НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет/(ННІ) **Інформаційних Технологій**

ПОГОДЖЕНО
Декан факультету (Директор ННІ)
Інформаційних Технологій
(назва факультету (ННІ))

_____ Ігор БОЛБОТ
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 20__ р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних систем і мереж
(назва кафедри)

_____ Дмитро КАСАТКІН
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 20__ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Моніторинг і керування граничними пристроями IOT на базі raspberry pi

Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма Комп'ютерні системи захисту інформації
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
д.пед.н., професор _____ Сергій МАМЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
д.т.н., професор _____ Валерій ЛАХНО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Анатолій КАСАТКІН
(підпис) ((ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 20___

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) Інформаційних Технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
к. пед. н., доцент _____ Дмитро КАСАТКІН
(науковий ступінь, вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)
"____" _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Касаткін Анатолій Дмитрович
(прізвище, ім'я, по батькові)

Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма Комп'ютерні системи захисту інформації
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Моніторинг і керування граничними пристроями IOT на базі Raspberry Pi

затверджена наказом. від "15"жовтня 2024р. №585 «С»

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи Інфраструктура «розумного будинку» на базі Raspberry Pi (Model B) з датчиками DS18B20, BMP180, TSL2561, локальна мережа передачі даних, програмне забезпечення для збору та зберігання телеметрії

1. Проаналізувати існуючі підходи до побудови IoT-інфраструктур та захисту даних з використанням технології блокчейн у системах «розумного будинку».

2. Розробити модель IoT-інфраструктури розумного будинку з інтеграцією технології блокчейн для фіксації телеметрії та подій, обґрунтувати її архітектуру та вибір програмно-апаратних засобів

3. Реалізувати програмний макет системи, провести експериментальні дослідження (продуктивність, затримки, обсяг даних у блокчейні) та оцінити ефективність запропонованого підходу щодо забезпечення цілісності та захищеності даних

Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ Валерій ЛАХНО
(підпис) (ім'я ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ _____Анатолій Касаткін_____
(ім'я ПРІЗВИЩЕ)

ЗМІСТ

ВСТУП 2

РОЗДІЛ 1. ОГЛЯД АРХІТЕКТУРИ МЕРЕЖІ ІНТЕРНЕТУ РЕЧЕЙ 4

1.1 Екосистема інтернету речей 5

1.2 Огляд архітектури інтернету речей 6

1.3 Огляд компонентів архітектури інтернету речей 9

1.4 Огляд технологій та протоколів передачі даних у мережі інтернет речей 14

1.5 Протоколи для передачі повідомлень в мережі інтернету речей 21

1.6 Огляд існуючих засобів моніторингу та управління елементами інтернету речей 27

1.7 Висновок до розділу 31

РОЗДІЛ 2. ОПИС І АНАЛІЗ АПАРАТНОЇ ЧАСТИНИ ТА НАЛАШТУВАННЯ ЗАСОБІВ МОНІТОРИНГУ ТА УПРАВЛІННЯ ЕЛЕМЕНТАМИ ГРАНИЧНИХ ПРИСТРОЇВ ІОТ 32

2.1 Опис вибраної платформи Raspberry Pi 32

2.2 Установлення та налаштування Raspberry Pi 35

2.3 Підключення зовнішнього диска та модуля Wi-Fi 37

2.4 Підключення до Raspberry Pi з допомогою ssh. Налаштування з’єднання та встановлення Domoticz 39

2.5 Аналіз Domoticz API, як способу управління 46

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАСОБІВ МОНІТОРИНГУ ТА УПРАВЛІННЯ ЕЛЕМЕНТАМИ ГРАНИЧНИХ ПРИСТРОЇВ ІОТ 47

3.1 Опис архітектури та інструментів розробки 47

3.2 Налаштування системи 56

3.3 Демонстрація роботи 58

3.4 Висновки до розділу 62

ВИСНОВКИ 63

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64

ВСТУП

Актуальність теми. Інтернет речей (Internet of Things, IoT) по суті є надмережею, сформованою з фізичних об’єктів, що мають унікальні ідентифікатори та здатні взаємодіяти між собою без прямого втручання людини. Нині до IoT належать мільярди підключених до Інтернету пристроїв по всьому світу. Поняття «інтернет речей» уже давно вийшло за межі простого набору датчиків, які передають показники на приймальний пристрій: сенсори інтегруються в єдині інфраструктури, де здійснюються обмін даними, їх оброблення, аналітика й керування. У результаті формується автономна система, що потребує мінімальної участі оператора або взагалі обходиться без неї, самостійно приймаючи рішення щодо керування окремим об’єктом чи сукупністю об’єктів залежно від призначення мережі.

