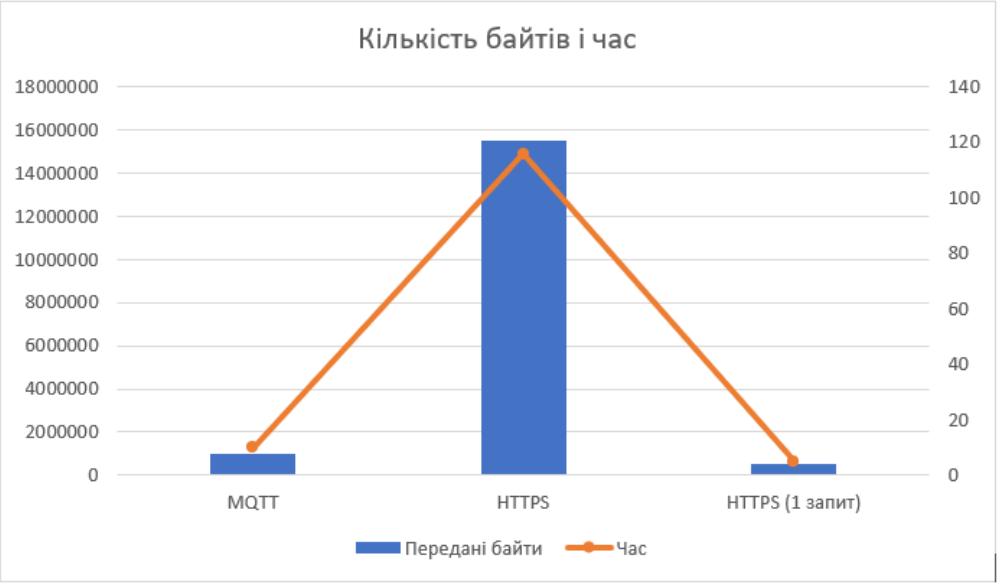

Рисунок 3.6 - Розмір повідомлень в порівнянні з часом відправки

На графіку показано, що HTTP несе найбільший розмір повідомлення та накладні витрати, а потім розмір та накладні витрати зменшуються до протокола MQTT, який має найменший розмір повідомлення та накладні витрати. MQTT і HTTP працюють на TCP; отже, вони несуть усі накладні витрати TCP-з'єднання на встановлення та закриття з'єднання. Однак MQTT не є легким і має найменший розмір заголовка 2-байт на повідомлення, але його вимога TCP-з'єднання збільшує загальні накладні витрати, а отже, і весь розмір повідомлення. HTTP є найбільш детальним і важким протоколом [10]. Спочатку він був розроблений для Інтернету, а не для IoT; отже, це вимагає максимальних накладних витрат і розміру повідомлення серед усіх.

3.2.2 Енергоспоживання та використання ресурсів

Енергоспоживання має велике значення для систем «розумного будинку» це впливає на можливість зберігати працездатність при відключенні електроенергії або при роботі від вбудованих елементів живлення. Для вимірювання енергоспоживання був обраний контролер Raspberry Pi 2 model B, цей контролер підтримує використання обох

протоколів для передачі даних з використанням SSL, є можливість вимкнути всі непотрібні сервіси Linux.

Було прийнято рішення не використовувати амперметри та вольтметри, був використаний акумулятор з характеристиками 5В 200мАг, налаштований контролер з безкінечним циклом для MQTT або HTTP та були виміряні:

а) кількість сесій до вимкнення контролеру;
б) час роботи контролеру до вимкнення;
в) середнє використання центрального процесору.

З проведеного дослідження були визначені два основні результати: кількість мАг на сесію та відсоток навантаження на центральний процесор. Результати представлені у таблиці 3.5.

Таблиця 3.5 – результати вимірювань енергоспоживання та використання ресурсів

Параметр	З'єднання за допомогою 3G модему		З'єднання за допомогою Ethernet	
	REST API з використанням curl	Mosquito MQTT клієнт	REST API з використанням curl	Mosquito MQTT клієнт
Кількість сесій	9243	11728	20964	22864
Час виконання тесту	2год 39хв	2год 50хв	5год 53хв	6год 21хв

Продовження таблиці 3.5

Параметр	З’єднання за допомогою 3G модему		З’єднання за допомогою Ethernet	
	REST API з використанням curl	Mosquito MQTT клієнт	REST API з використанням curl	Mosquito MQTT клієнт
Кількість мАг на сесію	0,2164	0,17	0,0954	0,089
Повідомлень на секунду	0,97	1,15	0,99	0,98
Використання центрального процесору, %	64	60	69	64

На рисунку 3.5 показаний графік порівняння цих протоколів обміну повідомленнями на основі їх енергоспоживання та навантаження на ресурси. На графіку показано подібні закономірності, як і в першому, де HTTP вимагає найбільшої потужності та ресурсу, ніж будь-які інші протоколи, а потім він до протоколу MQTT, який потребує найменшої потужності та ресурсу. MQTT розроблений для пристроїв з низькою пропускною здатністю та обмеженими ресурсами та можуть використовуватися на 8-розрядному контролері та 100 байтах пам’яті.

За отриманими результатами можна зробити висновок що навіть у короткостроковій перспективі MQTT на 22% енергоефективніший і на 15% швидший. І це не залежить від типу підключення до мережі, з використанням 3G або Ethernet.

