МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

РУЖИЦЬКИЙ Дмитро Андрійович

Програмний інтерфейс прикладного програмування для хмари Інтернету речей / Application programming interface for the Internet of Things cloud

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-43
Д. А. Ружицький

Науковий керівник:
к.т.н., доцент, О.Р.Осолінський

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

Тернопіль – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
РУЖИЦЬКИЙ Дмитро Андрійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний інтерфейс прикладного програмування для хмари Інтернету речей / Application programming interface for the Internet of Things cloud
керівник роботи О.Р. Осолінський, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 р. № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
– проаналізувати сучасні технології та протоколи побудови програмних інтерфейсів прикладного програмування (API) для хмарних IoT-систем;
– провести огляд існуючих архітектур IoT-рішень, що підтримують інтеграцію з хмарними сервісами, і проаналізувати їх структурні та функціональні особливості;
– сформулювати технічні вимоги до API хмарної IoT-системи з підтримкою віддаленого управління пристроями, автентифікації користувачів, обробки сенсорних даних та оновлення мікропрограм;
– розробити концепцію інтерфейсів прикладного програмування для хмари інтернету речей;
– реалізувати алгоритмічне забезпечення функціонування програмного інтерфейсу та функцій API;
– реалізувати інтерфейси прикладного програмування для серверів та клієнтів;
– провести тестування та оцінку продуктивності;
5. Перелік графічного матеріалу в роботі:
– архітектура архітектура модуля реалізації програмного інтерфейсу;
– функціональна схема програмного інтерфейсу;

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2025 р.	

Студент_____ Д. А. Ружицький
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ О.Р.Осолінський
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний інтерфейс прикладного програмування для хмари Інтернету речей» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 57 сторінок і містить 34 ілюстрацій, 4 додатки та 30 використаних джерел.

Метою роботи є розробка програмного інтерфейсу прикладного програмування для хмарного середовища Інтернету речей (IoT), що забезпечує реєстрацію пристроїв, автентифікацію користувачів, збір даних сенсорів, їх передавання, зберігання та обробку з використанням сучасних технологій і протоколів обміну даними.

В роботі використано методи системного аналізу, моделювання, проєктування API-інтерфейсів, інтелектуального аналізу даних, а також експериментальні методи тестування продуктивності та стійкості до навантажень у розподіленому середовищі.

Запропоновано архітектуру універсальної IoT-системи з REST- і MQTT-інтерфейсами, яка дозволяє масштабовано керувати пристроями, здійснювати обробку сенсорних даних у режимі реального часу, а також забезпечувати механізми оновлення прошивки пристроїв (OTA) і обробки виняткових ситуацій. Реалізовано Node.js-сервер, MQTT-брокер, базу даних MongoDB, клієнтську частину на React, а також інтеграцію з ESP32-пристроями.

Ключові слова: ІНТЕРНЕТ РЕЧЕЙ, IoT, API, MQTT, REST, ESP32, OTA, ХМАРНА СИСТЕМА, СЕНСОРИ, АВТОРИЗАЦІЯ.

ANNOTATION

Qualification Thesis on the Topic: " Application programming interface for the Internet of Things cloud" submitted for the attainment of the Bachelor's Degree in the specialty 122 "Computer Science" within the educational program "Computer Science" consists of 57 pages and includes 34 illustrations, 4 appendices, and 30 references.

The aim of the thesis is to develop an application programming interface (API) for a cloud-based Internet of Things (IoT) environment, providing functionality for device registration, user authentication, sensor data collection, transmission, storage, and processing using modern technologies and data exchange protocols.

The work employs methods of systems analysis, modeling, API interface design, intelligent data analysis, as well as experimental methods for testing performance and load resistance in a distributed environment.

A universal IoT system architecture is proposed with REST and MQTT interfaces, enabling scalable device management, real-time sensor data processing, firmware over-the-air (OTA) updates, and exception handling mechanisms. The implementation includes a Node.js server, MQTT broker, MongoDB database, client-side interface in React, and integration with ESP32 devices.

Keywords: INTERNET OF THINGS, IoT, API, MQTT, REST, ESP32, OTA, CLOUD SYSTEM, SENSORS, AUTHORIZATION.

ЗМІСТ

ВСТУП

В Інтернеті речей (IoT) тисячі об’єктів або пристроїв можуть бути плавно з’єднані між собою за допомогою зв’язку «пристрій-пристрій» або «машина-машина» [1]; таким чином, вони можуть працювати у співпраці один з одним інтелектуальним способом [2]. Значне збільшення даних, які генеруються пристроями IoT, також сприяє прогресу п’ятого покоління (5G) для IoT [3]. Щоб повністю розкрити потенціал IoT, необхідно використовувати та застосовувати технології хмарних обчислень [4]. Хмара IoT дозволяє пристроям IoT підключатися до хмарних інфраструктур через Інтернет. Завдяки необмеженому сховищу, обчислювальним і мережевим ресурсам хмара IoT пропонує можливість керувати великомасштабними постійними з’єднаннями для IoT [5]. Крім того, хмара може забезпечити аналітику даних у реальному часі, дозволяючи пристроям IoT розвантажувати обчислювальні завдання та величезний обсяг даних [6]. Таким чином, хмара IoT може добре підтримувати різноманітні програми та сервіси [7].

Як правило, клієнти можуть дистанційно користуватися послугами хмари IoT за допомогою інтерфейсів прикладного програмування (API). API можуть полегшити роботу з необхідними функціями або даними для розробників, дозволяючи розробникам використовувати та контролювати хмарні інфраструктури та ресурси IoT. Таким чином можна легко розробляти та розгортати хмарні програми IoT. API визначають методи та формати даних для взаємодії клієнтів і серверів один з одним. Інтерфейсами API керують програми, які працюють на серверах, тоді як програми, які працюють на стороні клієнт/пристрій, повинні мати можливість маніпулювати API. Таким чином, API інтегрують програми на стороні сервера та на стороні клієнт/пристрій. Тим часом API відокремлюють реалізації клієнтів від серверів; таким чином, клієнти, які працюють на різних платформах, можуть користуватися послугами, що надаються серверами. Таким чином, добре спроектовані та належним чином реалізовані API важливі для мінімізації часу розгортання додатків і забезпечення хмарних служб для IoT [8].

Щоб вирішити цю проблему, потрібен новий дизайн та реалізації API для хмари IoT. Обидва протоколи HyperText Transfer Protocol (HTTP) і Message Queuing Telemetry Transport (MQTT) підтримуються хмарою IoT з огляду на різні служби та пристрої. Як наслідок, потрібно два типи API, які відповідають кожному протоколу програми. За таких API послуги на основі HTTP та послуги на основі MQTT можуть якісно надаватися кінцевим користувачам.

Метою кваліфікаційної роботи є створення програмного інтерфейсу прикладного програмування для хмари Інтернету речей, який забезпечує ефективну реєстрацію користувачів і пристроїв, збір даних, керування пристроями та оновлення мікропрограм у реальному часі з використанням протоколів HTTP та MQTT.

Завдання кваліфікаційної роботи полягають у послідовному виконанні наступних етапів:

– проаналізувати сучасні технології та протоколи побудови програмних інтерфейсів прикладного програмування (API) для хмарних IoT-систем;

– провести огляд існуючих архітектур IoT-рішень, що підтримують інтеграцію з хмарними сервісами, і проаналізувати їх структурні та функціональні особливості;

– сформулювати технічні вимоги до API хмарної IoT-системи з підтримкою віддаленого управління пристроями, автентифікації користувачів, обробки сенсорних даних та оновлення мікропрограм;

– розробити концепцію інтерфейсів прикладного програмування для хмари інтернету речей;

– реалізувати алгоритмічне забезпечення функціонування програмного інтерфейсу та функцій API;

– реалізувати інтерфейси прикладного програмування для серверів та клієнтів;

– провести тестування та оцінку продуктивності;

Об’єкт дослідження - процес обміну інформацією між IoT-пристроями та хмарною платформою з врахуванням обмежень ресурсів і вимог до реального часу.

Предмет дослідження - методи та засоби проектування і реалізації прикладного програмного інтерфейсу для хмарної IoT-системи.

Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах студентської науково-практичної конференції “Інтелектуальні інформаційні технології в прикладних дослідженнях” (IITAR – 2025), м. Тернопіль, Україна, 27-29 травня 2025 р.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1. АНАЛІЗ ТЕХНОЛОГІЙ ПОБУДОВИ ІНТЕРФЕЙСІВ ПРИКЛАДНОГО ПРОГРАМУВАННЯ ДЛЯ ХМАРИ ІНТЕРНЕТУ РЕЧЕЙ

1.1 Опис технологій побудови інтерфейсів для хмари IoT

Інтернет речей (Internet of Things, IoT) відіграє надзвичайно важливу роль у сучасному цифровому світі. Завдяки цій технології можливим стало поєднання фізичних об’єктів з цифровими платформами, що відкрило нові горизонти для автоматизації, моніторингу та управління системами різного призначення. Одним із ключових елементів цього процесу є хмара Інтернету речей, яка об’єднує пристрої, сервіси, користувачів та програмне забезпечення в єдине інтегроване середовище. Програмний інтерфейс прикладного програмування, або API, слугує центральною ланкою в цій взаємодії, забезпечуючи зручний доступ до даних та функціональності пристроїв, які підключені до хмари.

Сучасні IoT-системи зазвичай складаються з декількох основних компонентів. До них належать сенсори й актуатори, які виконують роль джерел даних або виконавчих пристроїв. Інформація, зібрана з різних сенсорів, передається через мережеву інфраструктуру до хмарної платформи, де здійснюється її обробка, зберігання та аналіз. Саме на цьому етапі API забезпечує взаємодію між пристроями, хмарою та прикладними додатками користувачів, що уможливлює створення гнучких, масштабованих та ефективних рішень у багатьох галузях, включаючи промисловість, медицину, сільське господарство та розумні міста.

В основі хмарної архітектури IoT лежить концепція централізованого управління даними та ресурсами. Наприклад, в одному з досліджень [9] була запропонована система цифрового двійника виробничої клітини, яка використовує IoT, SCADA, HMI та хмарні обчислення для управління й оптимізації виробничих процесів у режимі реального часу. У цій моделі цифровий двійник є віртуальним відображенням фізичного об’єкта, що синхронізується з ним за допомогою API, який забезпечує двосторонній обмін даними між хмарою й фізичною системою. Це дає змогу вчасно реагувати на відхилення у роботі, здійснювати аналітику та прогнозування, а також забезпечити ефективне планування виробництва.

Програмний інтерфейс прикладного програмування, або API, виступає як універсальний засіб доступу до сервісів хмарної платформи. Завдяки API користувачі можуть отримувати дані з сенсорів, керувати пристроями, налаштовувати параметри системи, підключати нові компоненти та реалізовувати аналітичні задачі. Одним із найважливіших аспектів є можливість стандартизації взаємодії, що забезпечує сумісність різних пристроїв і систем. Наприклад, у дослідженні [10] описано використання ThingSpeak API для збору та обробки даних з сенсорів MQ135 та MQ-7, які вимірюють якість повітря в приміщенні. Завдяки контролеру ESP8266, що підтримує Wi-Fi, дані надсилаються до хмари, де доступ до них здійснюється через RESTful API.

З технічної точки зору, існує кілька підходів до реалізації API у контексті хмарного IoT. REST API є найпоширенішим варіантом завдяки своїй простоті, гнучкості та сумісності з HTTP-протоколом. У такій архітектурі використовуються стандартні методи запитів, зокрема GET, POST, PUT і DELETE. Іншим підходом є використання MQTT API — легковагового протоколу публікації-підписки, який ідеально підходить для IoT-пристроїв із обмеженими ресурсами. Цей протокол забезпечує мінімальне споживання енергії, низьку затримку передачі даних та надійність з’єднання. Ще одним сучасним методом є WebSocket API, який дозволяє підтримувати постійне з’єднання між клієнтом і сервером, що надзвичайно важливо для систем реального часу.

З огляду на розвиток складних інтерфейсів користувача, набуває популярності GraphQL API, який дозволяє клієнту самостійно визначати, які саме дані він хоче отримати від сервера. Це зменшує обсяг переданої інформації та підвищує ефективність взаємодії. У промислових системах часто використовується OPC UA API, що забезпечує стандартизовану взаємодію між промисловими пристроями, SCADA-системами та хмарними сервісами.

Безпека взаємодії через API в хмарних IoT-системах є критично важливим фактором. Із розвитком технологій зростає кількість потенційних загроз, тому необхідно забезпечити надійний захист на всіх рівнях системи. Одним із підходів є реалізація концепції Zero Trust, яка передбачає повну недовіру до будь-яких

підключень, доки вони не пройдуть автентифікацію та авторизацію. У цьому контексті використовуються такі інструменти як токени доступу, протоколи OAuth 2.0, SSL/TLS-шифрування, контроль ролей (RBAC), а також моніторинг та журналювання запитів до API. У дослідженні [11] розглянуто багаторівневу архітектуру з мікросегментацією, що забезпечує захищене середовище для передачі даних між IoT-пристроями, хмарою та користувачами.