Мета і завдання дослідження. Метою є дослідження та створення системи для моніторингу й керування елементами периферійних (граничних) пристроїв IoT, покликаної спростити роботу з IoT-мережею та її компонентами. Для досягнення мети необхідно:

- проаналізувати сучасні підходи до управління елементами граничних пристроїв інтернету речей;
- розглянути архітектуру IoT-мереж, відповідні технології та протоколи передавання даних;
- дослідити апаратну складову та виконати її налаштування;
- розробити програмне забезпечення для підвищення ефективності керування IoT-елементами;
- провести оцінювання якості створеного програмного забезпечення.

Об’єкт дослідження — система моніторингу та керування елементами інтернету речей.

Предмет дослідження — засоби та протоколи обміну даними в IoT-

мережах, а також підходи до моніторингу й управління їхніми складовими.

Практичне значення отриманих результатів. Виконано налаштування апаратних компонентів і створено програмне забезпечення, що забезпечує моніторинг і дистанційне керування граничними пристроями інтернету речей.

РОЗДІЛ 1. ОГЛЯД АРХІТЕКТУРИ МЕРЕЖІ ІНТЕРНЕТУ РЕЧЕЙ

Термін «Інтернет речей» (Internet of Things, IoT) з’явився завдяки Кевіну Ештону, який у 1997 році вперше запропонував використати технологію радіочастотної ідентифікації (RFID) для оптимізації процесів постачання товарів. Саме ця ідея привела його до Массачусетського технологічного інституту, де він разом із колегами створив дослідницький центр Auto-ID Center, що став відправною точкою розвитку концепції IoT. Згодом поняття Інтернету речей вийшло за межі RFID-технологій і перетворилося на широку екосистему взаємопов’язаних пристроїв.

Першим прикладом функціонування Інтернету речей вважається автомат із газованими напоями, який у 1982 році був підключений до мережі в Університеті Карнегі-Меллон. Відтоді кількість підключених пристроїв поступово зростала: у 1991 році компанія Hewlett-Packard презентувала перший мережевий принтер, а вже у 1993 році до інтернету приєднали першу веб-камеру в Кембриджському університеті. Значним поштовхом для розвитку IoT стала поява у 1998 році організації Bluetooth SIG, яка сприяла поширенню енергоефективного бездротового зв’язку, що дозволило пристроям працювати автономніше.

У 2000 році компанія HP розробила концепцію «всепроникної комп’ютеризації» (Cooltown), яка передбачала поєднання обчислювальних і комунікаційних технологій для створення середовища, де люди, місця та об’єкти взаємодіють через Інтернет. Уже у 2005 році Міжнародний союз електрозв’язку (МСЕ) опублікував перший звіт, що містив прогнози щодо подальшого розвитку Інтернету речей. А поява у 2008 році ефективних напівпровідникових світлодіодів започаткувала ідею «розумного» освітлення, яка згодом еволюціонувала у концепцію «розумного дому».

На сьогодні аналітики прогнозують, що протягом найближчих 5–7 років технологія Інтернету речей охопить практично всі сфери — від промисловості та бізнесу до охорони здоров’я і побутових послуг, ставши

ключовим елементом цифрової трансформації сучасного суспільства.

1.1 Екосистема інтернету речей

До екосистеми Інтернету речей (IoT) належать платформи, сервіси та технологічні складові, що забезпечують створення й роботу IoT-рішень. Їх доцільно групувати так:

- Шар сприйняття (сенсори та вимірювальні модулі). Сюди входять вбудовані контролери й плати, операційні системи реального часу (RTOS), системи безперервного/резервованого живлення, а також мікроелектромеханічні системи (MEMS).
- З'єднання на рівні сенсорів. Канали короткої дальності (від фактично 0 см до ≈100 м), орієнтовані на малопотужний і низькошвидкісний обмін між датчиками; нерідко реалізовані без повного IP-стеку.
- Локальні мережі (LAN). Переважно IP-орієнтовані сегменти (зокрема 802.11 Wi-Fi для високошвидкісного радіозв'язку), з типовими топологіями «зірка» або peer-to-peer.
- Проміжна інфраструктура (edge/fog): агрегатори, маршрутизатори, шлюзи, пограничні пристрої. Це як апаратні компоненти (процесори, DRAM, накопичувачі, модулі та пасивні елементи, тонкі клієнти, стільникові та інші радіомодулі), так і ПЗ міжплатформного рівня, платформи туманних обчислень, інструменти для edge-аналітики, засоби захисту edge-сегментів і системи керування сертифікатами (PKI).
- Глобальна мережна інфраструктура. Оператори мобільного та супутникового зв'язку, а також провайдери мереж великого радіусу дії з низьким енергоспоживанням (LPWAN).