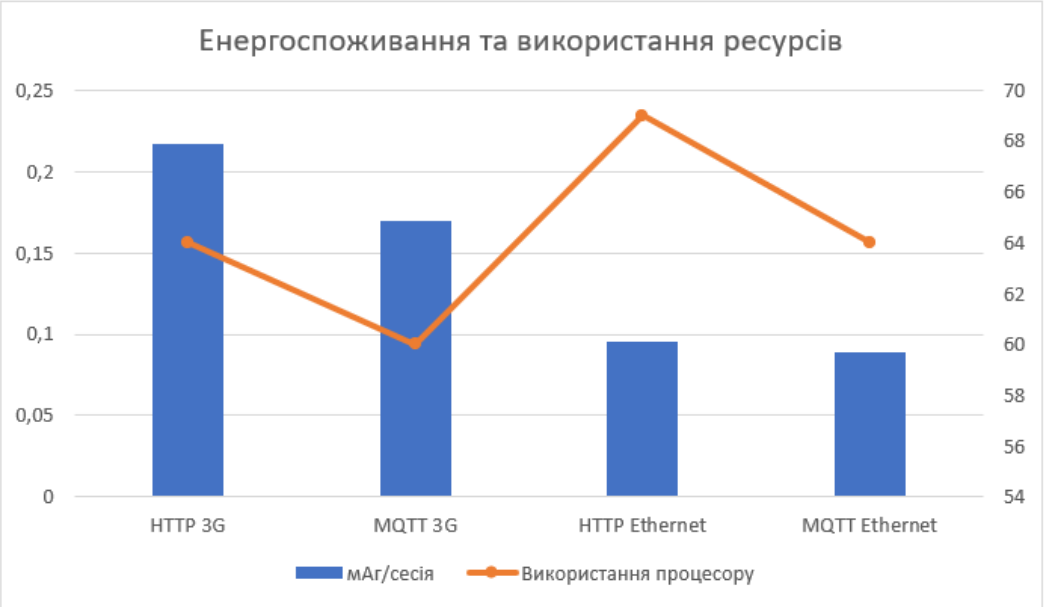

Рисунок 3.5 - Енергоспоживання та використання ресурсів

3.2.3 Затримка

Затримка – це запізнення під час мережного обміну даними. Тут відображається час, необхідний передачі даних по мережі. Мережі з більш тривалою затримкою мають високу пропускну здатність, у той час як мережі зі швидким відгуком мають нижчу пропускну здатність. На відміну від цього пропускна здатність – це середній обсяг даних, який може пройти через мережу протягом певного часу. У цьому вимірюванні представлені результати вимірювання затримки з використанням HTTP та MQTT протоколів. Використана бібліотека flespi_receiver яка дозволяє приймати та обробляти повідомлення через REST API. Для MQTT протоколу використаний брокер Mosquito. Результати представлені у таблиці 3.6 та на рисунку 3.6.

Таблиця 3.6 – Затримка під час використання протоколів HTTP та MQTT

Затримка від клієнта до брокера, секунд	REST API модуль	MQTT Mosquito брокер
Середня	0,768	0,032
Максимальна	1,274	0,035
Мінімальна	0,379	0,032

На рисунку 3.6 показано графік затримки цих протоколів обміну повідомленнями. Графік показує дуже схожі закономірності, як і перші два, де HTTP передбачає найбільшу затримку, ніж будь-які інші протоколи, а потім затримка зменшується до MQTT, що має нижчу затримку. Використання TCP у MQTT і HTTP є основним фактором у визначенні вимог до затримки. На жаль, TCP не допомагає зменшити затримку. Він не повністю використовує доступну пропускну здатність мережі для перших кількох проходжень з’єднання через повільний підхід до запуску, щоб уникнути перевантаження мережі. Відправник TCP поступово відкриває вікно перевантаження та подвоює кількість пакетів за кожен час зворотного проходження(RTT). Але використання одного з’єднання для протоколу MQTT дозволяє добитись низьких показників затримки, що гарно впливає на загальну швидкодію мережі.