Реальні приклади реалізації API у хмарних IoT-системах ілюструють широкі можливості цього підходу. Наприклад, система моніторингу якості повітря, описана [12], дозволяє в режимі реального часу зчитувати показники газових сенсорів та передавати їх до хмари, де вони обробляються й виводяться на інформаційну панель. У роботі [13] описується використання ESP32 як веб-сервера для управління роботизованим маніпулятором через браузерний інтерфейс, де API забезпечує обробку запитів від користувача й передачу команд на сервоприводи. Ще один приклад — телемедична система [14], де дані про стан здоров’я пацієнтів збираються за допомогою сенсорів і передаються через API до хмарного середовища, де аналізуються за допомогою нечеткої логіки.

Незважаючи на значний прогрес, існують певні виклики у розробці й використанні API для хмарних IoT-систем. Однією з основних проблем є гетерогенність пристроїв, що використовують різні протоколи, формати даних і можливості обробки. Це ускладнює стандартизацію API й потребує створення адаптивних рішень. Також варто враховувати затримки у передачі даних, особливо у системах, де потрібна обробка в реальному часі. Масштабованість API також є проблемою, адже із збільшенням кількості підключених пристроїв зростає навантаження на сервери й комунікаційні канали.

Ще одним важливим аспектом є інтероперабельність, тобто здатність API взаємодіяти з іншими сервісами, платформами та протоколами. Для цього необхідно забезпечити підтримку таких форматів, як JSON, XML, CBOR, та таких протоколів, як CoAP, HTTP, MQTT. У цьому контексті дедалі більшого значення набувають API-шлюзи (API Gateways), які забезпечують централізоване

управління запитами, кешування, обмеження швидкості, автентифікацію та моніторинг усього API-трафіку.

У майбутньому варто очікувати подальший розвиток інтелектуальних API, які використовуватимуть штучний інтелект для адаптації відповідей, автоматичного виявлення аномалій, оптимізації маршрутизації запитів та прогнозування навантаження. Також зростатиме роль Edge API, які дозволяють переносити частину обчислень ближче до периферії мережі, зменшуючи затримки й навантаження на хмару. Такий підхід сприятиме створенню ще більш ефективних і гнучких рішень у сфері Інтернету речей.

У підсумку, програмні інтерфейси прикладного програмування є невід'ємною складовою хмарних IoT-систем. Вони забезпечують стандартизовану, безпечну та ефективну взаємодію між пристроями, хмарними сервісами та прикладними програмами. Правильна архітектура API є ключовим фактором успіху в реалізації масштабованих та інтероперабельних IoT-рішень. У контексті сучасних викликів і тенденцій розвитку технологій, API продовжують відігравати провідну роль у забезпеченні зв'язку між фізичним і цифровим світами, роблячи Інтернет речей справді розумним і ефективним інструментом трансформації суспільства.

1.2 Огляд і аналіз існуючих рішень

У сучасному світі Інтернет речей (IoT) перетворюється на одну з ключових технологій цифрової трансформації в різних галузях – від охорони здоров'я до промисловості та сільського господарства. Основою взаємодії в IoT-середовищах є програмні інтерфейси прикладного програмування (API), які забезпечують зв'язок між пристроями, хмарними сервісами та користувацькими додатками. Саме API визначає, яким чином дані передаються, обробляються та виводяться у вигляді аналітичних результатів чи дій у реальному часі. У цьому контексті важливим завданням є аналіз існуючих рішень щодо реалізації API у хмарних IoT-системах.

Однією з провідних галузей, де застосування хмарних API продемонструвало значну ефективність, є медична сфера. Наприклад, у проєкті [15] реалізовано RESTful API з підключенням до платформ ThingSpeak та Particle Cloud. Така система дозволяє здійснювати безперервний моніторинг життєвих показників пацієнтів і передавати дані у хмару для подальшого аналізу лікарем. Подібні технології також використовуються у системах на базі штучного інтелекту для віддаленого моніторингу стану здоров'я, як це показано у роботі [16]. Там API забезпечує передачу медичних даних, які потім обробляються в хмарі алгоритмами глибокого навчання для виявлення ризиків.

Ще більш розгалуженою є система [17] де API об'єднує дані з носимих пристроїв, мобільних додатків та електронної медичної документації. Така мультиджерельна модель вимагає наявності надійного API-шару з підтримкою як REST, так і більш сучасних форматів, зокрема GraphQL.

У сільському господарстві хмарні API використовуються для моніторингу довкілля, стану ґрунтів, вологості, температури тощо. Так, у дослідженні IoT [18] система сенсорів збирає дані, які через REST API передаються у хмару для візуалізації на панелі управління. Подібна реалізація є типовою для проєктів з використанням легковагових IoT-плат у польових умовах. В іншому прикладі, де використовувалась система злиття даних від кількох сенсорів, API також відповідає за уніфікацію даних і відправлення їх на аналітичні сервіси AWS.

Міське середовище теж активно застосовує IoT Cloud API, зокрема у проектах «розумного міста». Наприклад, у роботі, присвяченій інфраструктурі API для Smart City, реалізовано єдину інтеграційну платформу з підтримкою OpenAPI 3.0. Це забезпечує можливість швидкого підключення нових сенсорів, сервісів та зовнішніх додатків через стандартизовані API-виклики. Такі архітектури дозволяють інтегрувати розумне освітлення, екологічні датчики, управління трафіком у межах єдиної цифрової системи міста.

Інший приклад стосується використання API у дронах для агрономічного моніторингу або екстрених служб. Тут API керує маршрутом дрона, збирає відео- та фотодані, які передаються через MQTT у хмару. Легковагові протоколи, такі як

MQTT та CoAP, є доцільними у таких рішеннях, оскільки забезпечують мінімальне енергоспоживання та швидку передачу.

У промисловості API виконує роль мосту між фізичними пристроями (PLC, датчиками) та віртуальними системами контролю. Наприклад, у рішенні PLC-based IoT Cloud Interface використовується REST API для передачі даних між контролером і хмарною платформою, що дозволяє реалізувати концепцію Digital Twin. Такі рішення забезпечують візуалізацію та управління виробничими процесами в режимі реального часу. В роботі [19] застосовуються бездротові сенсори та REST API, що передають дані у хмару, де відбувається попередження про знос обладнання.

Варто також розглянути технологічний аспект реалізації API. Найпоширенішим стилем проєктування API є REST, завдяки своїй простоті та повній сумісності з HTTP. Майже всі проаналізовані системи використовують REST API для стандартної передачі даних. Водночас, для ресурсно обмежених пристроїв перевага надається протоколам MQTT та CoAP. Наприклад, у рішенні [20] реалізовано передачу даних через MQTT, що дає змогу заощаджувати енергію та знижує затримки.

Ще одним важливим питанням є вибір хмарної платформи. Найчастіше використовуються AWS, Azure, Blynk, Google Cloud, ThingSpeak. Наприклад, AWS широко застосовується в проектах для створення API-шлюзів, зберігання даних (S3) та запуску обчислювальних функцій через AWS Lambda. У проєкті з моніторингу систем осмосу [21] дані з мікроконтролера надходять на дашборд через API, розгорнутий в AWS.

Архітектурно API може бути реалізовано або у вигляді централізованої служби у хмарі, або у вигляді гібридної моделі з edge-компонентом. У проєкті Edge-Cloud IoT API Implementation частина API реалізована на периферійних пристроях, що дозволяє обробляти дані локально навіть за відсутності з’єднання з хмарою. Це особливо актуально для систем із високими вимогами до надійності та швидкості.

Однією з найактуальніших тем є безпека API. Сучасні системи впроваджують Zero Trust-архітектуру, де кожен API-виклик вимагає окремої автентифікації. У

роботі, [22] присвяченій Zero Trust API, впроваджено мікросегментацію та використано OpenID Connect. Інші системи використовують блокчейн для реєстрації API-викликів, що дозволяє забезпечити достовірність даних, які проходять через API-шлюзи.

Підсумовуючи, можна сказати, що програмні інтерфейси прикладного програмування є критично важливим компонентом будь-якої IoT Cloud-системи. Їх реалізація визначає не лише ефективність обміну даними, а й масштабованість, безпеку, надійність та інтеграційний потенціал рішень. Актуальні проєкти демонструють різноманітність архітектурних підходів, протоколів та платформ. У перспективі очікується посилення автоматизації у генерації API, поширення AI-інтегрованих API та розвиток low-code API платформ.

1.3 Постановка задачі

У сучасних умовах швидкого розвитку Інтернету речей (IoT) зростає потреба у створенні масштабованих, безпечних та ефективних інтерфейсів прикладного програмування (API), що забезпечують взаємодію між фізичними пристроями, хмарними сервісами та користувацькими застосунками. Системи IoT генерують великий обсяг даних у реальному часі, який має бути зібраний, оброблений та доставлений до відповідних сервісів з мінімальними затримками. Це вимагає уніфікованих, стандартизованих підходів до побудови API, які підтримують різні протоколи взаємодії, такі як HTTP та MQTT, з урахуванням обмежених ресурсів пристроїв та необхідності в безперервному з'єднанні.

Попри наявність ряду готових рішень, більшість існуючих API не враховують повною мірою потребу в адаптивності, безпеці, гнучкості та простоті інтеграції з різними типами пристроїв і платформ. Важливо забезпечити ефективне керування пристроями, реєстрацію, оновлення мікропрограм, передачу даних і зворотній зв’язок, реалізуючи при цьому механізми автентифікації, авторизації та контролю доступу.

Таким чином, актуальною є задача розробки структури та функціональних можливостей API для хмарної IoT-системи, що відповідатиме сучасним вимогам до надійності, масштабованості та взаємодії з обмеженими пристроями.

Для досягнення цієї мети необхідно виконати наступні завдання:

1. Сформулювати технічні вимоги до API хмарної IoT-системи з підтримкою віддаленого управління пристроями, автентифікації користувачів, обробки сенсорних даних та оновлення мікропрограм.

2. Розробити концепцію інтерфейсів прикладного програмування для хмари інтернету речей.

3. Реалізувати алгоритмічне забезпечення функціонування програмного інтерфейсу та функцій API.

4. Реалізувати інтерфейси прикладного програмування для серверів та клієнтів.

5. Провести тестування та оцінку продуктивності.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ПРОГРАМНОГО ІНТЕРФЕЙСУ ПРИКЛАДНОГО ПРОГРАМУВАННЯ ДЛЯ ХМАРИ ІОТ

2.1 Розробка концепції інтерфейсів прикладного програмування для хмари Інтернету речей

Хмара IoT може пропонувати різноманітні послуги, які допомагають розробникам програмного забезпечення розробляти та запускати програми на основі хмарної інфраструктури IoT або постачальникам/виробникам оригінального обладнання (OEM) для виробництва пристроїв, які мають доступ до хмари IoT. Ці служби розгортаються на двох типах серверів додатків у хмарі IoT, тобто на серверах HTTP(S) та серверах MQTT. Кожна служба містить кілька різних функцій, як показано на рисунку 2.1.

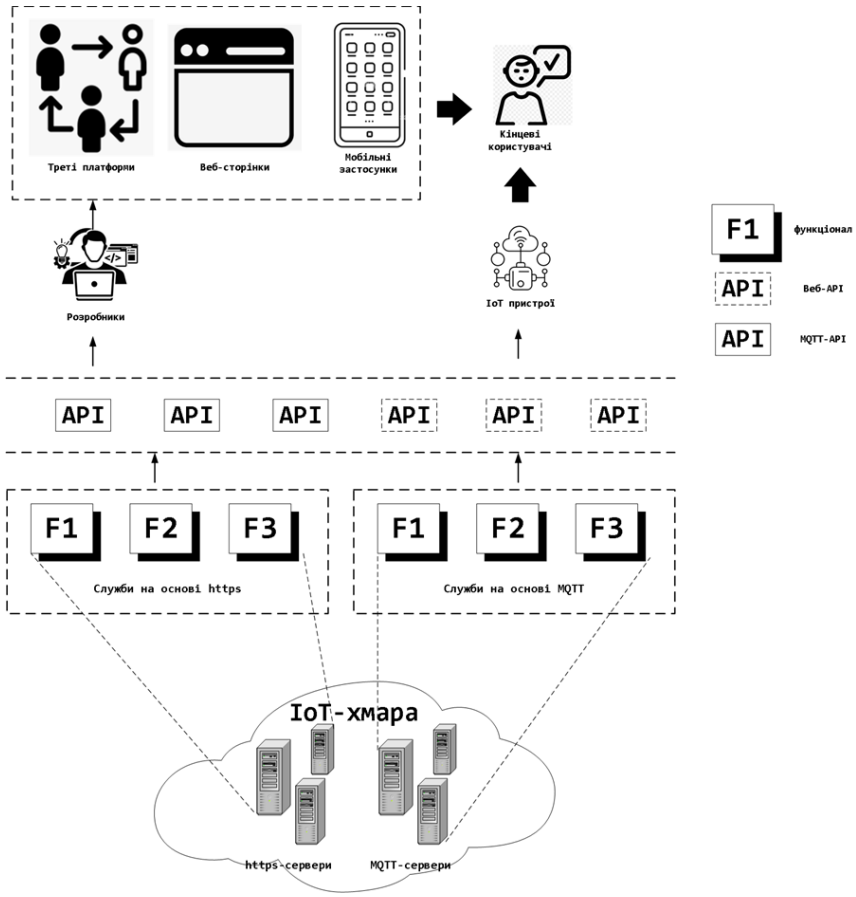


Рисунок 2.1 - Типи сервісів і дизайн API в хмарі IoT. API, інтерфейс прикладного програмування; IoT, Інтернет речей; HTTPS, протокол передачі гіпертексту; MQTT, телеметричний транспорт у черзі повідомлень

Сервіси, які пропонує хмара IoT, можна розділити на дві категорії, тобто послуги на основі HTTPS та послуги на основі MQTT, відповідно до різних протоколів додатків, які вони прийняли. Надсилаючи відповідні запити до серверів, клієнти можуть використовувати ці API.