На транспортному рівні в IoT поширені легковагові протоколи обміну даними MQTT і CoAP; у частині сценаріїв застосовують також HTTP.

- Хмара. Моделі IaaS і PaaS, керовані СУБД, сервіси потокової та пакетної обробки, інструменти аналітики, рішення класу SaaS, платформи озер даних, програмно-визначені мережі/функції (SDN/NFV) та сервіси машинного навчання.

- Аналітичні сервіси. Значні масиви телеметрії зазвичай передаються у хмару, де потребують комплексної обробки подій (CEP), аналітики та методів ML для отримання практичної цінності.
- Безпека. Питання кіберзахисту виникають на кожному рівні—від сенсора й апаратури до радіоканалів і мережевих протоколів. Необхідно забезпечити цілісність, достовірність і конфіденційність наскрізно, адже IoT є привабливою ціллю для атак і не терпить «слабких ланок».

Узагальнено елементи IoT поділяють на чотири великі блоки [1–2]: сервіси (Services), апаратна частина (Hardware), набір правил (Rules) та програмна частина (Software). Кожен із них може деталізуватися на підгрупи за призначенням (див. рис. 1.1).

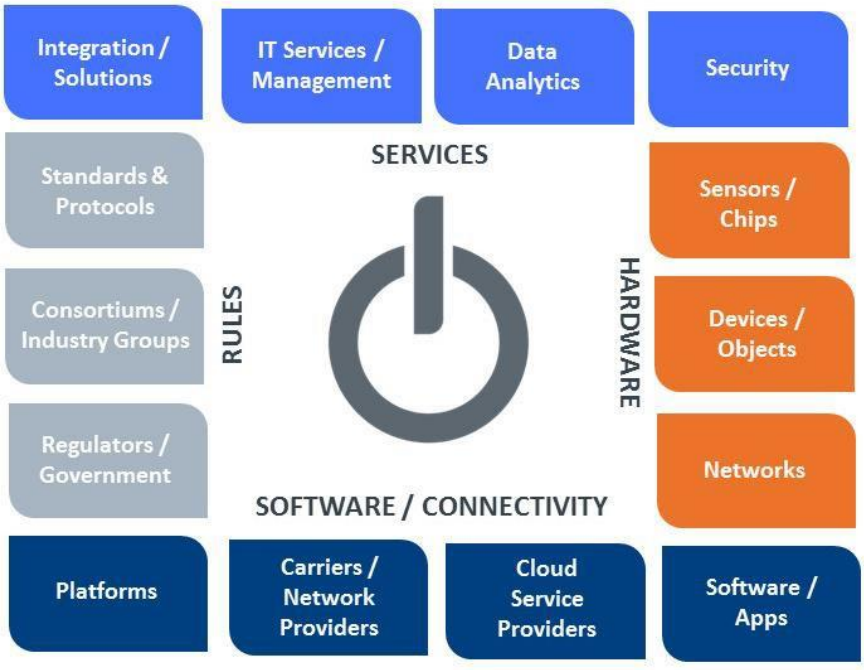


Рис. 1.1. зв’язок між елементами інтернету речей

1.2 Огляд архітектури інтернету речей

Конкретна побудова систем Інтернету речей залежить від призначення та способу впровадження, однак у загальному вигляді вона наближається до архітектури класичних АСУ ТП (автоматизованих систем

керування технологічними процесами) (див. рис. 1.2).

Отримання інформації з фізичного світу та вплив на нього забезпечують сенсори (sensors) і виконавчі механізми (actuators) — подібно до контурів керування в АСУ ТП для будь-якого об’єкта. Сукупність цих пристроїв разом із засобами підключення до мережі та інтеграції з рівнем оброблення подій формує периферійний (edge) шар.

Потоки подій/даних, що надходять із периферії, накопичуються та опрацьовуються відповідно до поставлених завдань на рівні оброблення подій і аналітики (event processing, platform).