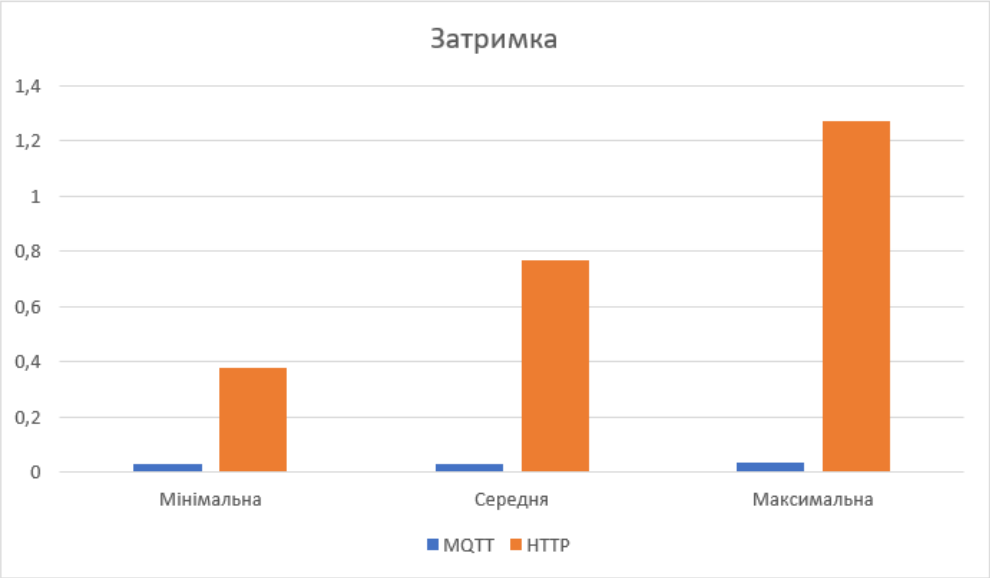

Рисунок 3.6 - Затримка

Виходячи з вищесказаного та з результатів вимірювань у випадку створення розумного будинку на системі ESP32 найкраще підходить протокол передачі повідомлень MQTT. Цей протокол виграє за показниками розміру повідомлення, швидкості відправки повідомлень, затримкою під час використання протоколу для відправки невеликих повідомлень від вимірювальних пристроїв до хабу. Основним видом передачі даних у системі «розумний будинок» є передача даних від вимірювальних пристроїв до хабу та від хабу до виконуючих пристроїв. Це доволі сучасний протокол, який є легковісним та невибагливим до апаратних ресурсів. Має гарні показники пропускної здатності, та достатньо гарну надійність, що може гарантувати доставку повідомлення до кінцевого підписника. Додатково цей протокол підтримують багато міжнародних компаній, які є гігантами у сфері інтернету речей. Цей протокол підходить якнайбільше у системі «розумна» вентиляція.

4 ПРИКЛАД ВИКОРИСТАННЯ ПРОТОКОЛУ MQTT

Система зв'язку, побудована на MQTT, складається з сервера-видавця, сервера-брокера та одного або кількох клієнтів, схема представлена на рисунку 4.1. Видавець не вимагає жодних налаштувань за кількістю або розташуванням передплатників, які отримують повідомлення. Крім того, передплатникам не потрібне налаштування на конкретного видавця. У системі може бути кілька брокерів, які розповсюджують повідомлення.

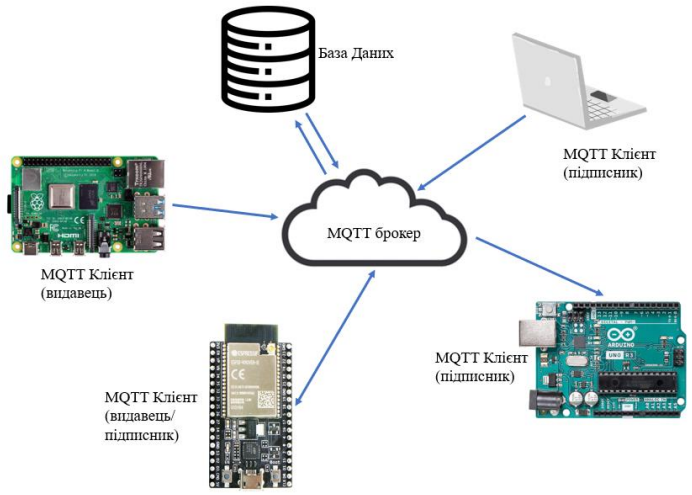

Рисунок 4.1 – Схема підключення пристроїв до брокеру

MQTT надає спосіб створення ієрархії каналів зв'язку - свого роду гілка з листям. Щоразу, коли видавець має нові дані для поширення серед клієнтів, повідомлення супроводжується приміткою контролю доставки. Клієнти вищого рівня можуть отримувати кожне повідомлення, тоді як клієнти нижчого рівня можуть отримувати повідомлення, які стосуються лише одному чи двом базовим каналам, «відгалуженим» у нижній частині ієрархії. Це полегшує обмін інформацією розміром від двох байт до 256 мегабайт.

Згідно до специфікації протоколу[13], MQTT має наступні властивості:

- нейтральний до вмісту повідомлення;
- ідеально підходить для розподілених комунікацій «один до багатьох» та роз'єднаних додатків;
- оснащений функцією LWT (Last Will and Testament, «остання воля і заповіт») для повідомлення сторін про аномальне відключення клієнта;
- покладається на TCP/IP для базових завдань зв'язку;
- розроблено для доставки повідомлень за шаблонами «максимум один раз», «мінімум один раз» та «одноразово».

Учасник системи MQTT може взяти він роль видавця, споживача чи обидві ролі відразу.

Однією з відмінностей MQTT є унікальне розуміння каналів: кожен з них обробляється як шлях до файлу, наприклад:

```
channel = "user/path/channel"
```

Канали гарантують, що кожен клієнт отримує повідомлення, призначені йому. Обробляючи канали як шляхи до файлів, MQTT виконує всі види корисних функцій зв'язку, у тому числі фільтрацію повідомлень на основі того, де на якому рівні або в якій гілці клієнти підписуються на шлях до файлу.

Розглянемо два компоненти, з яких складається кожне повідомлення за протоколом MQTT:

- байт 1: містить тип повідомлення (запит клієнта на підключення, підтвердження підписки, запит ping тощо), прапор дублювання, інструкції для збереження повідомлень та інформацію про рівень якості обслуговування (QoS);

– байт 2: містить інформацію про довжину повідомлення, що залишилася, включаючи корисне навантаження і будь-які дані в заголовку необов'язкової змінної.