Служби на основі HTTP надаються клієнтам для обміну гіпертекстами між програмами клієнтів і серверів через Інтернет. Ці служби дозволяють клієнтам і пристроям реєструватися, входити в хмару IoT або виходити з неї за допомогою відповідних веб-API перед подальшими операціями.

Крім того, клієнти повинні прив’язувати пристрої за допомогою прив’язки пристроїв, перш ніж вони зможуть підключитися до них. Тим часом вони також можуть випускати пристрої, щоб звільнити відносини з пристроями. Що стосується пристроїв, вони повинні використовувати надання пристрою для доступу до серверів MQTT.

Крім того, бездротовий зв’язок для пристроїв для оновлення мікропрограми надається як веб-API. За допомогою бездротового зв’язку пристрої можуть надсилати запити на HTTPS-сервери для останньої версії мікропрограми. Після цього сервери можуть відповісти пристроям із номером версії та уніфікованим покажчиком ресурсу (URL), на який можна завантажити мікропрограмне забезпечення; таким чином, пристрої можуть зручно оновлювати мікропрограму. Конкретний дизайн веб-API показано в таблиці 2.1.

Веб-API вимагають, щоб повідомлення запиту складалося з чотирьох частин, тобто запиту, заголовка HTTP, порожнього рядка та повідомлень (необов’язково). Приклад запиту через реєстр користувачів показано наступним чином.

Метод HTTP, URL-адреса та номер версії утворюють поле запиту повідомлення запиту. Серед усіх методів запиту, які визначені в HTTPS, лише GET, POST і DELETE використовуються в хмарі IoT, оскільки ці три методи можуть задовольнити більшість вимог до операцій.

Таблиця 2.1 – Приклади веб-API, розроблених для хмари IoT

Назва API	Метод HTTP	Кінцева точка (URL)	Параметри	Роль	Код статусу	Значення відповіді
Реєстрація користувача	POST	user/register	Ім’я користувача та пароль	Реєстрація клієнта	1000/1001	UID токен
Реєстрація пристрою	POST	device/register	Ключ продукту, MAC-адреса	Клієнт реєструє пристрій	201/400	DID
Вхід користувача	POST	user/login	Ім’я користувача та пароль	Клієнт вхід	5000/5001	UID
Перелік пристроїв	GET	list/device	UID	Вивід усіх пристроїв, які належать UID	8000/8001	Список пристроїв
Інформація про пристрій	GET	/p/device/<DID>	UID і DID	Отримати інформацію про пристрій	6000/6001	---
Вив’язування пристрою	DELETE	unbind	UID і DID	Клієнт вив’язує пристрій	7000/7001	---
Встановлення таймера	POST	timer	Таймер, дата і час	Налаштування таймера	9000/9001	---
Надання пристрою	GET	device/provision	DID	Отримання адреси MQTT-сервера	201/400	Адреса MQTT сервера
OTA (оновлення)	GET	device/ota/<DID>/update	UID і код доступу	OTA для пристрою	201/400	Версія оновлення

Метод GET дозволяє клієнтам отримувати інформацію або дані з серверів, тоді як POST (рисунок 2.2) дозволяє клієнтам надсилати дані на сервери.

```
POST /server_address/user_register HTTP/1.1
Content-Type: Application/JSON
Connection: Keep-alive
Host: localhost

{
  "username": "123456",
  "password": "123456"
}
```

Рисунок 2.2 – POST запит

Метод DELETE може видалити ресурс, указаний клієнтами. URL-адреса допомагає клієнтам знайти ресурс, оскільки вона показує розташування ресурсу. Щоб підтримувати REST, кожен ресурс або дані повинні мати унікальну URL-адресу в Інтернеті. URL-адреса складається з імені домену та кінцевої точки API. Розробка належних URL-адрес є однією з головних частин розробки API, оскільки вона може відображати ієрархічну структуру даних, які можуть запропонувати API.

HTTP-заголовок (рисунок 2.3) містить деякі параметри, які показують використаний формат даних і тип з’єднання. Деякі інші параметри можна додати в це поле; наприклад, клієнти можуть вказати серверам, який формат даних доступний для повідомлення відповіді. Додаткове поле повідомлень містить інформацію, яку потрібно доставити на сервери. Зазвичай вони додаються в кінці запиту.

```
StatusCode:1000

{
  "UID": "1",
  "token": "abcdefg",
  "expire_at": "133288820"
}
```

Рисунок 2.3 - HTTP-заголовок

HTTP-сервери повинні надсилати відповідь клієнтам на кожен запит. Повідомлення відповіді містить код стану та дані (необов’язково). За допомогою коду статусу клієнти можуть бути проінформовані про статус запиту.

До відповіді можуть бути додані необов'язкові дані за умови, що це потрібно клієнтам. Приклад повідомлення відповіді показано таким чином, тобто різним веб-API призначаються різні коди стану; деякі з них попередньо визначені в HTTP [13]; інші спеціально розроблені для хмари IoT. Ці коди можуть допомогти клієнтам легко проаналізувати відповіді. Остання цифра коду стану вказує на те, успішні чи невдалі запити. Цифра «0» означає успіх, а інші цифри — невдачу. Якщо запит не виконується, клієнти можуть визначити тип помилки за останнім номером; наприклад, «1001» означає, що необхідна інформація в запитах є неповною, тоді як «1002» означає, що на HTTP/S-серверах щось не так. Попередньо визначені коди стану зазвичай використовуються для пристроїв для розширення. Код 200 означає, що запити оброблено успішно, тоді як код 201 означає, що новий пристрій створено. Крім того, код 400 означає, що запити від пристроїв незрозумілі для серверів

Для підтримки пристроїв IoT з обмеженими ресурсами зазвичай пропонуються служби на основі MQTT для зв’язку з цими пристроями через протокол MQTT. Сервери MQTT дозволяють кінцевим користувачам надсилати команди на пристрої з обмеженими ресурсами, у той час як ці пристрої можуть повідомляти про свій статус серверам. API, які виконують функції служб на основі MQTT, називаються API MQTT.

На відміну від веб-API, MQTT API викликаються через надсилання та отримання контрольних пакетів між MQTT-серверами та клієнтами/пристроями. Кілька контрольних пакетів визначено в протоколі MQTT, наприклад CONNECT, CONNACK або SUBSCRIBE. MQTT API повторно інкапсулюють контрольні пакети та пропонують більш дружні інтерфейси для споживачів. Конструкція API MQTT показана в таблиці 2.2, і також відповідні зв’язки з контрольними пакетами.

Таблиця 2.2 – Розробка MQTT API для хмари IoT

Типи MQTT API	Параметри	Запит керуючого пакету	Відповідь керуючого пакету	Роль
connect	UID/DID, пароль/код доступу	CONNECT	CONNACK	Клієнти або пристрої створюють з'єднання з MQTT-сервером
subscribe	Топік	SUBSCRIBE	SUBACK	Клієнти або пристрої підписуються на теми
unsubscribe	Топік	UNSUBSCRIBE	UNSUBACK	Користувачі або пристрої відписуються від тем
publish	Топік, повідомлення	PUBLISH	PUBACK	Клієнти або пристрої публікують повідомлення
OTA	DID	—	—	Пристрої оновлюють мікропрограму (firmware)
ping	—	PINGREQ	PINGRESP	Виявлення активності (heartbeat detection)
disconnect	—	DISCONNECT	—	Від'єднання від MQTT-серверів

Клієнти або пристрої можуть встановлювати з’єднання з серверами MQTT через підключення, яке, по суті, надсилає пакети CONNECT на сервер MQTT. Після отримання пакетів CONNECT сервери MQTT можуть надіслати пакети CONNACK, щоб повідомити клієнтів або пристрої про те, що з’єднання налаштовано. Оскільки цей шаблон підписки–публікації прийнятий на серверах MQTT, клієнтам або пристроям потрібно підписатися на теми, які їх цікавлять, перш ніж отримувати повідомлення. Подібним чином сервери MQTT надсилають

їм пакети SUBACK як підтвердження успішної підписки. Клієнти або пристрої також можуть публікувати повідомлення на певні теми; таким чином інші, хто підписався на ці теми, можуть отримувати повідомлення. OTA також включено в API MQTT, що дозволяє пристроям IoT робити гнучкі вибори, або через сервер HTTP, або сервер MQTT, під час оновлення.

З іншого боку, сервери MQTT повинні підтримувати довгі з'єднання з клієнтами та пристроями; тому їм потрібно періодично контролювати стан з'єднань. Цей процес розцінюється як виявлення серцебиття. Типовий MQTT API призначений для виявлення серцевих скорочень, тобто ping. За допомогою ping клієнти та пристрої можуть повідомляти про поточний стан підключень до серверів MQTT. Отримавши повідомлення PINGREQ, сервери MQTT можуть знати, що з'єднання активні.

2.2 Алгоритмічне забезпечення функціонування програмного інтерфейсу

Функціонування IoT-системи з використанням хмарних сервісів передбачає низку послідовних етапів, що забезпечують її повноцінну роботу. На першому етапі йде ініціалізація пристроїв. На цьому етапі виконується запуск сенсорів, перевірка їх працездатності та підготовка до обміну даними. Система переконується, що всі пристрої правильно з'єднані й готові до подальших операцій.

Наступним етапом є реєстрація, тобто користувачі, а також фізичні пристрої, мають бути зареєстровані в системі через HTTP API. При цьому здійснюється генерація унікальних ідентифікаторів (UID для користувача, DID для пристрою), які надалі використовуються для взаємодії з сервером. Після реєстрації пристрій отримує адресу MQTT-сервера, необхідну для встановлення постійного з'єднання.

Після отримання параметрів з'єднання відбувається підключення до MQTT. Цей етап передбачає надсилання пакету CONNECT на сервер та отримання підтвердження у вигляді CONNACK. Як тільки з'єднання встановлено, пристрій переходить у режим моніторингу.

В режимі моніторингу система збирає дані з датчиків, формує повідомлення й передає їх до хмари через MQTT-протокол. Також пристрій підписується на відповідні топіки (тематики), щоб отримувати керувальні команди або інші повідомлення. Це дозволяє не тільки передавати інформацію, але й реагувати на зміну параметрів або віддалені команди (рисунок 2.4).

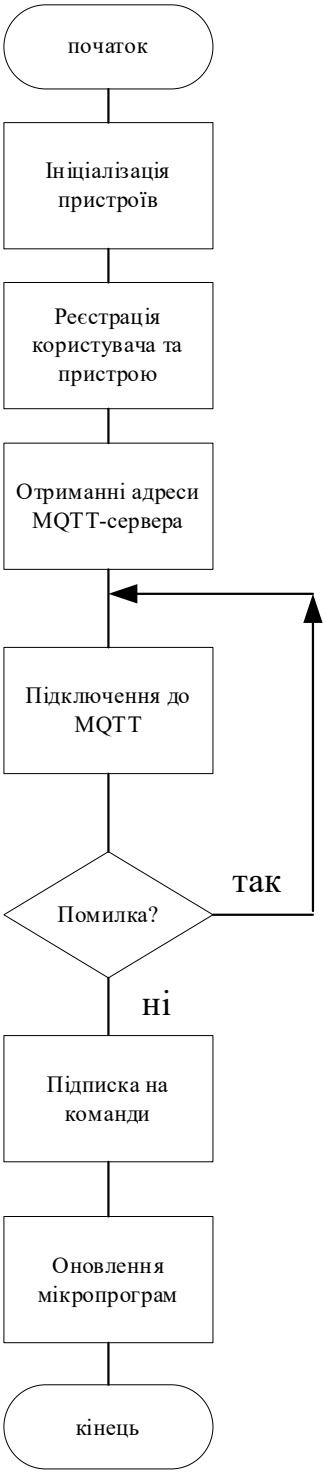


Рисунок 2.4 - Узагальнений алгоритм функціонування програмного інтерфейсу

Одночасно відбувається контроль версії програмного забезпечення. Якщо сервер виявляє наявність оновлення, система ініціює процедуру оновлення мікропрограми. Це передбачає запит на сервер для отримання URL з оновленням, завантаження нового програмного модуля та його встановлення (рисунок 2.5).

Рисунок 2.5 - Узагальнений алгоритм контролю версії програмного забезпечення

Після завершення основних функціональних завдань або в разі виникнення помилок система переходить до фази роз’єднання. Відправляється запит DISCONNECT, і пристрій відключається від MQTT-сервера, завершуючи цикл своєї роботи.