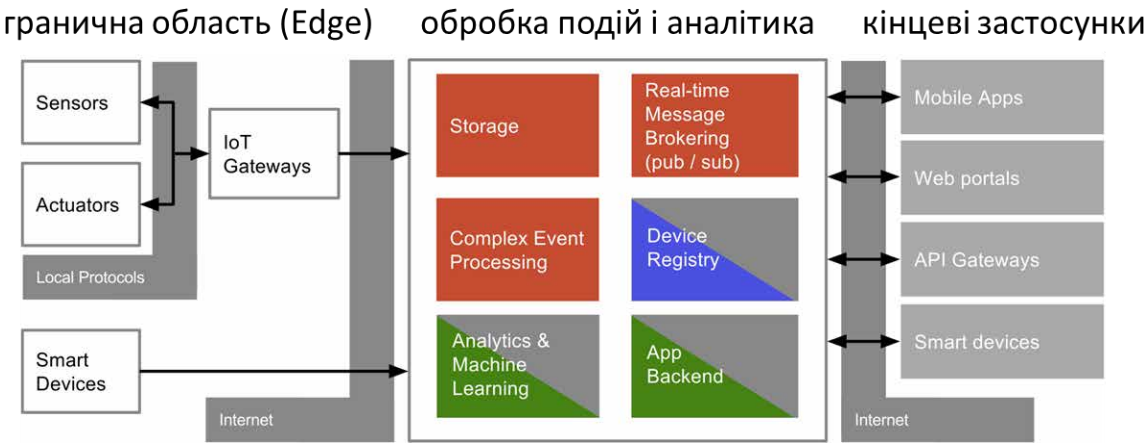


Рис. 1.2. Приклад архітектури IoT

Потоки даних, що надходять із периферійного шару (edge), далі накопичуються та опрацьовуються відповідно до поставлених завдань. На цьому рівні реалізують: сховище даних (storage), оброблення подій (event processing), а також доставку повідомлень у реальному часі та потокову обробку (real-time message brokering, stream processing) для передавання інформації прикладним сервісам. Додатково тут функціонують засоби обліку та адміністрування пристроїв (device registry, edge device management), що походять із граничної області.

На основі отриманих подій запускають моделі машинного навчання (machine learning), які разом з інструментами аналітики (analytics)

дозволяють робити висновки про стан і поведінку об’єкта. Типово цей рівень розгортають у середовищах хмарних (cloud) або туманних (fog) обчислень. За аналогією з АСУ ТП, це відповідає рівню контролерів і SCADA (без частини, що стосується HMI).

Доступ до результатів, моніторинг, віддалене керування та адміністративні операції здійснюються через кінцеві застосунки з використанням мережі Internet.

На рис. 1.3 наведено сервісно-орієнтоване подання близької за змістом архітектури. Область Edge репрезентована як шар сенсорики, до неї також віднесено Device Hub/Gateway (збирання й маршрутизація даних) та Device Management (керування пристроями), причому частина цих функцій може виконуватись як у хмарі, так і на периферійних вузлах. Усі операції зі збереження та первинного оброблення подій/даних зведено до блоку Data Management. Решта обчислювальних функцій, зокрема аналітичні, подані як PaaS-додатки, що взаємодіють із сервісами керування даними через API (Application Programming Interface).