Прапор QoS в байті 1 заслуговує на особливу увагу, оскільки він лежить в основі змінної функціональності, яку підтримує MQTT. Прапори QoS містять такі значення, засновані на намірі та терміновості повідомлення:

– 0 – трохи більше одного разу: сервер спрацьовує і забуває. Повідомлення можуть бути втрачені або продубльовані;
– 1 – принаймні один раз: одержувач підтверджує доставку. Повідомлення можуть дублюватися, але доставка гарантована;
– 2 – одноразово: сервер забезпечує доставку. Повідомлення надходять точно один раз без втрати чи дублювання

4.1 Схема роботи проекту

У прикладі демонструється як виконується налаштування MQTT для обміну даними між двома платами ESP32. Саме цей контролер використовується тому що він був обраний як основний для використання в системі «розумна вентиляція». Був сконструйований проект, для ілюстрації найважливіших концептів MQTT. На рисунку 4.2 продемонстрована схема проекту. Використовуються дві ESP32 плати.

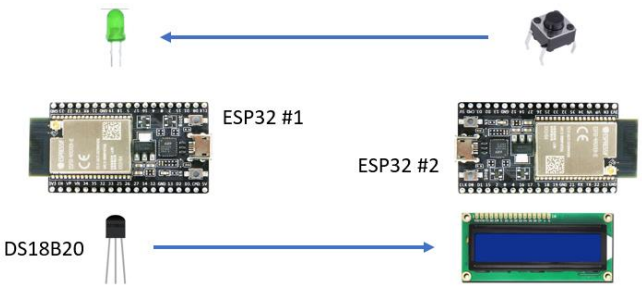

Рисунок 4.2 – Схема проєкту

– ESP32 #1 підключена до світлодіоду та зчитує температуру з датчика DS18B20;
– ESP32#2 підключена до кнопки, при натисканні на кнопку, світлодіод, підключений до ESP32 #1 буде вмикатися;
– ESP32#2 підключена до LCD-дисплею, на якому будуть виводитися дані про температуру, отримані від ESP32#1.

На рисунку 4.3 наглядно показано, які функції виконує MQTT-брокер.

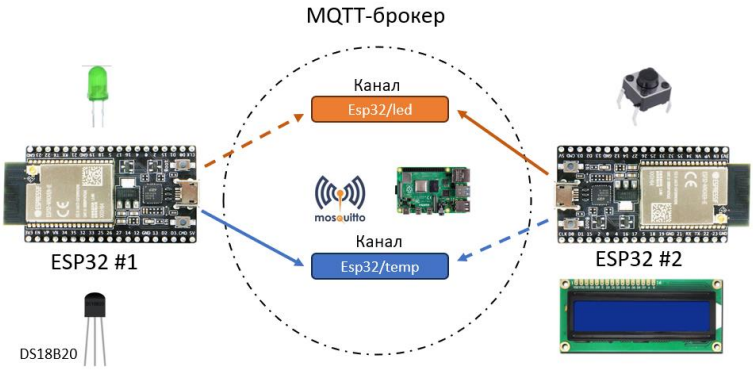

Рисунок 4.3 – Схема передачі даних

– ESP32#1 підписана на канал «esp32/led» та публікує дані про температуру у канал «esp32/temperature»;
– При натисканні на кнопку, підключену до ESP32#2, цей контролер буде публікувати відповідне повідомлення у канал «esp32/led», за допомогою якого керується світлодіод, підключений до ESP32#1.

4.2 Програмування контролеру ESP32#1

Фрагмент коду нижче імпортує у скетч усі необхідні бібліотеки для роботи:

```
#include <WiFi.h>
extern "C" {
#include "freertos/FreeRTOS.h"
```

```
#include "freertos/timers.h"
}
#include <AsyncMqttClient.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Далі необхідно вписати у скетч SSID та пароль від Wi-Fi мережі, а також вказати адрес MQTT-брокеру.

```
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"
#define MQTT_HOST IPAddress(192, 168, 1, XXX)
#define MQTT_PORT 1883
```

Створення об’єкту для управління MQTT-клієнтом та таймери, які знадобляться для повторного підключення до MQTT-брокеру або Wi-Fi-мережі, якщо зв’язок раптом обірветься.

```
//об’єкти для управління MQTT-клієнтом
AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;
```

Оголошено декілька змінних: одну для зберігання даних про температуру та дві допоміжні таймерні змінні – для зчитування даних кожні 5 секунд, задання станів вихідних контактів, та присвоєння контактів, див. Додаток А.

Розглянемо деякі MQTT-функції: підключення до Wi-Fi, підключення до MQTT, WiFi-події. Наприклад, функція connectToWiFi() підключає ESP32 до Wi-Fi-мережі:

```
void connectToWifi() {
Serial.println("Connecting to Wi-Fi...");
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}
```

Функція connectToMqtt() підключає ESP32 до MQTT-брокеру:

```
void connectToMqtt() {
Serial.println("Connecting to MQTT...");
mqttClient.connect();
}
```

Функція WiFiEvent() (див. Додаток Б) відповідає за керування WiFi-подіями. Наприклад, після успішного підключення до мережі та MQTT-брокеру вона виведе IP-адрес ESP32. Крім того, при обриві з'єднання вона запустить таймер та спробує встановити з'єднання.

Задача функції onMqttConnect() – підписка ESP32 на канали.

```
void onMqttConnect(bool sessionPresent) {
     Serial.println("Connected to MQTT.");
     Serial.print("Session present: ");
     Serial.println(sessionPresent);
     //підключення до каналу  "esp32/led"
     uint16_t                packetIdSub                =
mqttClient.subscribe("esp32/led", 0);
     Serial.print("Subscribing at QoS 0, packetId: ");
     Serial.println(packetIdSub);
}
```

Наступний рядок – важлива частина показаного фрагменту. Вона підписує ESP32 на MQTT-канал за допомогою метода subscribe(). Параметрами цього методу є MQTT-канал, на який необхідно підписати ESP32 та рівень якості обслуговування QoS.