2.3 Алгоритмічне забезпечення функцій API

Алгоритм оновлення прошивки пристрою (OTA — over-the-air) забезпечує віддалене оновлення програмного забезпечення без фізичного втручання (рисунок 2.6). В процесі роботи пристрій періодично перевіряє, чи є нова версія прошивки. Це може відбуватися або за ініціативою користувача, або автоматично через заданий інтервал часу.

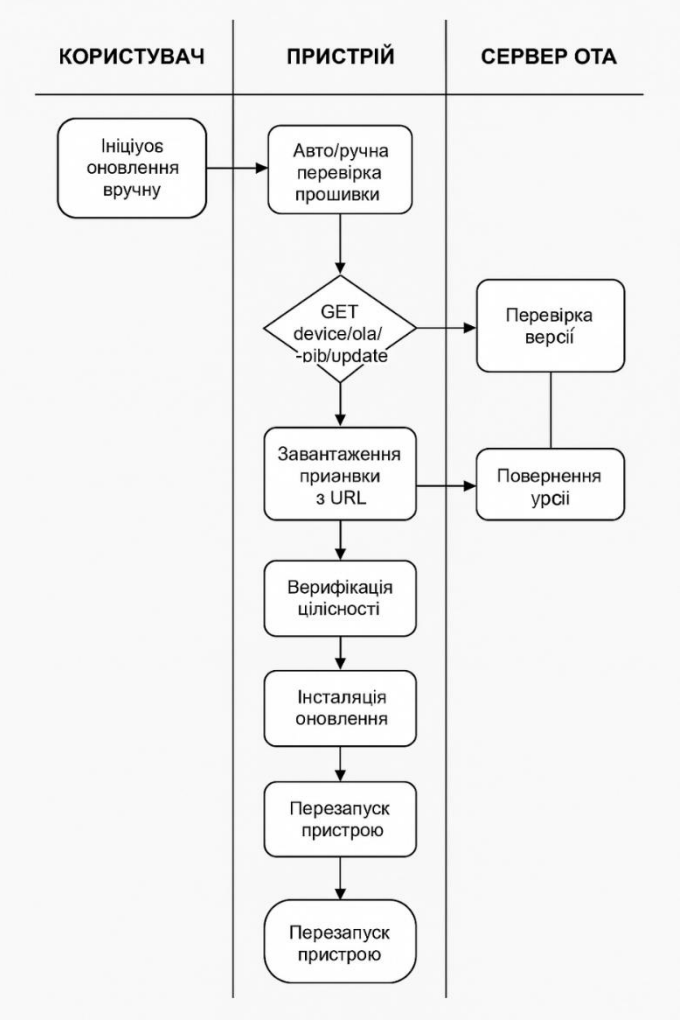


Рисунок 2.6 - Діаграма діяльності алгоритму оновлення прошивки

На першому кроці система виконує перевірку актуальності встановленої мікропрограми. Якщо виявлено, що поточна версія застаріла, пристрій надсилає GET-запит на вказаний endpoint — наприклад, device/ota/<DID>/update, де <DID> — це унікальний ідентифікатор пристрою. Сервер аналізує запит і, у разі успішної обробки, повертає URL-адресу, за якою можна завантажити нову прошивку.

Далі пристрій переходить до завантаження файлу з оновленням. Цей етап критичний, оскільки вимагає стабільного з'єднання для уникнення пошкодження прошивки. Після завершення завантаження розпочинається процес інсталяції нової версії. Успішне завершення оновлення супроводжується перезапуском пристрою для застосування змін.

Оцінка точності та достовірності обчислень (рисунок 2.7) у системі базується на контролі якості взаємодії між пристроями та сервером через API.

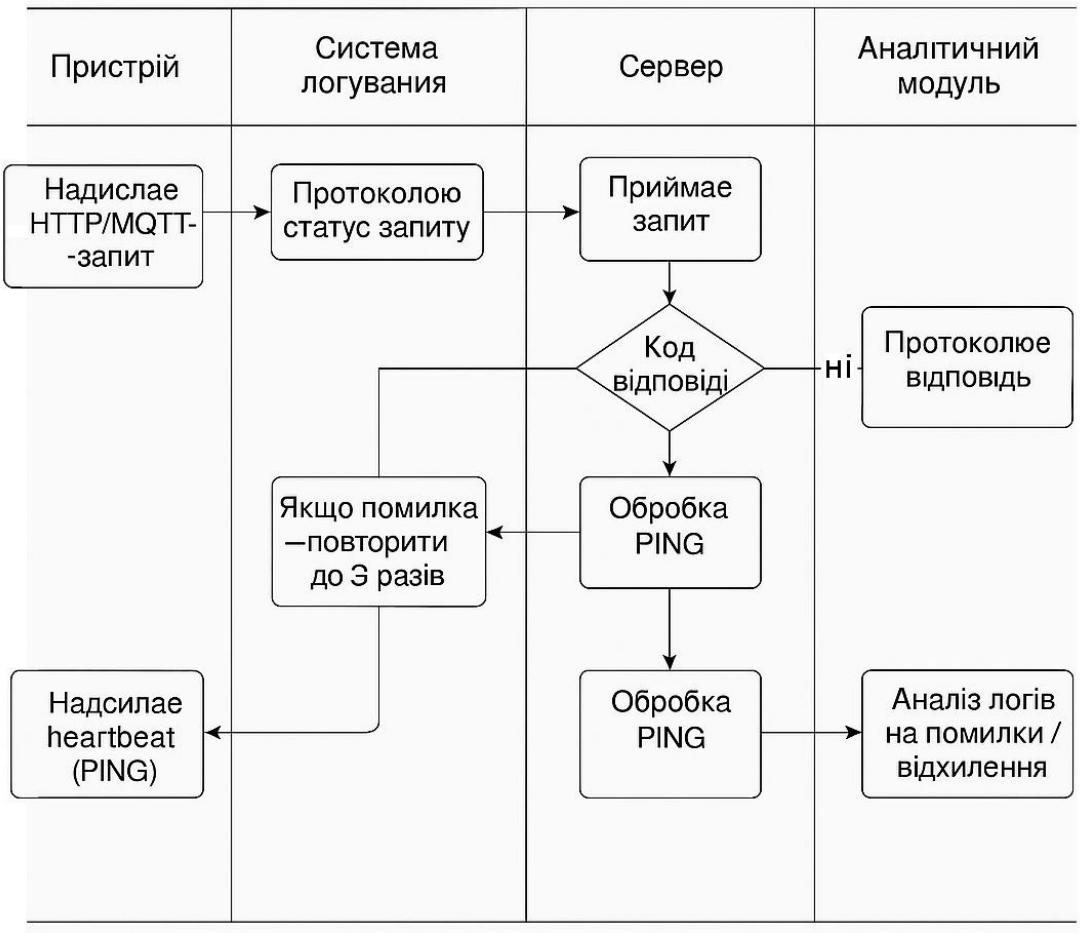


Рисунок 2.7 - Діаграма діяльності алгоритму оцінки точності та достовірності обчислень

Для цього реалізовано кілька ключових механізмів. По-перше, кожен запит до API супроводжується протоколюванням — зберігаються статуси запитів та отриманих відповідей. Це дозволяє виявляти помилки у взаємодії та вчасно реагувати на збої. Наприклад, якщо сервер повертає код 400 або 1001, система фіксує ці події для подальшого аналізу.

В разі виникнення помилки під час надсилання HTTP- або MQTT-запиту реалізовано механізм повторної спроби. Система автоматично повторює запит до трьох разів, що підвищує надійність зв’язку. Це особливо важливо у випадках нестабільного з’єднання або тимчасової недоступності сервера.

Ще одним критичним компонентом є механізм синхронізації стану пристрою з хмарним середовищем. Для цього використовуються heartbeat-запити (типу PING), які надсилаються від пристрою до сервера через певні інтервали часу. Якщо відповідь не отримується, система може вважати з’єднання неактивним і ініціювати повторне підключення або сповіщення про помилку.

В процесі роботи можуть виникати виняткові ситуації, які потребують спеціальної обробки (рисунок 2.8) . Усі такі випадки передбачені в алгоритмах і мають відповідні механізми реагування, що забезпечує стабільність та надійність роботи.

Перший тип виняткової ситуації це відсутність доступу до сервера. Якщо пристрій не може встановити з'єднання з HTTP- або MQTT-сервером, генерується повідомлення про помилку. Це повідомлення може відображатися у вигляді коду статусу, лог-запису або повідомлення в інтерфейсі користувача. Додатково реалізується логіка повторного з’єднання із заданим таймером затримки.

В разі, коли UID користувача або DID пристрою не є дійсними, сервер відмовляє в авторизації. Це означає, що пристрій не буде допущений до подальших дій, таких як публікація даних чи отримання команд. У цьому випадку може ініціюватись повторну авторизацію або запит на перевірку ідентифікаторів.

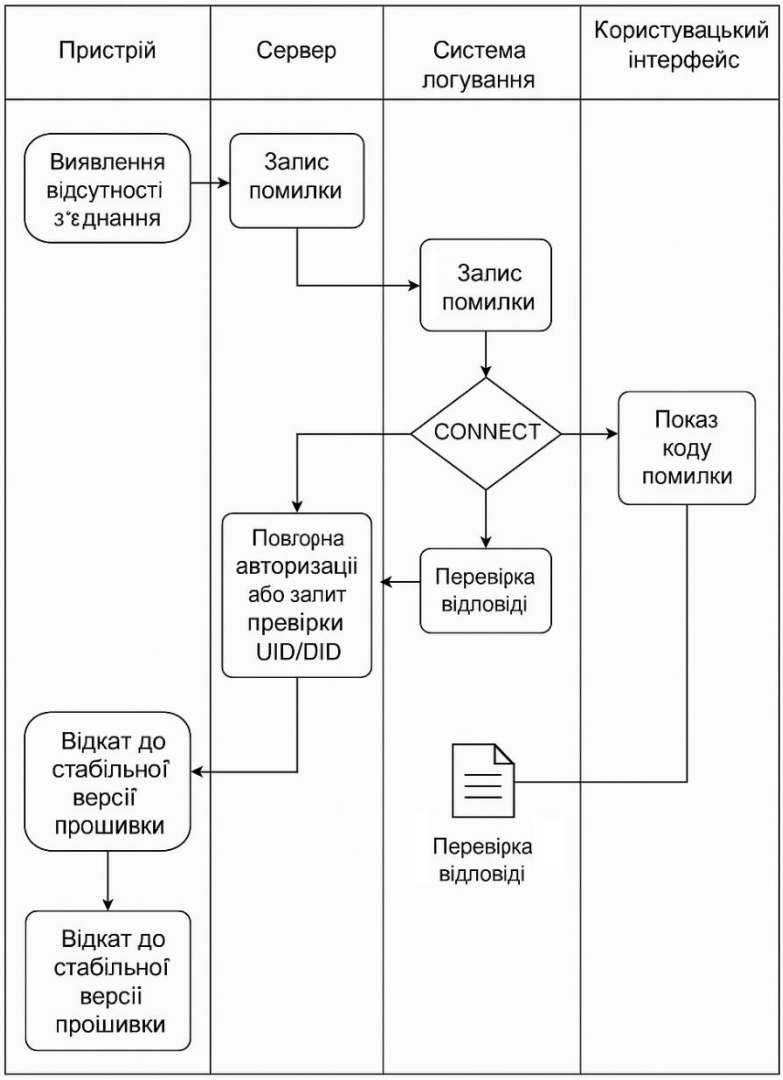


Рисунок 2.8 - Діаграма діяльності алгоритму обробки виняткових ситуацій

Проблеми з MQTT-з'єднанням також можуть виникати при втраті мережі або відключенні брокера. Для цього реалізовано автоматичне повторне підключення. Після втрати з'єднання пристрій ініціює повторний запит `CONNECT`, а також перевірку через `PINGREQ` для виявлення активності.

Якщо оновлення прошивки з певних причин не може бути завершене (наприклад, файл не завантажився повністю або встановлення завершилося помилкою), система виконує відкат до попередньої стабільної версії. Це дозволяє уникнути повної втрати працездатності пристрою та забезпечує його подальшу роботу в штатному режимі.

3. РЕАЛІЗАЦІЯ ІНТЕРФЕЙСІВ ПРИКЛАДНОГО ПРОГРАМУВАННЯ

Однією з основних функцій API є інтеграція серверів і клієнтів/пристроїв. Таким чином, два типи програм на серверах і клієнтах потрібні разом, щоб реалізувати API окремо. API фактично надаються програмами на стороні сервера. Вони очікують, що клієнти запитають у них дані. З іншого боку, програми на стороні клієнта можуть маніпулювати API та отримувати через них дані.

3.1 Реалізація інтерфейсів прикладного програмування для серверів

З метою високого паралелізму та продуктивності в реальному часі Node.js використовується для створення HTTP-серверів, MQTT-серверів і програм на стороні сервера, які на них працюють. Node.js використовує неблокуючу модель вводу/виводу для забезпечення високої продуктивності для серверів додатків. Таким чином, багато клієнтів або пристроїв можуть бути підключені до однієї машини в хмарі IoT.