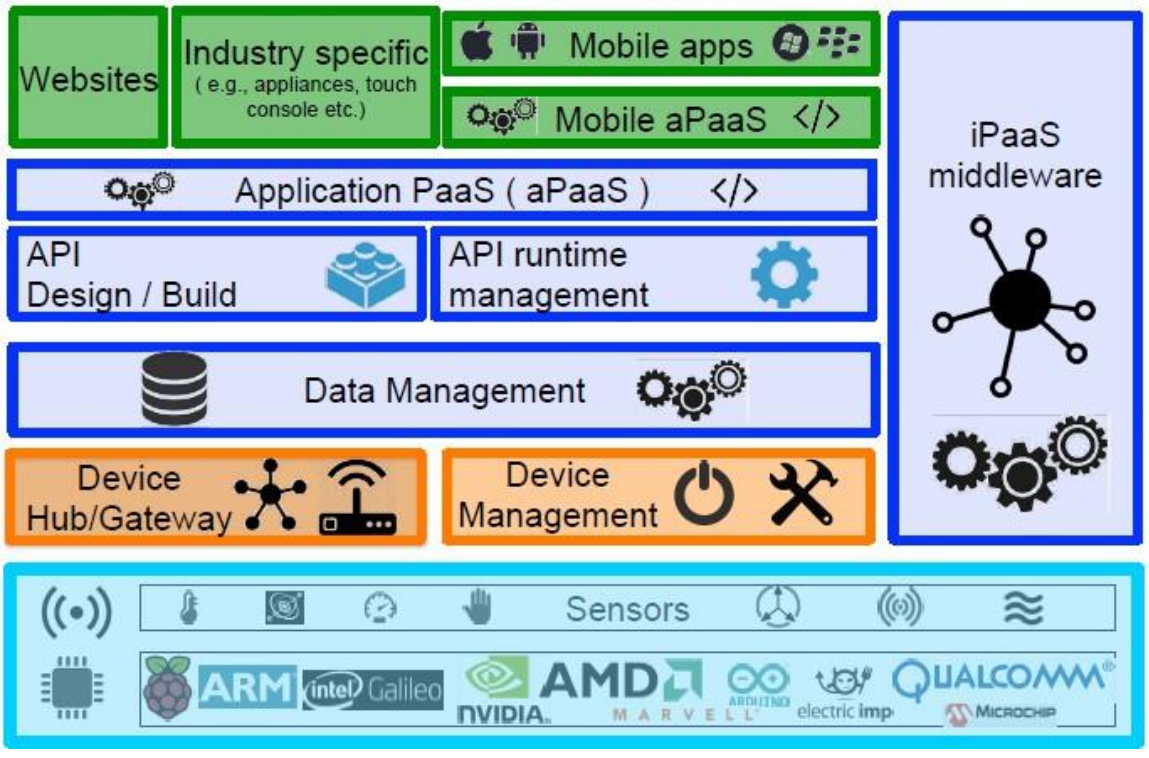


Рис. 1.3. Архітектура інтернету речей представлена у вигляді сервісів

На рис. 1.4 подано ще один варіант архітектури Інтернету речей на модульному рівні. З ілюстрації видно, що всі розглянуті підходи мають спільні ознаки: трирівневу структуру, схожі функціональні блоки, використання хмарних сервісів і застосування Інтернету як інтеграційного шару.

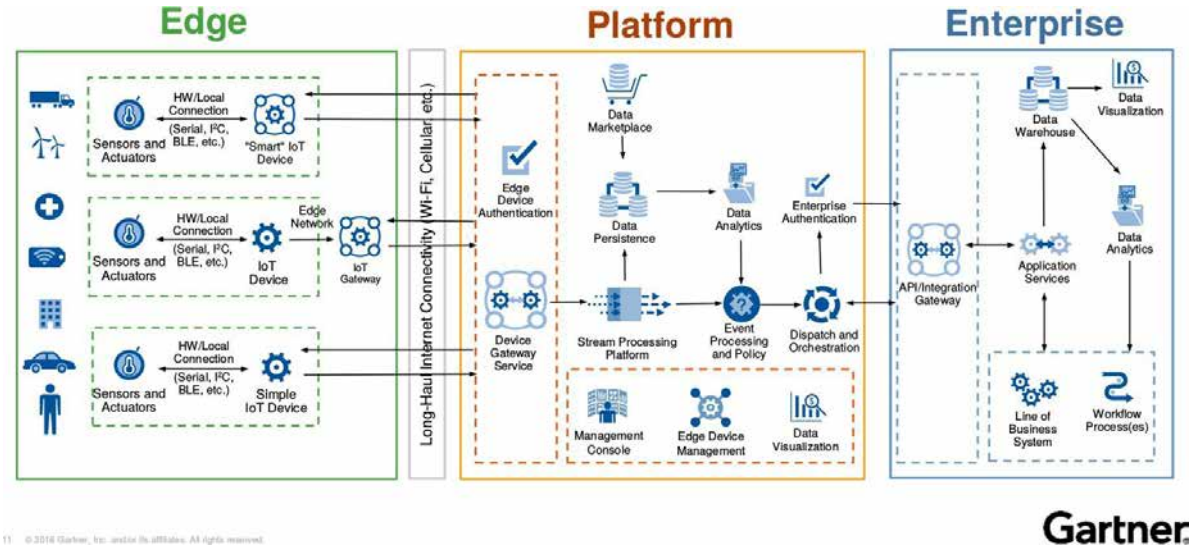


Рис. 1.4. Архітектура інтернету речей на рівні модулів

1.3 Огляд компонентів архітектури інтернету речей

1.3.1. Датчики та живлення.

Початковою й кінцевою точкою взаємодії систем Інтернету речей із реальним світом є подія: це може бути ледь помітний рух, зміна температури чи спрацювання механічного важеля, що замикає замок. На відміну від більшості класичних IT-систем, IoT завжди прив’язаний до фізичного тригера та породжує реакцію на конкретний фактор довкілля (див. рис. 1.5). Обсяг породжених сенсорами даних може кардинально відрізнятися: інколи один акустичний сенсор для превентивного обслуговування обладнання генерує величезні масиви телеметрії, тоді як в інших сценаріях достатньо одного біта, щоб повідомити критично важливий параметр, наприклад, про стан пацієнта. За останні роки сенсорні системи істотно еволюціонували: відповідно до закону Мура відбулася глибока мініатюризація (до мікро- та нанорівня) і суттєве здешевлення елементної бази. Саме ці тенденції лежать

в основі прогнозів про підключення мільярдів пристроїв до IoT — і пояснюють, чому такі прогнози є реалістичними. Безперервність роботи сенсорики забезпечує живлення, вибір якого визначається сценарієм застосування (автономні джерела, резервування, енергоощадні режими), що є критично важливим для надійності всього IoT-контуру.