```
uint16_t packetIdSub = mqttClient.subscribe("esp32/led", 0);
```

Додаткові «сервісні» функції: відключення, підписка, відписка та публікація.

Якщо ESP32 втратить з'єднання з MQTT-брокером, функція onMqttDisconnect() виведе у моніторі порту відповідне повідомлення.

```
void        onMqttDisconnect(AsyncMqttClientDisconnectReason
reason) {
```

```
    Serial.println("Disconnected from MQTT.");
    if (WiFi.isConnected()) {
        xTimerStart(mqttReconnectTimer, 0);
    }
}
```

Якщо ESP32 підпишеться на MQTT-канал, функція onMqttSubscribe() напечатає ID пакету та рівень якості обслуговування (QoS):

```
void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print("  packetId: ");
    Serial.println(packetId);
    Serial.print("  qos: ");
    Serial.println(qos);
}
```

Якщо ESP32 відпишеться від каналу, функція onMqttUnsubscribe() напетатає повідомлення про це:

```
void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print("  packetId: ");
    Serial.println(packetId);
}
```

І якщо у MQTT-каналі буде опубліковано нове повідомлення, функція onMqttPublich() напечатає у моніторі порту ID цього пакету:

```
void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print("  packetId: ");
    Serial.println(packetId);
}
```

Отримання MQTT-повідомлень. Якщо у канал, на який підписана ESP32 (у даному випадку – «esp32/led») прийде нове повідомлення, буде виконана функція onMqttMessage() (див. Додаток Г). У ній задається те, що

повинно бути зроблене при з’явленні у цьому каналі нового повідомлення, ці події можна редагувати.

Наступний етап, написання коду функції setup(). Запускаю датчик DS18B20, роблю контакт світлодіоду вихідним, задаю йому значення LOW та запускаю послідовну комунікацію.

```
sensors.begin();
pinMode(ledPin, OUTPUT);
digitalWrite(ledPin, LOW);
Serial.begin(115200);
```

В наступних рядках створюю таймери для повторного підключення до MQTT-брокеру та WiFi-мережі у випадку, якщо з’єднання буде втрачено.

```
    mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
    wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));
```

Рядок нижче задає, щоб при підключенні ESP32 до WiFi-мережі була запущена функція зворотнього виклику WiFiEvent() яка напечатає дані про WiFi-з’єднання.

```
WiFi.onEvent(WiFiEvent);
```

Всі функції які створено вище мають бути присвоєні до івентів. Це означає, що всі ці функції будуть викликані автоматично – у випадку, якщо вони знадобляться. Наприклад, коли ESP32 підключиться до брокеру, автоматично буде викликана функція onMqttConnect().

```
mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
```

У блоці loop() створений таймер, який дозволяє публікувати нові температурні дані у каналі «esp32/temperature» кожні 5 секунд. Таймер представлений у додатку Б.

На рисунку 4.4 показано результат успішного підключення мікроконтролера ESP32#1 до мережі Wi-Fi та успішної підписки та відправки даних до відповідних каналів.

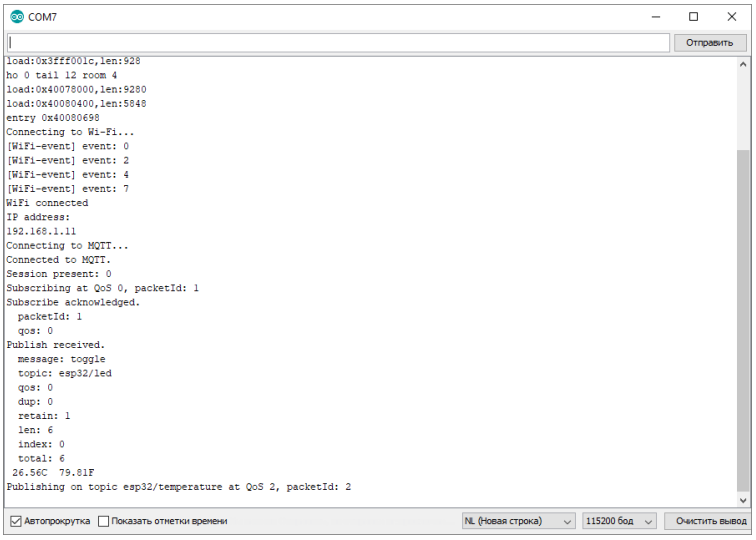

Рисунок 4.4 – Результат успішного підключення контролера ESP32#1

4.3 Програмування контролеру ESP32#2

Більша частина скетчу аналогічна до розділу 4.2. (див. розділ 4.2) Додається код для управління LCD-дисплеєм, а саме:

Кількість стовбців та рядків дисплея. У нашому випадку дисплей розміром 16x2 символів.

```
//кількість стовпців та рядків LCD-дисплея:
const int lcdColumns = 16;
const int lcdRows = 2;
```

Нижче представлений байтовий масив для того, щоб показати на LCD-дисплеї іконку термометра.

```
// іконка термометра:
byte thermometerIcon[8] = {
     B00100,
     B01010,
     B01010,
     B01010,
     B01010,
     B10001,
     B11111,
     B01110
};
```

Заданий контакт, до якого підключена кнопка, змінну для поточного стану кнопки, змінну для останнього стану кнопки та допоміжні змінні для створення таймера антидребезжання(щоб уникнути «фейкових» натискань на кнопку).