На серверах HTTP, і на MQTT кожен API має слухач, який може відстежувати стани викликів API. URL-адреса або тип пакету MQTT спрямовує наступний запит до відповідного API, запускаючи подію прослуховування слухача. Після цього процес запиту вставляється в чергу подій у наступну позицію, де обробник подій оброблятиме подію поруч із поточною подією. Підхід до обробки кожного API визначається як функція зворотного виклику, яка може бути виконана обробником події, коли подія запускається. Наприклад, коли викликається реєстрація користувача, виконується функція зворотного виклику цього API, що HTTP-сервер зберігає ім’я користувача та пароль у базі даних і відповідає ідентифікатором користувача (UID) клієнтам.

Однак основний процес аналізу пакетів даних відрізняється для веб-API та MQTT API. На серверах HTTP пакети даних із рівня протоколу керування передачею (TCP) аналізуються як пакети HTTP, де розпізнаються заголовки HTTP.

Незважаючи на це, сервери MQTT можуть підтримувати з’єднання MQTT, а також з’єднання WebSocket, як показано на рисунку 3.1.

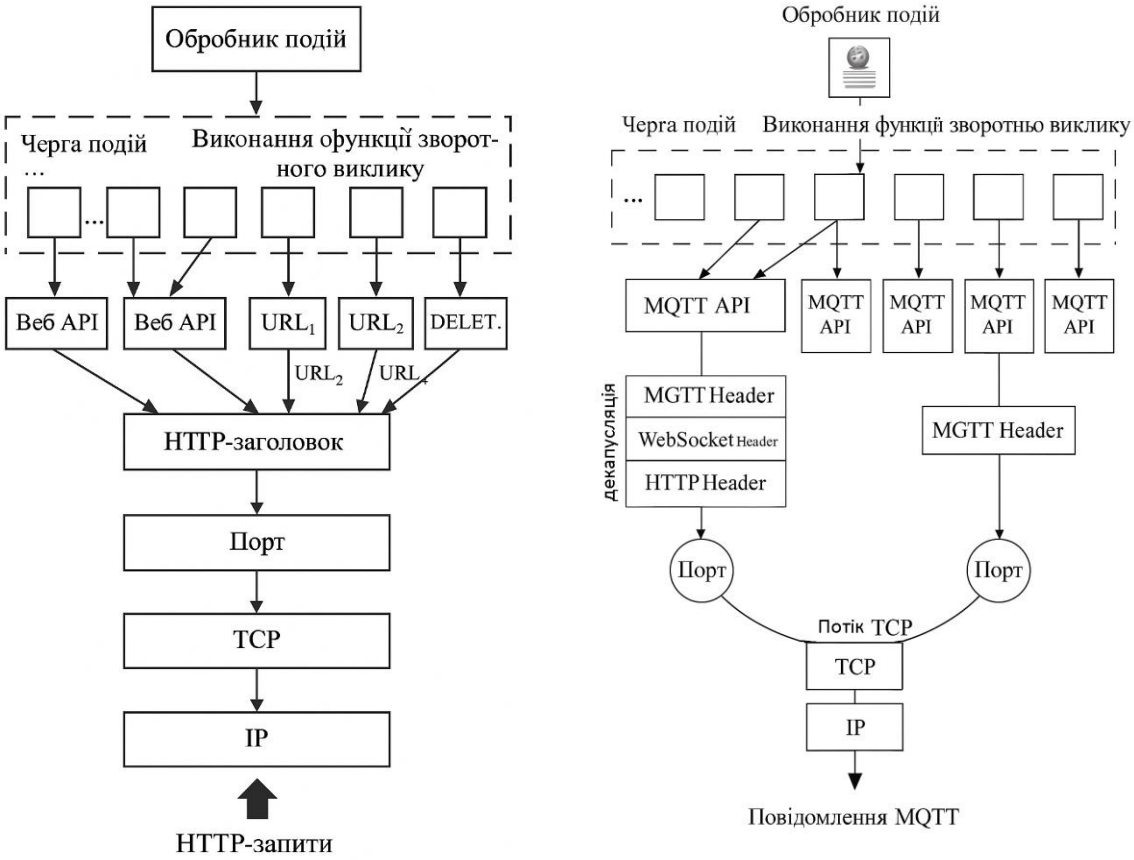


Рисунок 3.1 - Впровадження API

Підключення MQTT використовується для звичайних клієнтів або пристроїв, тоді як підключення WebSocket передбачено для веб-браузерів або деяких сторонніх платформ.

WebSocket має проаналізувати три рівні заголовків пакетів даних, тобто заголовок HTTP, заголовок WebSocket і заголовок MQTT, перш ніж отримати корисне навантаження

3.2 Реалізація інтерфейсів прикладного програмування для клієнтів

Щоб отримати необхідні дані, клієнти повинні робити відповідні запити до серверів додатків. У хмарі IoT існує три види клієнтів, тобто веб-додатки, мобільні

додатки та сторонні платформи. Ці клієнти працюють на різних платформах, але використовують однакові API, щоб дозволити кінцевим користувачам користуватися послугами хмари IoT.

3.2.1 Веб-додатки

Веб-програми зазвичай запускаються у веб-браузерах, які найчастіше використовуються кінцевими користувачами для перегляду Інтернету. Веб-програми зазвичай розгортаються як веб-сторінки. Ці веб-сторінки створюються за допомогою мови гіпертекстової розмітки, каскадних таблиць стилів і JavaScript. Мова гіпертекстової розмітки та каскадні таблиці стилів використовуються для створення статичних веб-сторінок, тоді як JavaScript визначає дії щодо поведінки кінцевих користувачів. Наприклад, коли кінцеві користувачі натискають гіперпосилання, яке запитує інформацію про зв’язані пристрої, програми JavaScript можуть генерувати запити, формати та вміст яких визначаються списком пристроїв, і надсилати ці запити на сервери HTTP. Цей процес реалізується за допомогою асинхронного JavaScript і XML (Ajax), який призначений для того, щоб клієнти могли асинхронно обмінюватися даними з серверами. Ajax викликається за допомогою популярної бібліотеки JavaScript, відомої як jQuery. За допомогою Ajax запити можна надсилати на сервери, не перериваючи існуючі сторінки. HTTP-сервери зазвичай відповідають веб-переглядачам веб-сторінками, які можна аналізувати та відображати кінцевим користувачам або розробникам. Таким чином кінцеві користувачі та розробники можуть отримати необхідну інформацію.

Хмара IoT в основному пропонує функції керування для кінцевих користувачів для керування пристроями, прив’язаними до їхніх мобільних додатків, і для розробників для налагодження прикладних програм за допомогою веб-додатків. Для цих функцій у хмарі IoT створено веб-API, тобто список пристроїв. За допомогою списку пристроїв кінцеві користувачі можуть переглядати детальну інформацію про прив’язані пристрої, наприклад статус роботи. З іншого боку, розробники можуть перевіряти інформацію як кінцевих користувачів, які використовують програми, так і пристрої, якими керують

програми. Крім того, розробники можуть запитувати зв’язки між цими кінцевими користувачами та пристроями, що може допомогти розробникам змінювати програми належним чином.

3.2.2 Системи для розробки програмного забезпечення та мобільні додатки

Для зручності розробки мобільних додатків розробникам надано клієнтські бібліотеки, які містять коди реалізації клієнтської сторони API. Ці бібліотеки входять до комплектів розробки програмного забезпечення (SDK) або доступні окремо. Завдяки цим SDK можна заощадити час для розробників, оскільки їм не потрібно генерувати необроблені запити чи повідомлення, а викликати наявні та зручні для користувача інтерфейси в SDK.

Взаємодію між мобільними додатками та серверами додатків за допомогою API показано на рисунку 3.2.

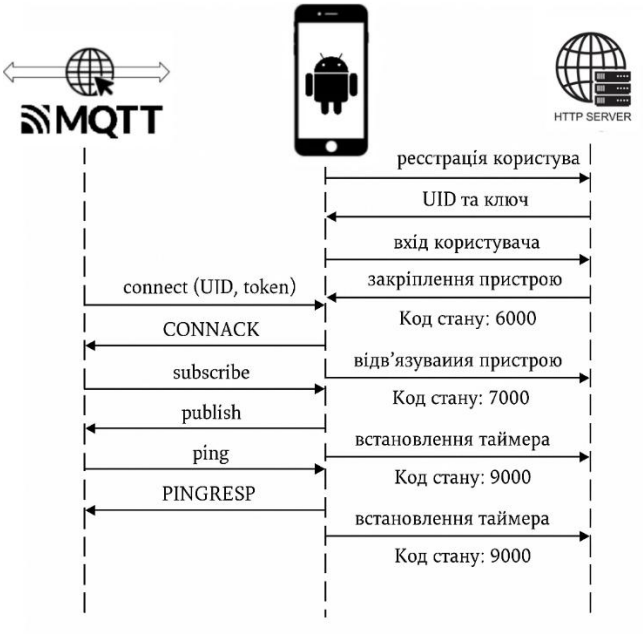


Рисунок 3.2 - Взаємодія мобільних застосунків із серверами застосунків

3.2.3 Додавання пристроїв Інтернету речей

Постачальники можуть проектувати та виготовляти пристрої, які мають доступ до спеціально розроблених API. Через всюдисущість і велику кількість пристроїв хмара IoT повинна їх легко розрізняти. Як правило, пристрої IoT

оснащені модулями Wi-Fi, які дозволяють пристроям отримувати доступ до хмари IoT через Інтернет. Основний компонент одного модуля Wi-Fi, тобто контролер мережевого інтерфейсу Wi-Fi, має унікальну адресу керування доступом до медіа (MAC), яка призначена для визначення місця розташування пристрою в мережах. Завдяки унікальності MAC-адреси її можна використовувати, щоб відрізнити пристрій від інших. З іншого боку, постачальники/виробники оригінального обладнання використовують один ключ продукту для ідентифікації одного типу пристрою. Крім того, серія паролів вбудована в пристрої, які використовуються для хмари IoT для перевірки ідентифікації пристроїв, зареєстрованих у хмарі. За допомогою MAC-адреси, ключа продукту та пароля хмара IoT може генерувати унікальний ідентифікатор пристрою (DID) для точної ідентифікації пристроїв. Пристрої можуть отримати свої DID із серверів HTTP через процес реєстрації.

Взаємодія між пристроями IoT і серверами додатків показано на рисунку 3.3.

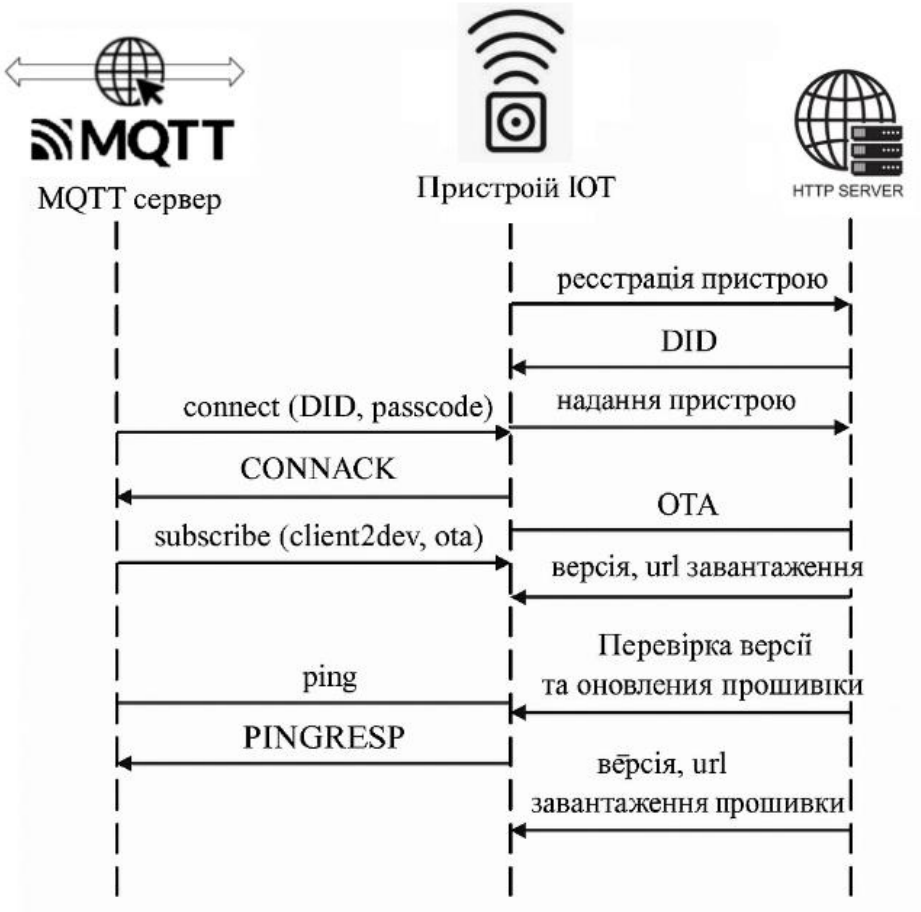


Рисунок 3.3 - Приклад взаємодії між пристроями Інтернету речей із серверами додатків

Щоб отримати доступ до хмари IoT, нові пристрої повинні спочатку зареєструватися на серверах HTTP за допомогою реєстрації пристрою. Під час цього процесу MAC-адреси, ключі продукту та паролі потрібно надсилати на сервери HTTP. Потім HTTP-сервери генерують DID для пристроїв і надсилають їх назад як відповіді. Після цього пристрої повинні підключитися до серверів MQTT, перш ніж вони зможуть спілкуватися з користувачами. Тому вони повинні надсилати запити на надання через надання пристрою, щоб отримати адреси та порти серверів MQTT, які зберігаються на серверах HTTP. Потім пристрої можуть створювати підключення до серверів MQTT за допомогою своїх DID і паролів. Ці зв'язки мають бути стійкими; таким чином, пристрої можуть обмінюватися повідомленнями з кінцевими користувачами через підписку та безперервну публікацію

Пристрої IoT мають підписатися на дві основні теми, а саме «client2dev» і «ota», коли вони підключаються до серверів MQTT. Кожен пристрій має унікальну тему «client2dev», яка зазвичай використовується для програм для публікації повідомлень на пристрої. Коли кінцеві користувачі прив'язують пристрій, вони також повинні підписатися на ту саму тему «client2dev», за допомогою якої вони можуть спілкуватися з пристроєм.