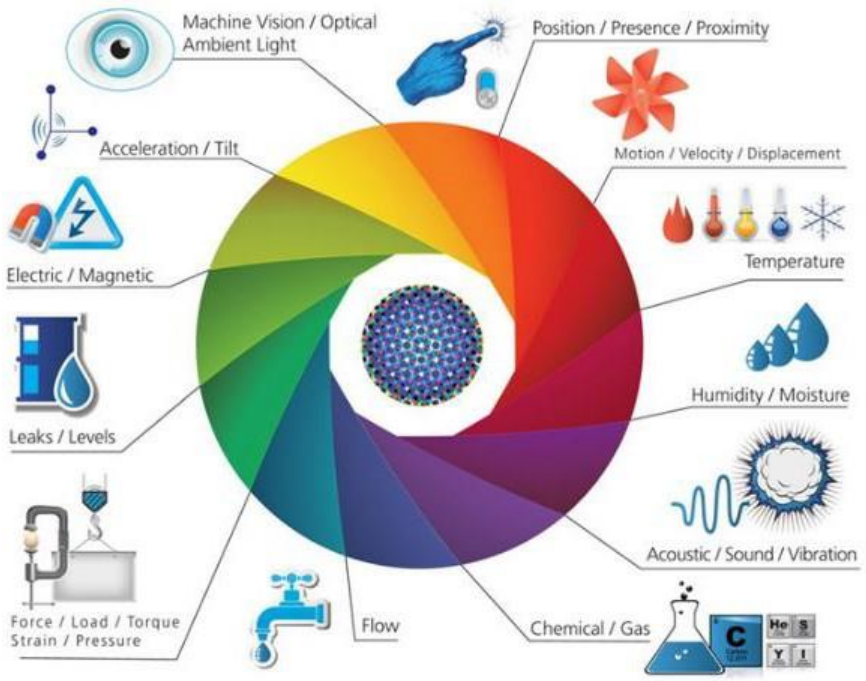


Рис. 1.5. Схематичне зображення сил та явищ, для обробки яких може використовуватись інтернет речей

Розглядаючи Інтернет речей, необхідно враховувати мікроелектромеханічні системи, сенсори та інші недорогі периферійні (граничні) пристрої разом із їхніми електрофізичними характеристиками. Водночас слід зважати на силові й енергетичні підсистеми, що забезпечують ці edge-вузли живленням. Не слід припускати, що енергія для периферії «наявна за замовчуванням»: навіть малі за розмірами датчики, коли їх мільярди, у сукупності потребують значних енергоресурсів. Питання енергоживлення безпосередньо пов’язані також із проєктуванням і організацією хмарних сервісів IoT [1].

1.3.2. Передача даних. Під час проєктування систем IoT

першочергову увагу приділяють встановленню з'єднань і роботі мереж. Інтернет речей не міг би функціонувати без надійних засобів доставки інформації з найвіддаленіших і найскладніших умов у масштабні центри обробки даних таких компаній, як Google, Amazon, Microsoft та IBM. Оскільки у словосполученні «Інтернет речей» ключовим є саме «інтернет», необхідно враховувати аспекти мережевих технологій, протоколів обміну й навіть основ теорії сигналів. Фундаментом IoT є не стільки сенсори чи застосунки, скільки здатність забезпечити стійке підключення.

Організація каналу передавання та мережевого з'єднання часто спирається на системи зв'язку ближньої дії — персональні мережі (PAN), які нерідко працюють без повного IP-стеку. Це можуть бути як дротові, так і бездротові рішення. До поширених бездротових технологій для IoT відносять Bluetooth, різновиди mesh-мереж, Zigbee та Z-Wave (див. рис. 1.6), а для промислових застосувань (IIoT) — також WirelessHART і ISA100 [2].

1.3.3.