```
// GPIO-контакт, до якого підключена кнопка
const int buttonPin = 32;
//поточне значення вхідного контакту(кнопки)
int buttonState;
//попереднє значення вхідного контакту(кнопки)
int lastButtonState = LOW;
//час, коли в останній раз перемикався вихідний контакт
unsigned long lastDebounceTime = 0;
//час антидребезжання (якщо входяще значення «стрибає»,
необхідно збільшити значення)
unsigned long debounceDelay = 50;
```

Функції connectToWiFi(), connectToMQTT() та WiFiEvent() ідентичні до одноіменних функцій у розділі 4.2.(див. розділ 4.2)

У функції onMqttConnect() додані канали для підписки.

```
void onMqttConnect(bool sessionPresent) {
     Serial.println("Connected to MQTT.");
     Serial.print("Session present: ");
```

```
        Serial.println(sessionPresent);
        uint16_t packetIdSub = mqttClient.subscribe("esp32/temperature", 0);
        Serial.print("Subscribing at QoS 0, packetId: ");
        Serial.println(packetIdSub);
    }
```

Як розповідалось у попередньому розділі, у функції onMqttMessage() треба задати поведінку, коли у канал на який підписана ESP32 прийде нове повідомлення. У цю функцію можливо додати додаткові оператори if() – для перевірки інших каналів та вмісту повідомлень.

У випадку коли у канал «esp32/temperature» прийде нове повідомлення, ESP32 покаже його на LCD-дисплеї, який до неї підключений.

У блоці setup() запускається LCD-дисплей, вмикається підсвітка та виводиться іконка термометру.

```
    // ініціалізую LCD-дисплей:
    lcd.init();
    // вмикаю підсвітку LCD-дисплея:
    lcd.backlight();
    // створюю іконку термометра:
    lcd.createChar(0, thermometerIcon);
```

Створені таймери для повторного підключення при втраті останнього.

```
      mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
      wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));
```

Задані функції зворотнього виклику для Wi-Fi та MQTT-подій, створених раніше.

```
    WiFi.onEvent(WiFiEvent);
    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
```

```
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
```

У блоці loop() публікються повідомлення у канал «esp32/led», якщо кнопка була натиснута. Якщо кнопка була натиснута, оператор if() нижче поверне «true» а у каналі «esp32/led» буде опубліковано повідомлення «toggle».