3.3 Тестування та оцінка продуктивності

Щоб оцінити продуктивність програмного інтерфейсу прикладного програмування та розроблених API, було проведено кілька експериментів на тестовій системі IoT за різних умов.

3.3.1 Конфігурації

Сервери додатків тестової хмари IoT складаються з одного HTTP-сервера та одного MQTT-сервера відповідно. Крім того, сервер Redis розгортається як база даних. Детальну інформацію про ці сервери наведено в таблиці 3.1, а також API, які тестуються.

Таблиця 3.1 - Деталі конфігурації серверів у хмарі Інтернету речей

Тип сервера	Кількість серверів	Кількість CPU	Модель CPU	Частота (ГГц)	Пам’ять (ГБ)	API/Інтерфейси
HTTP сервер	1	1	Intel Xeon E5-2680	2.7	16	Прив'язка пристрою, розрив прив’язки, вхід користувача, підготовка пристрою, реєстрація користувача
MQTT сервер	1	1	Intel Xeon E5-2680	2.7	16	Публікація
Redis сервер	1	1	Intel Xeon E5-2680	2.7	16	----

Locust (http://locust.io/) використовувався для тестування продуктивності веб-API. Він може імітувати навантаження HTTP-сервера на різну кількість користувачів. Такий тест створює багато віртуальних користувачів спочатку на машині. Кожен віртуальний користувач щоразу випадково надсилає випадковий запит через певний веб-API, який може імітувати реальну ситуацію, у якій використовуються веб-API. У цьому процесі Locust може підрахувати середній час відповіді, а також винятки.

Ще один тест проводився для оцінки основного MQTT API, тобто публікації. У цьому тесті кількість користувачів, які публікують повідомлення, коливалось від 50 000, 100 000, 150 000 і 200 000. Ці користувачі публікували випадкові повідомлення на тему кожні 2 хвилини, що відповідає фактичним умовам. Ці затримки можуть відображати продуктивність публікації. Крім того, відстежуються робочі стани, особливо зайнятість пам’яті, сервером MQTT різною кількістю користувачів і порівнюються з сервером Redis.

3.3.2 Результати

На рисунку 3.4 показано, що середній час відповіді різних веб-API зростає зі збільшенням кількості користувачів, тому що чим більше користувачів надсилають запити до HTTP-сервера, тим більше стає навантаження на сервер. Таким чином, середній час відповіді стає довшим. З іншого боку, середній час відповіді зв'язування пристрою залишається найвищим, головним чином через те, що процес зв'язування вимагає зберігання великої кількості даних, таких як UID та DID. Однак середній час відповіді між веб-API насправді невеликий.

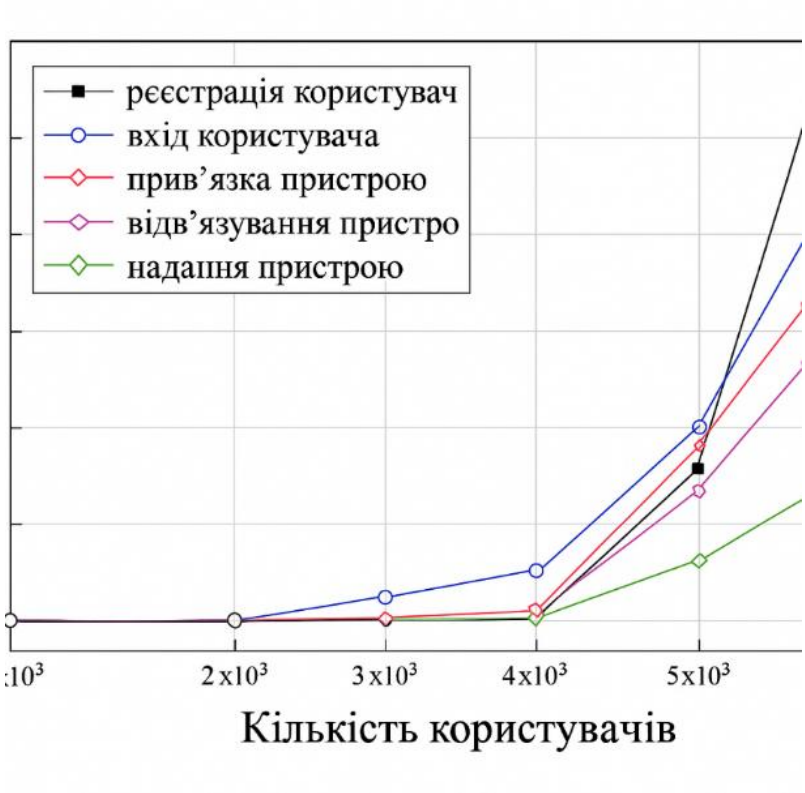


Рисунок 3.4 - Середній час відгуку для різної кількості користувачів та інтерфейсів веб-застосунків

Процеси зміни середнього часу відгуку проілюстровано на рисунку 3.5 Середній час відгуку збільшується зі збільшенням часу роботи. Серед них середній час відгуку залишається близьким до нуля, коли кількість користувачів залишається меншою за 2000.

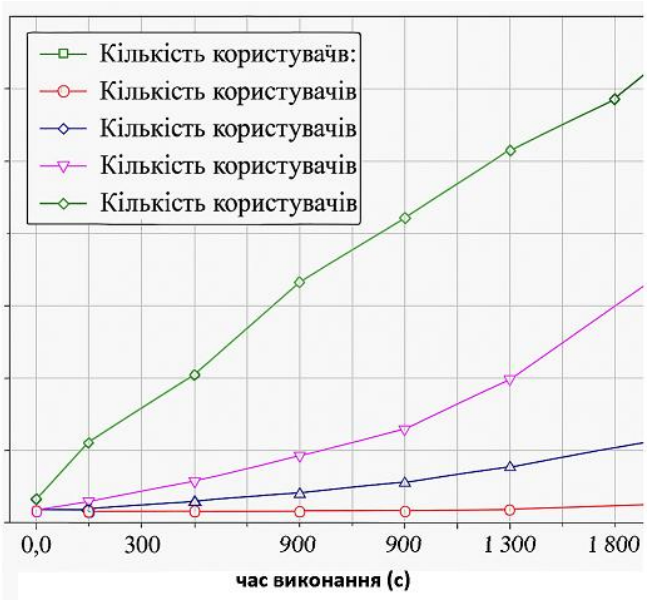


Рисунок 3.5 - Середній час відгуку для різної кількості користувачів та часу виконання

Однак середній час відгуку стає високим, коли кількість користувачів досягає близько 5000. Через обмеження машин для тестування середній час відгуку продовжує зростати. Продуктивність публікації на серверах MQTT зображена на рисунку 3.6.

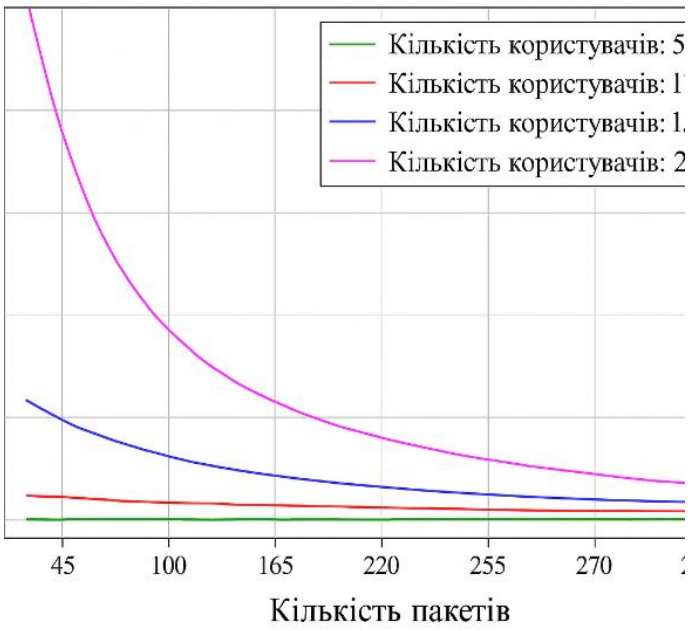


Рисунок 3.6 - Середня затримка передачі публікації на серверах телеметрії черги повідомлень (MQTT) для різної кількості користувачів

Як видно, кількість користувачів має великий вплив на продуктивність публікації. Чим більше користувачів публікують повідомлення на сервері, тим довшими стають затримки. Коли кількість користувачів досягає 200000, середні затримки вдвічі перевищують інші умови. Це означає, що продуктивність публікації досягає свого обмеження в ситуації, коли кількість користувачів досягає приблизно 200 000. З іншого боку, через те, що сервер MQTT стає все більш і більш стабільним під час роботи, середня затримка передачі поступово зменшується.

Статуси зайнятості пам’яті на сервері MQTT і сервері Redis показані на рисунку 3.7.

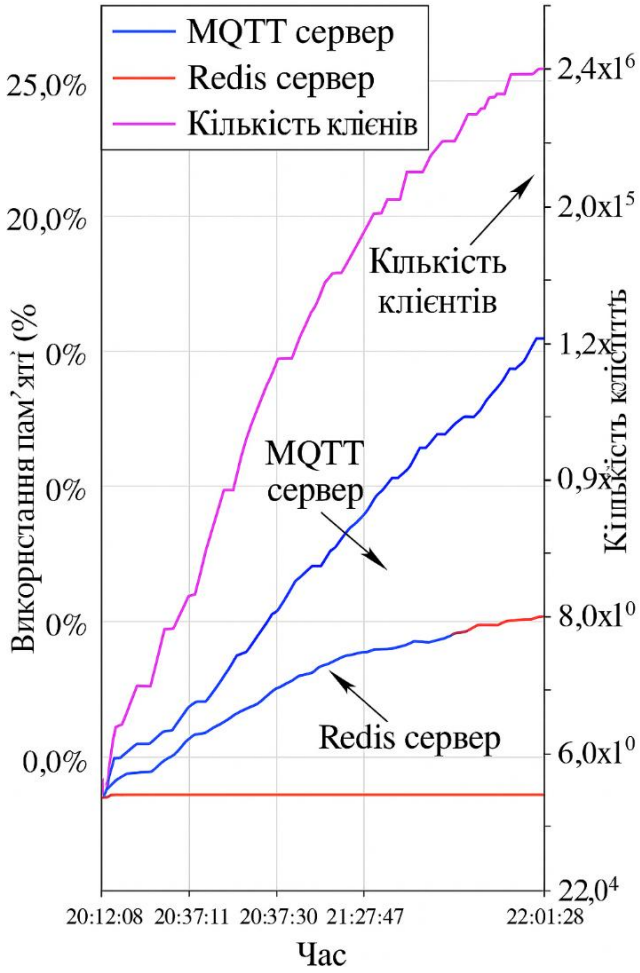


Рисунок 3.7 - Зайнятість пам'яті сервером телеметрії черги повідомлень (MQTT) та сервером Redis у різний час та за різним номером клієнта

Із збільшенням кількості користувачів на сервері MQTT зайнятість пам’яті зростає з тією ж швидкістю. Однак опускання кривої MQTT в кінці означає, що

сервер MQTT досяг своїх обмежень, що більше користувачів не можна підключати до нього. Це показує, що публікацію на одному сервері MQTT можуть одночасно використовувати щонайбільше 230 000 користувачів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було створено прототип програмного інтерфейсу прикладного програмування для хмари Інтернету речей (API), яка забезпечує безпечну реєстрацію пристроїв і користувачів, обробку сенсорних даних, дистанційне керування та оновлення мікропрограм з використанням протоколів HTTP та MQTT.

1. Проаналізовано сучасні технології та протоколи побудови API для хмарних IoT-систем, зокрема REST, MQTT, WebSocket, GraphQL, OPC UA, а також принципи безпеки Zero Trust, використання JWT, OAuth2 та інших засобів автентифікації й захисту API-трафіку.

2. Проведено огляд існуючих IoT-рішень, включаючи системи для розумних міст, агросектору, медицини та промисловості. Проаналізовано підходи до реалізації API-шарів, API-шлюзів, багаторівневих мікросервісних архітектур з використанням хмарних сервісів AWS, Azure, Google Cloud, ThingSpeak.