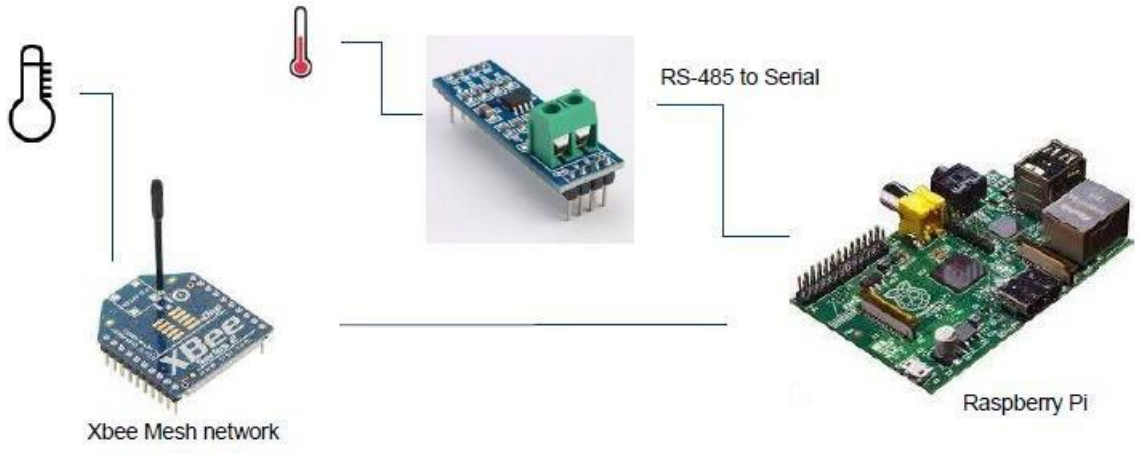


Рис. 1.6. Приклад пристроїв передачі даних в мережі PAN

Окрім PAN, у системах IoT використовують бездротові локальні мережі та IP-орієнтовані канали зв'язку: широкий спектр Wi-Fi на основі сімейства IEEE 802.11, а також 6LoWPAN і технологію Thread. Часто задіюють і стільникові телекомунікації (3G, 4G LTE), разом із новими профілями, спеціально орієнтованими на Інтернет речей та міжмашинну взаємодію, зокрема LTE Cat-1 і Cat-NB (NB-IoT). Додатково застосовують

власницькі LPWAN-рішення — LoRaWAN і Sigfox — розроблені саме для потреб IoT (рис. 1.7).

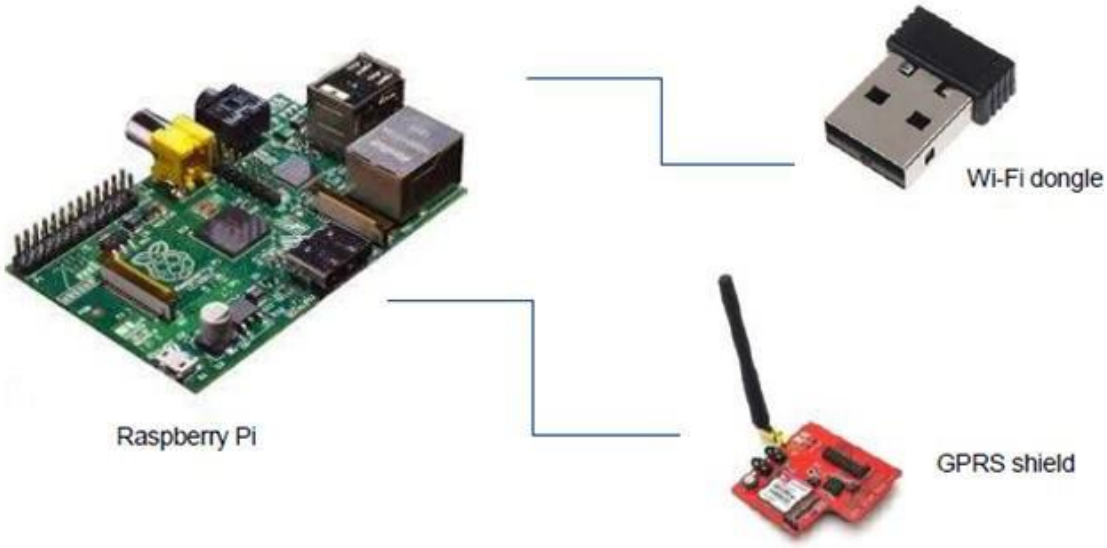


Рис. 1.7. Приклади пристрої для передачі даних в мережі LAN

1.3.4. Маршрутизація. Для винесення даних від сенсорів у інтернет-простір задіюють дві ключові складові: шлюзовий маршрутизатор та базові мережеві протоколи, що забезпечують ефективний обмін. Маршрутизатор критично важливий для безпеки, керування і маршрутизації потоків. Граничні маршрутизатори (edge routers) моніторять стан підлеглих mesh-мереж, вирівнюють і підтримують якість даних. Особлива увага приділяється приватності й захищеності інформації. Маршрутизатор також слугує інструментом побудови VPN, VLAN і програмно визначених глобальних мереж (SD-WAN). Такі інфраструктури можуть охоплювати тисячі вузлів, що обслуговуються єдиним граничним маршрутизатором, який фактично розширює межі хмари (edge device) [2].