```
if (buttonState == HIGH) {
     mqttClient.publish("esp32/led", 0, true, "toggle");
     Serial.println("Publishing on topic esp32/led topic at QoS 0");}
```

На рисунку 4.5 показане успішне підключення мікроконтролера до мережі WiFi. Також показане логування основних кроків підписки, отримання повідомлення, та відправки повідомлення.

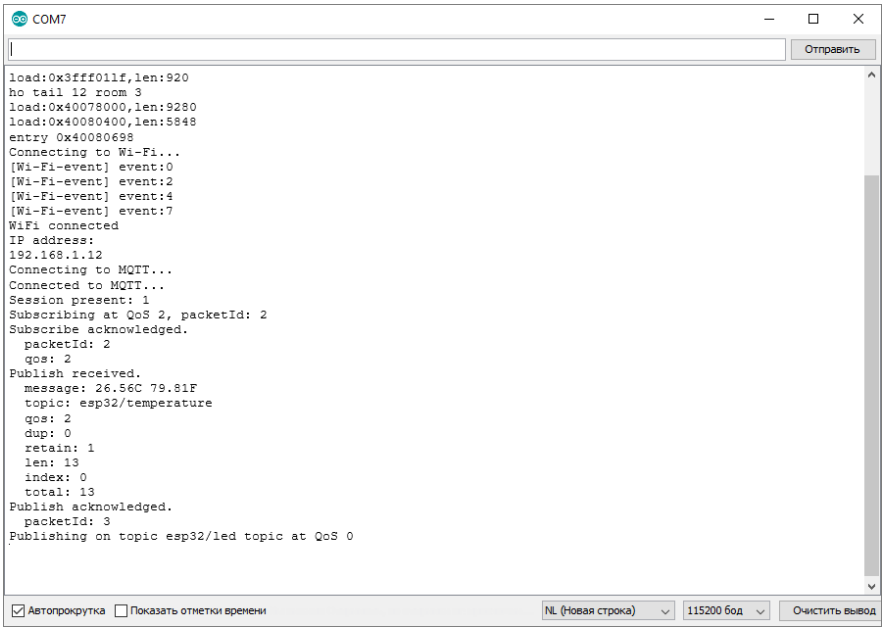

Рисунок 4.5 – Результат успішного підключення мікроконтролера ESP32#2

ВИСНОВКИ

У кваліфікаційній роботі було виконано дослідження масштабованих систем «розумний будинок» а саме протоколи передачі даних. У процесі дослідження були проаналізовані стандарти, моделі оцінки якості програмного забезпечення. Були проаналізовані протоколи передачі даних у системі «розумний будинок». Змістом роботи було: на основі існуючих моделей та стандартів оцінки якості програмного забезпечення обгрунтувати комплексну модель оцінки якості саме систем інтернету речей «розумний будинок» якої є розділом, підвищити якість роботи системи «розумний будинок» шляхом порівняння двох обраних протоколів передачі даних. Представити приклад використання обраного протоколу передачі даних у системі «розумного будинку»

В кваліфікаційній роботі розв’язана низка взаємопов’язаних часткових завдань, розв’язання яких дозволило досягти мети роботи.

В роботі розроблено підхід до оцінки якості систем «розумний будинок» та проведений один із рівнів оцінки якості який включає виміри енергоефективності, розміру переданих даних, швидкодії, затримки та навантаження на контролер.

В роботу представлений приклад використання протоколу який краще підходить до системи що розробляється.

Виконання роботи дозволяє підвищити якість роботи системи «розумний будинок» шляхом використання кращого протоколу передачі даних, та кращої швидкості обробки даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Крепич С. Я. Якість програмного забезпечення та тестування: базовий курс : навч. посіб. / С. Я. Крепич, І. Я. Співак. – Тернопіль : ФОП Паляниця В.А., 2020. – 478 с.
2. Ghezzi C. Fundamentals of software engineering / Carlo Ghezzi. – 2nd ed. – Upper Saddle River, NJ : Prentice Hall, 2003. – 604 p.
3. Лавріщева К. М. Основи програмної інженерії : Підручник / К. М. Лавріщева. – Київ : Знання, 2008. – 319 с.
4. ДСТУ ISO 9001:2015. Системи управління якістю. Вимоги. – Чинний від 2016-07-01. – Вид. офіц. – Київ : ДП «УкрНДНЦ», 2016. – 21 с.
5. ДСТУ ISO/IEC 25000:2015. ДСТУ ISO/IEC 25000:2015 Інженерія програмних засобів і систем. Вимоги щодо якості та оцінювання систем і програмного продукту (SQuaRE). Настанова щодо SQuaRE (ISO/IEC 25000:2014, IDT). Чинний від 2016-01-01. Вид. офіц. URL: https://www.iso.org/ru/standard/64764.html (дата звернення: 01.12.2023).
6. 1061-1998. IEEE Standard for a Software Quality Metrics Methodology. – Effective from 1998-12-31. – Official edition. – [S. l. : s. n.], 1998.
7. Боем Б. У. Інженерне проектування програмного забезпечення. / Б. У. Боем ; пер. з англ А. А. Красилів. – [Б. м.] : Радіо і зв'язок, 1985. – 512 с.
8. Авраменко А. Тестування програмного забезпечення : навч. посіб. / А. Авраменко, В. Авраменко, Г. Косенюк. – Черкаси : ЧНУ ім. Богд. Хмельн., 2017. – 284 с.
9. Жураковський Б. Ю. ТЕХНОЛОГІЇ ІНТЕРНТУ РЕЧЕЙ : навч. посіб. / Б. Ю. Жураковський, І. О. Зенів. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 271 с.
10. IoT: Source of test challenges / E. J. Marinissen [et al.] // Proceedings of 21st IEEE European Test Symposium (ETS), Amsterdam, 14 December 2016. – [S. l.]. – P. 1–10.
11. How to test iot-based services before deploying them into real world / E. S. Reetz [et al.] // Proceedings of the 19th European Wireless Conference, Guildford, 6 September 2013. – [S. l.]. – P. 1–6.

12. Integrated environment for testing IoT and RFID technologies applied on intelligent transportation system in Brazilian scenarios / A. G. Leal [et al.] // Proceedings of the IEEE Conference on Brazil RFID, Sao Paulo, 13 June 2014. – [S. l.]. – P. 22–24.
13. A distributed test system architecture for open-source IoT software / P. Rosenkranz [et al.] // Proceedings of the Workshop on IoT Challenges in Mobile and Industrial Systems, New York, 13 April 2015. – [S. l.]. – P. 43–48.
14. Коваленко І. О. ПЕРСПЕКТИВНІ ШЛЯХИ РОЗВИТКУ ТЕСТОВИХ СПЕЦИФІКАЦІЙ ІНТЕРНЕТУ РЕЧЕЙ / І. О. Коваленко, І. В. Шаріпова // Інформатика, інформаційні системи та технології: тези доповідей двадцятої всеукраїнської конференції студентів і молодих науковців : тези доп., Одеса, 28 квіт. 2023 р. – Одеса, 2023. – С. 121–123.
15. Коваленко І. О. Проблеми масштабування системи «Розумний будинок» / І. О. Коваленко // Конкурс студентських наукових робіт факультету математики, фізики та інформаційних технологій Одеського національного університету імені І.І. Мечникова, Одеса, 27 берез. 2023 р. – Одеса, 2023.
16. Коваленко І. О. КОМПЛЕКСНА МОДЕЛЬ ТЕСТУВАННЯ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ / І. О. Коваленко, В. С. Михайленко // ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ: МОДЕЛІ, АЛГОРИТМИ, СИСТЕМИ : IV Всеукр. науково-практ. інтернет конф. : тези доп., Миколаїв, 30–31 жовт. 2023 р. – Миколаїв, 2023. – С. 87–89.
17. Коваленко І. О. ГЕНЕРАЦІЯ ТЕСТОВИХ ДАНИХ ЗА ДОПОМОГОЮ МІКРОКОНТРОЛЕРІВ, ЗАДЛЯ ТЕСТУВАННЯ СИСТЕМИ «РОЗУМНИЙ» БУДИНОК / І. О. Коваленко, І. В. Шаріпова // Інформаційні технології та інженерія : Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів : тези доп., Миколаїв, 7–10 лют. 2023 р. – Миколаїв, 2023. – С. 72–73.
18. Kovalenko I. Systematism in detecting errors during testing of hardware and software complexes of the "Internet of Things / Ivan Kovalenko, Ilnara Sharipova // International Scientific and Practical Forum «Digital Reality»

2023: Cyber security and information technologies in the hybrid wars conditions : тези доп., Odesa - Kharkiv, 1–2 March 2023. – Kharkiv, 2023.

19. Bandyopadhyay S. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices [Electronic resource] / S. Bandyopadhyay, A. Bhattacharyya // 2013 International Conference on Computing, Networking and Communications (ICNC 2013), San Diego, CA, 28–31 January 2013. – [S. l.], 2013. – Mode of access: https://doi.org/10.1109/iccnc.2013.6504105 (date of access: 16.12.2023).
20. MQTT V3.1 Protocol Specification. MQTT [Electronic resource]. – Mode of access: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html (date of access: 16.12.2023).
21. Basics of HTTP - HTTP | MDN [Electronic resource] // MDN Web Docs. – Mode of access: https://developer.mozilla.org/ru/docs/Web/HTTP/Basics_of_HTTP (date of access: 16.12.2023).
22. Grigorik I. Making the web faster with HTTP 2.0 [Electronic resource] / Ilya Grigorik // Communications of the ACM. – 2013. – Vol. 56, no. 12. – P. 42–49. – Mode of access: https://doi.org/10.1145/2534706.2534721 (date of access: 16.12.2023).
23. Ghezzi C. Fundamentals of software engineering. Englewood Cliffs, N.J : Prentice-Hall, 1991. 573 с.
24. MQTT V3.1 Protocol Specification. MQTT. URL: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html (дата звернення: 01.12.2023).
25. Performance evaluation of MQTT and CoAP via a common middleware / D. Thangavel та ін. 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), м. Singapore, 21–24 квіт. 2014 р. 2014. URL: https://doi.org/10.1109/issnip.2014.6827678 (дата звернення: 01.12.2023).

ДОДАТОК А

Змінні скетчу