3. Сформульовано технічні вимоги до API хмарної IoT-системи, які передбачають автентифікацію користувачів, реєстрацію пристроїв, обробку сенсорних повідомлень у реальному часі, оновлення прошивок OTA, а також механізми повторних підключень, heartbeat-запити, протоколювання та обробку виняткових ситуацій.

4. Розроблено концепцію API, що охоплює HTTP-вебсервіси для реєстрації, автентифікації, керування пристроями, а також MQTT-інтерфейси для публікації сенсорних даних, підписки на команди, оновлення мікропрограм. Створено RESTful URL-ендпоінти, схеми відповідей та кодів статусів.

5. Реалізовано алгоритмічне забезпечення API, включно з процесами ініціалізації пристроїв, реєстрації, підключення, моніторингу, OTA-оновлень, обробки помилок, повторних спроб та механізмів діагностики стану з’єднання через PING-запити.

6. Реалізовано програмні інтерфейси прикладного програмування, як для серверної частини, так і для клієнтської.

7. Проведено тестування та оцінку продуктивності API, зокрема навантажувальне тестування HTTP-інтерфейсів за допомогою Locust та MQTT-серверів з підключенням до 200 000 віртуальних пристроїв. Проаналізовано затримки, час відповіді, використання пам’яті та надійність сервісів під час пікових навантажень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M. Internet of things: a survey on enabling technologies, protocols and applications. *IEEE Communications Surveys & Tutorials*. 2015. Vol. 17, No. 4. P. 2347–2376.

2. Arora, S., Hastings, J. Microsegmented Cloud Network Architecture Using Open-Source Tools for a Zero Trust Foundation [Електронний ресурс] // arXiv:2411.12162. – 2024. – Режим доступу: https://arxiv.org/abs/2411.12162.

3. Barros, L. L. Cloud Interface for Osmosis System [Електронний ресурс] // UTFPR Repository. – 2019. – Режим доступу: http://riut.utfpr.edu.br/jspui/handle/1/35660.

4. Chen, J., Yu, Q., Chai, B., Sun, Y., Fan, Y., Shen, X. Dynamic channel assignment for wireless sensor networks: a regret matching based approach. IEEE *Transactions on Parallel and Distributed Systems*. 2015. Vol. 26, No. 1. P. 95–106.

5. Chen, S., Zhao, J. The requirements, challenges, and technologies for 5G of terrestrial mobile telecommunication. *IEEE Communications Magazine*. 2014. Vol. 52, No. 5. P. 36–43.

6. Islam, M. M. AI-Driven Multi-Purpose Platform for Smart Healthcare Systems: дис. ... магістра наук. University of Waterloo. 2025.

7. Ionescu, D., Filipescu, A., Simion, G., Filipescu, A. Internet of Things-Cloud Control of a Robotic Cell Based on Inverse Kinematics, Hardware-in-the-Loop, Digital Twin, and Industry 4.0/5.0 *Sensors.* 2025. Vol. 25. Article No. 1821. DOI: 10.3390/s25061821.

8. Lei, L., Kuang, Y., Cheng, N., Shen, X., Zhong, Z., Lin, C. Delay-optimal dynamic mode selection and resource allocation in device-to-device communications part 2: practical algorithm. IEEE Transactions on Vehicular Technology. 2016. Vol. 65, No. 5. P. 3491–3505.

9. Maulana, I., Fitria, M., Faizah, C. N., Adharani, D. H., Jinan, S. N., Dawood, R. Development of a Mobile Monitoring Application for a Message Queuing Telemetry

Transport (MQTT)-Based Struvite Fertilizer Production System. *Proc. of 10th Int. HCI and UX Conf. in Indonesia* (CHIuXiD). 2024. P. 84–89.

10. Montesines-Nagayo, A., Al-Ajmi, M., Guduri, N. V. R. IoT-Based Telemedicine Health Monitoring System with a Fuzzy Inference-Based Medical Decision Support Module for Clinical Risk Evaluation. LNNS, *Springer*. 2023. Vol. 612. P. 275–289.

11. Nagayo, A. M., Al Ajmi, M. Z. K., Guduri, N. R. K., AlBuradai, F. S. H. IoT-Based Telemedicine Health Monitoring System [Електронний ресурс]. 2024. Режим доступу: https://www.researchgate.net/publication/369346948.

12. O'Mahony, N., Walsh, J. WSN and IoT-Based Wire Monitoring. *In: Smart Innovation, Systems and Technologies*. Springer, 2024.

13. Ramezani, R., Iranmanesh, S., Naeim, A. AI and Remote Monitoring in Digital Health. *Frontiers in Digital Health.* 2025. – Режим доступу: https://www.frontiersin.org/articles/10.3389/fdgth.2025.1584443.

14. Su, Z., Qichao, X., Qi, Q. Big data in mobile social networks: a QoE oriented framework. *IEEE Network*. 2016. Vol. 30, No. 1. P. 52–57.

15. Su, Z., Xu, Q., Zhu, H., Wang, Y. A novel design for content delivery over software defined mobile social networks. IEEE Network. 2015.Vol. 29, No. 4.P. 62–67.

16. Suhaeb, S., Risal, A. Implementation of ESP32-Based Web Host for Control and Monitoring of Robotic Arm. *JESSI.* 2024. Vol. 5, No. 3. P. 249–254.

17. Wahyudi, A. E., Sari, S. K., Aziz, F. Jeffry. Implementation of an Internet of Things (IoT)-Based Air Quality Monitoring System for Enhancing Indoor Environments. *JSCE*. 2025. Vol. 6, No. 1. P. 124–130.

18. Yuan, D., Jin, J., Grundy, J., Yang, Y. A framework for convergence of cloud services and Internet of things. Proc. of 19th Int. Conf. on Computer Supported Cooperative Work in Design (CSCWD). Calabria, Italy: IEEE, 2015. P. 349–354.

19. Zhang, H., Cheng, P., Shi, L., Chen, J. Optimal DoS attack scheduling in wireless networked control system. IEEE Transactions on Control Systems Technology. 2016. Vol. 24, No. 3. P. 843–852.

20. Ziuziun, V. Mathematical Justification of IoT System for Soil Moisture Monitoring. *KSAU Journal*. 2019. Режим доступу: https://www.journals.ksauniv.ks.ua/index.php/tech/article/download/728/680/1379.

21. Raza S., Wallgren L., Voigt T. Cloud of Things: Architecture, Research Challenges, Security Threats, Mechanisms, and Open Challenges. *Computer Communications.* 2020. Vol. 156. P. 1–20. DOI: 10.1016/j.comcom.2020.03.003.

22. EMQX Team. Connecting MQTT and REST API: A Comprehensive Tutorial [Електронний ресурс]. – Режим доступу: https://www.emqx.com/en/blog/connecting-mqtt-and-rest-api

23. RadView Software. Advanced Guide to API Performance Testing [Електронний ресурс]. – Режим доступу: https://www.radview.com/blog/blog-api-performance-testing-advanced-guide/

24. Kong Inc. The Critical Role of API Security in the Internet of Things (IoT) [Електронний ресурс]. – Режим доступу: https://konghq.com/blog/enterprise/iot-api-security-guide

25. Particle Industries. IoT Scalability: What It Means, Common Challenges, and How to Scale Effectively [Електронний ресурс]. – Режим доступу: https://www.particle.io/iot-guides-and-resources/iot-scalability/

26. Espinal R. How to Connect an ESP32 to the IoT Cloud [Електронний ресурс]. – Режим доступу: https://www.instructables.com/How-to-Connect-an-ESP32-to-the-IoT-Cloud/

27. Dev A. Build REST API Using Node.js, Express and MongoDB [Електронний ресурс]. – YouTube, 2022. – Режим доступу: https://www.youtube.com/watch?v=Yvcv0mh1LOc

28. Ружицький Д., Осолінський О. Концепція інтерфейсів прикладного програмування для хмари інтернету речей. Інтелектуальні інформаційні технології в прикладних дослідженнях : тези доп. студентської наук.-практ. конф. (м. Тернопіль, 27–29 травня 2025 р.). Тернопіль, 2025. С.312-314

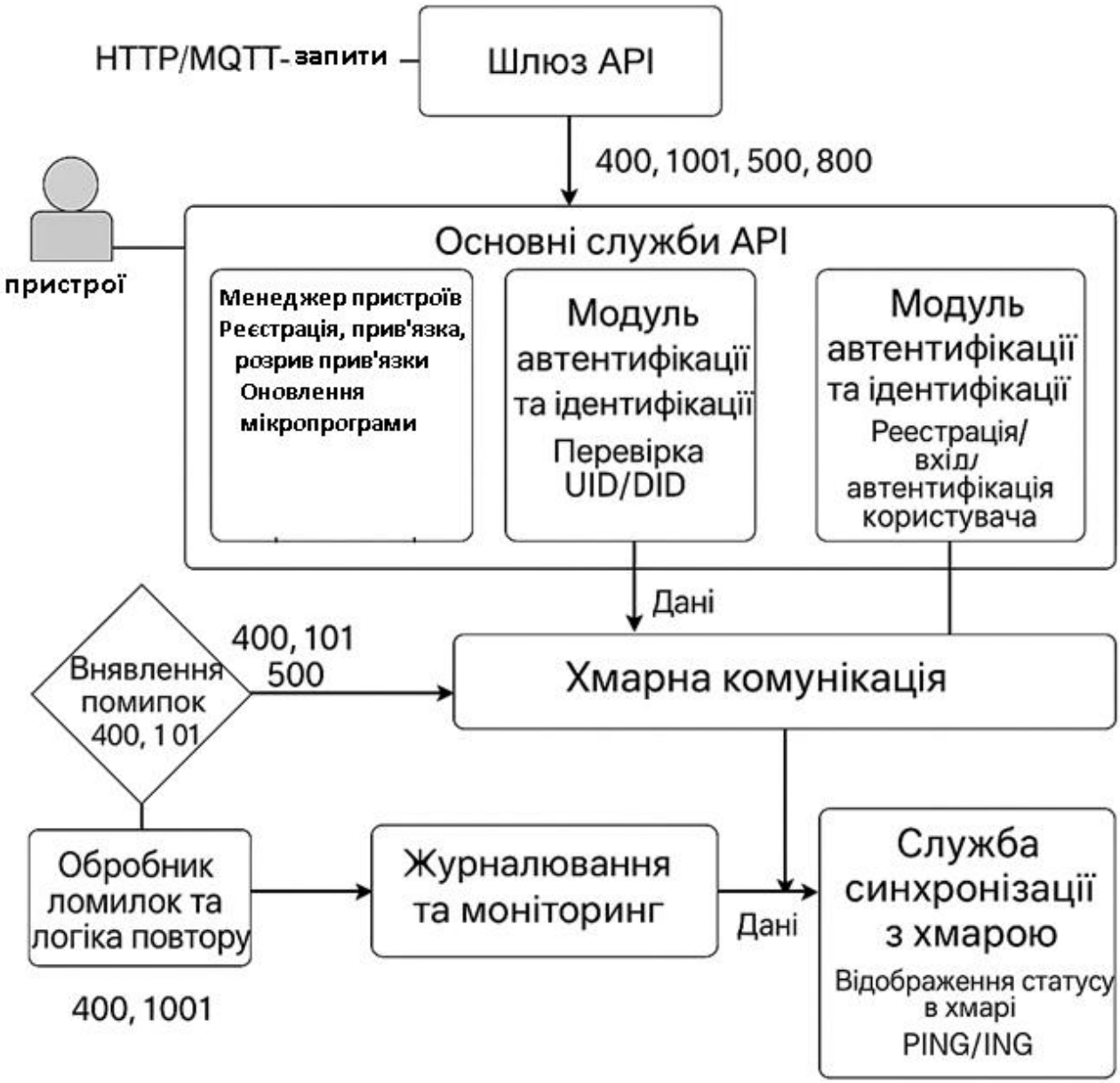
29. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної

роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

30. Островерхов В. М., Біловус Л. І., Возьний К. З., Луцишин О. О., Монастирський Г. Л., Надвиничний С. А., Питель С. В., Шандрук С. К. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