```
//змінна для зберігання даних про температуру
    string temperatureString = "";
    //змінна для зберігання інформації, коли останній раз була опублікована //температура
    unsigned long previousMillis = 0;
    //інтервал між публікаціями даних з датчику
    const long interval = 5000
    //GPIO-контакт, до якого підключений світлодіод
    const int ledPin = 25;
    // Актуальний стан вихідного контакту
    int ledState = LOW;
    // GPIO-контакт, до якого підключений датчик DS18B20:
    const int oneWireBus = 32;
    // робимо так, щоб об’єкт «oneWire» комунікував з іншими OneWire-пристроями:
    OneWire oneWire(oneWireBus);
    // передаємо об’єкт «oneWire» датчику температури:
    DallasTemperature sensors(&oneWire);
```

ДОДАТОК Б

Код функції WiFiEvent()

```
void WiFiEvent(WiFiEvent_t event) {
          Serial.printf("[WiFi-event] event: %d\n", event);
          switch(event) {
               case SYSTEM_EVENT_STA_GOT_IP:
               Serial.println("WiFi connected");
               Serial.println("IP address: ");
               Serial.println(WiFi.localIP());
               connectToMqtt();
               break;
          case SYSTEM_EVENT_STA_DISCONNECTED:
               Serial.println("WiFi lost connection");
               xTimerStop(mqttReconnectTimer, 0);
                    xTimerStart(wifiReconnectTimer, 0);
               break;
          }
     }
```

ДОДАТОК В

Код функції onMqttMessage()

```
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
        String messageTemp;
        for (int i = 0; i < len; i++) {
             messageTemp += (char)payload[i];
        }
        if (strcmp(topic, "esp32/led") == 0) {
             if (ledState == LOW) {
                  ledState = HIGH;
             } else {
                  ledState = LOW;
             }
        digitalWrite(ledPin, ledState);
        }
        Serial.println("Publish received.");
        Serial.print("  message: ");
        Serial.println(messageTemp);
        Serial.print("  topic: ");
        Serial.println(topic);
        Serial.print("  qos: ");
        Serial.println(properties.qos);
        Serial.print("  dup: ");
        Serial.println(properties.dup);
        Serial.print("  retain: ");
        Serial.println(properties.retain);
        Serial.print("  len: ");
        Serial.println(len);
        Serial.print("  index: ");
        Serial.println(index);
        Serial.print("  total: ");
        Serial.println(total);}
```

ДОДАТОК Г

Таймер оновлення даних

```
unsigned long currentMillis = millis();
// кожні X секунд («interval» = 5 секунд)
// у канал «esp32/temperature»
// буде публікуватися MQTT-повідомлення:
if (currentMillis - previousMillis >= interval) {
    // зберігаю у змінну «previousMillis» час,
    // коли в останній раз були опубліковані дані:
    previousMillis = currentMillis;
    // нові температурні дані:
    sensors.requestTemperatures();
    temperatureString = " " + String(sensors.getTempCByIndex(0)) + "C " + String(sensors.getTempFByIndex(0)) + "F";
    Serial.println(temperatureString);
    // піблікую MQTT-повідомлення у каналі «esp32/temperature»
    // з температурою у градусах Цельсія та Фаренгейта:
    uint16_t packetIdPub2 = mqttClient.publish("esp32/temperature", 2, true, temperatureString.c_str());
    Serial.print("Publishing on topic esp32/temperature at QoS 2, packetId: ");
    Serial.println(packetIdPub2);
}
```