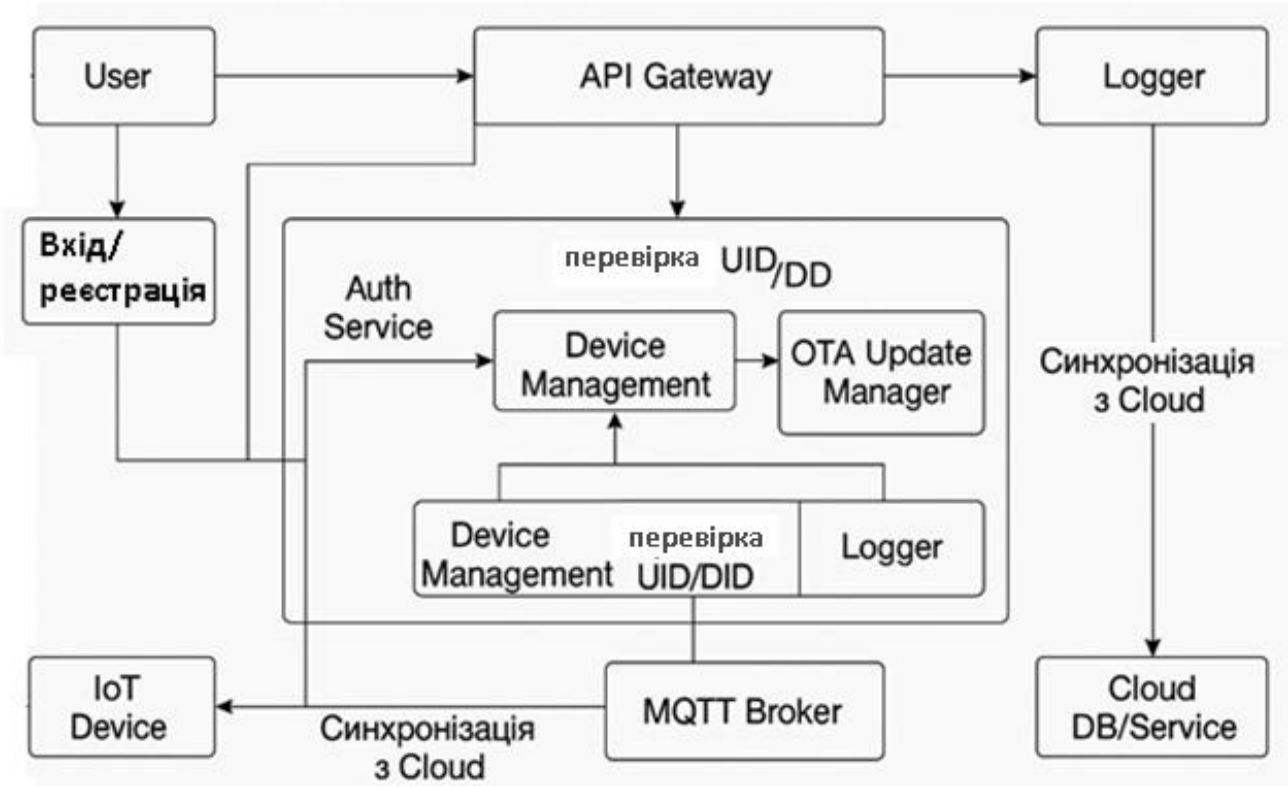
Додаток А

Архітектура модуля реалізації програмного інтерфейсу



Додаток Б

Функціональна схема програмного інтерфейсу



Додаток В

Код-прототип частини системи

app.js — запуск сервера

```
require('dotenv').config();
const express = require('express');
const app = express();
const deviceRoutes = require('./routes/devices');
app.use(express.json());
app.use('/api/devices', deviceRoutes);
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

routes/devices.js — маршрути для роботи з пристроями

```
const express = require('express');
const router = express.Router();
const deviceController = require('../controllers/deviceController');
router.post('/register', deviceController.registerDevice);
router.post('/publish', deviceController.publishData);
module.exports = router;
```

controllers/deviceController.js — логіка реєстрації та публікації

```
const mqttClient = require('../services/mqttClient');
const devices = [];

exports.registerDevice = (req, res) => {
    const { uid, type } = req.body;
    if (!uid || !type) {
        return res.status(400).json({ message: 'UID and type are required'
});
    }
    devices.push({ uid, type });
    return res.status(201).json({ message: 'Device registered', uid });
};

exports.publishData = (req, res) => {
```

```
    const { uid, payload } = req.body;
    if (!uid || !payload) {
        return res.status(400).json({ message: 'UID and payload required' });
    }

    const topic = `iot/data/${uid}`;
    mqttClient.publish(topic, JSON.stringify(payload));
    return res.status(200).json({ message: 'Data published', topic });
};
```

services/mqttClient.js — MQTT-клієнт

```
const mqtt = require('mqtt');
const client = mqtt.connect('mqtt://broker.hivemq.com');

client.on('connect', () => {
    console.log('Connected to MQTT broker');
});

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

module.exports = client;
```

Додаток Г
Апробація отриманих результатів

Західноукраїнський національний університет

Факультет комп'ютерних інформаційних технологій

Кафедра інформаційно-обчислювальних систем і управління

ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

Студентської науково-практичної конференції

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРИКЛАДНИХ ДОСЛІДЖЕННЯХ

(IITAR-2025)

27-29 травня 2025 року

Тернопіль

2025

Збірник тез доповідей студентської науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (IITAR-2025). (Тернопіль, 27-29 травня 2025 року). Тернопіль: ЗУНУ, 2025. 372 с.

До збірника увійшли тези доповідей учасників студентської науково-практичної конференції "Інтелектуальні інформаційні технології в прикладних дослідженнях" (IITAR – 2025), що відбувалась у рамках AI Week на базі кафедри інформаційно-обчислювальних систем і управління Західноукраїнського національного університету. Мета конференції — об'єднати науковців, освітян, представників IT-бізнесу та здобувачів освіти для обміну досвідом, презентації сучасних досліджень і впровадження інноваційних рішень у сфері комп'ютерних наук, штучного інтелекту та суміжних галузей.

За зміст наукових праць та достовірність наведених фактологічних і статистичних матеріалів відповідальність несуть автори публікацій та їхні наукові керівники. У збірнику зберігається стилістика та орфографія авторів матеріалів.

Склад організаційного комітету конференції

Ружицький Дмитро
студент групи КН-43
ruzh.dmyrto@gmail.com
Осолінський Олександр
к.т.н., доцент
oso@wunu.edu.ua
Західноукраїнський національний університет
Тернопіль, Україна

КОНЦЕПЦІЯ ІНТЕРФЕЙСІВ ПРИКЛАДНОГО ПРОГРАМУВАННЯ ДЛЯ ХМАРИ ІНТЕРНЕТУ РЕЧЕЙ

В Інтернеті речей (IoT) тисячі об'єктів або пристроїв можуть бути плавно з'єднані між собою за допомогою зв'язку «пристрій-пристрій» або «машина-машина» [1]; таким чином, вони можуть працювати у співпраці один з одним інтелектуальним способом [2]. Значне збільшення даних, які генеруються пристроями IoT, також сприяє прогресу п'ятого покоління (5G) для IoT [3]. Щоб повністю розкрити потенціал IoT, необхідно використовувати та застосовувати технології хмарних обчислень [4]. Хмара IoT дозволяє пристроям IoT підключатися до хмарних інфраструктур через Інтернет. Завдяки необмеженому сховищу, обчислювальним і мережевим ресурсам хмара IoT пропонує можливість керувати великомасштабними постійними з'єднаннями для IoT [5]. Крім того, хмара може забезпечити аналітику даних у реальному часі, дозволяючи пристроям IoT розвантажувати обчислювальні завдання та величезний обсяг даних [6]. Таким чином, хмара IoT може добре підтримувати різноманітні програми та сервіси [7].

Як правило, клієнти можуть дистанційно користуватися послугами хмари IoT за допомогою інтерфейсів прикладного програмування (API). Таким чином можна легко розробляти та розгортати хмарні програми IoT. Отже добре спроектовані та належним чином реалізовані API важливі для мінімізації часу розгортання додатків і забезпечення хмарних служб для IoT [8].

Щоб вирішити цю проблему, потрібен новий дизайн та реалізації API для хмари IoT. Обидва протоколи HyperText Transfer Protocol (HTTP) і Message Queuing Telemetry Transport (MQTT) підтримуються хмарою IoT з огляду на різні служби та пристрої. Як наслідок, потрібно два типи API, які відповідають кожному протоколу програми. За таких API послуги на основі HTTP та послуги на основі MQTT можуть якісно надаватися кінцевим користувачам.

Розробка концепції інтерфейсів прикладного програмування для хмари Інтернету речей

Хмара IoT може пропонувати різноманітні послуги, які допомагають розробникам програмного забезпечення розробляти та запускати програми на основі хмарної інфраструктури IoT або постачальникам/виробникам оригінального обладнання (OEM) для виробництва пристроїв, які мають доступ до хмари IoT. Ці служби розгортаються на двох типах серверів додатків у хмарі

IoT, тобто на серверах HTTP(S) та серверах MQTT. Кожна служба містить кілька різних функцій, як показано на рисунку 1.

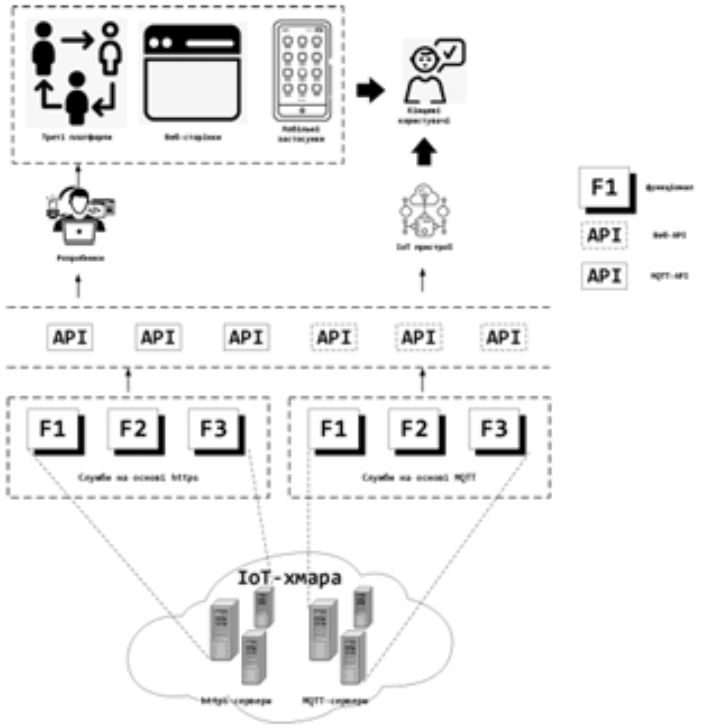


Рисунок 1 - Типи сервісів і дизайн API в хмарі IoT. API, інтерфейс прикладного програмування; IoT, Інтернет речей; HTTPS, протокол передачі гіпертексту; MQTT, телеметричний транспорт у черзі повідомлень

Сервіси, які пропонує хмара IoT, можна розділити на дві категорії, тобто послуги на основі HTTPS та послуги на основі MQTT, відповідно до різних протоколів додатків, які вони прийняли. Служби на основі HTTP надаються клієнтам для обміну гіпертекстами між програмами клієнтів і серверів через Інтернет.

Що стосується пристроїв, вони повинні використовувати надання пристрою для доступу до серверів MQTT.

Після цього сервери можуть відповісти пристроям із номером версії та уніфікованим покажчиком ресурсу (URL), на який можна завантажити мікропрограмне забезпечення; таким чином, пристрої можуть зручно оновлювати мікропрограму.

Висновки

Було розроблено концепцію програмного інтерфейсу прикладного програмування (API) для хмари IoT, яка забезпечує уніфіковану взаємодію між пристроями, хмарними сервісами та клієнтськими додатками. API орієнтовано на дві групи сервісів: на основі протоколу HTTP та на основі протоколу MQTT.

Такий підхід дозволяє ефективно інтегрувати як потужні обчислювальні пристрої, так і ресурсообмежені сенсорні вузли.

Розроблена модель дозволяє масштабувати систему до тисяч пристроїв, що одночасно взаємодіють із хмарною платформою, а також інтегрувати її з зовнішніми сервісами завдяки підтримці відкритих стандартів API.

Список використаних джерел

1. Walia, G. K., Kumar, M., & Gill, S. S. (2023). AI-empowered fog/edge resource management for IoT applications: A comprehensive review, research challenges, and future perspectives. *IEEE Communications Surveys & Tutorials*, 26(1), 619-669.
2. Arora, S., Hastings, J. Microsegmented Cloud Network Architecture Using Open-Source Tools for a Zero Trust Foundation [Електронний ресурс] // arXiv:2411.12162. – 2024. – Режим доступу: https://arxiv.org/abs/2411.12162.
3. Barros, L. L. Cloud Interface for Osmosis System [Електронний ресурс] // *UTFPR Repository*. – 2019. – Режим доступу: http://riut.utfpr.edu.br/jspui/handle/1/35660.
4. Chen, J., Yu, Q., Chai, B., Sun, Y., Fan, Y., Shen, X. Dynamic channel assignment for wireless sensor networks: a regret matching based approach // *IEEE Transactions on Parallel and Distributed Systems*. – 2015. – Vol. 26, No. 1. – P. 95–106.
5. Chen, S., Zhao, J. The requirements, challenges, and technologies for 5G of terrestrial mobile telecommunication // *IEEE Communications Magazine*. – 2014. – Vol. 52, No. 5. – P. 36–43.
6. Islam, M. M. AI-Driven Multi-Purpose Platform for Smart Healthcare Systems: *дис. магістра наук* / University of Waterloo. – 2025.
7. Ionescu, D., Filipescu, A., Simion, G., Filipescu, A. Internet of Things-Cloud Control of a Robotic Cell Based on Inverse Kinematics, Hardware-in-the-Loop, Digital Twin, and Industry 4.0/5.0 // *Sensors*. – 2025. – Vol. 25. – Article No. 1821. – DOI: 10.3390/s25061821.
8. Teng, Y., Liu, M., Yu, F. R., Leung, V. C., Song, M., & Zhang, Y. (2018). Resource allocation for ultra-dense networks: A survey, some research issues and challenges. *IEEE Communications Surveys & Tutorials*, 21(3), 2134-2168.