На цьому рівні застосовується набір протоколів, необхідних для взаємодії вузлів, маршрутизаторів і хмарних сервісів у межах IoT-системи. Поява IoT дала поштовх новим спеціалізованим протоколам, які працюють поряд із традиційними HTTP та SNMP, відомими вже десятиліттями. Для передавання IoT-даних потрібні енергоощадні, малозатримкові та захищені рішення, здатні безпечно транспортувати інформацію в хмару і назад. Серед

поширених варіантів — MQTT, AMQP та CoAP.

1.3.5. Туманні і граничні обчислення, аналітика і машинне навчання. На цьому етапі слід визначити стратегію оброблення потоку даних, що надходить із крайового вузла (Edge Device) у хмарну інфраструктуру. Щоб коректно прогнозувати масштабування й подальший розвиток системи, потрібно розуміти особливості архітектур хмарних рішень і вплив затримок на роботу IoT. При цьому не вся телеметрія має прямувати в хмару: суцільне передавання часто дорожче за локальне опрацювання на межі мережі (edge computing) або за перенесення частини функцій у зону, суміжну з хмарою (fog computing). Для туманних обчислень уже запропоновано стандарти, зокрема архітектуру OpenFog.

Дані, що виникають унаслідок перетворення аналогових впливів на цифрові сигнали, можуть бути масивними або «важкими». Тут у роботу вступають аналітичні сервіси та процесори правил IoT-платформи. Рівень складності впровадження залежить від цільового сценарію. У простому випадку достатньо розмістити на крайовому маршрутизаторі легкий рушій правил, який наглядає за, скажімо, аномальними стрибками температури на кількох датчиках. У складнішому — великі масиви структурованих і неструктурованих потоків у реальному часі надходять до хмарного озера даних; це вимагає високої продуктивності для прогнозної аналітики та побудови довгострокових прогнозів на основі розвинених моделей машинного навчання, наприклад рекурентних нейронних мереж із урахуванням часових кореляцій у сигналі.

Аналітичні виклики розв’язують різними методами: застосуванням складних обробників подій (CEP), байєсівських мереж, а також проєктуванням і навчанням нейронних мереж під специфіку предметної області.

1.3.6. Загрози і безпека в IoT. Чимало IoT-рішень працюватимуть не в захищених межах дому чи офісу, а в публічних просторах, на віддалених локаціях, у рухомому транспорті й навіть усередині людського тіла. Через

таке розміщення Інтернет речей формує надвеликий, єдиний периметр для різних типів кібератак. Уже зафіксовано численні цілеспрямовані атаки на IoT-пристрої, добре організовані злами та навіть уразливості, що зачіпають безпеку на національному рівні. Розробник IoT має розуміти природу цих слабких місць і способи їх нейтралізації та застосовувати стандартні заходи кіберзахисту як до всієї IoT-інфраструктури, так і до кожного її мережевого компонента.

1.4 Огляд технологій та протоколів передачі даних у мережі інтернет речей

1.4.1 Технології та протоколи передачі на малі відстані. Оскільки парадигма Інтернету речей передбачає близьке розташування вузлів і їхню роботу без постійного джерела живлення, для таких пристроїв потрібні спеціальні, енергоощадні технології зв’язку та відповідні протоколи передавання даних. Нижче наведено кілька нині поширених рішень:

Z-Wave — радіотехнологія субгігагерцового діапазону (до ≈1 ГГц), придатна для швидкої передачі простих команд із низькою затримкою;

NFC — обмін даними на дуже малій відстані, зазвичай до 20 см;

RFID — радіочастотна ідентифікація об’єктів за допомогою міток;

BLE — варіант Bluetooth із зниженим енергоспоживанням;

Wi-Fi HaLow — 802.11ah у районі 900 МГц; споживання енергії подібне до Bluetooth, а швидкість обміну вища.

Далі розглянемо технології детальніше, та визначимо, для чого їх потрібно.

1.4.1.1 Z-Wave. Бездротова радіотехнологія з низьким енергоспоживанням

Z-Wave працює в субгігагерцовому діапазоні (до приблизно 1 ГГц) і оптимізована для передавання простих керувальних команд із малими

затримками — наприклад, перемикання каналів чи вмикання/вимикання приладів [3]. Частотну смугу обрано так, щоб зменшити вплив перешкод від уже поширених технологій на 2,4 і 5 ГГц, зокрема Wi-Fi. Попри внутрішню складність, користуватися Z-Wave нескладно; вона вирізняється енергоощадністю та зручністю в повсякденних сценаріях. Система, що застосовує малопотужні радіосигнали для дистанційного керування, зазвичай забезпечує стабільне покриття в межах помешкання: сигнал проходить крізь стіни, міжповерхові перекриття та меблі, що дає змогу охопити більшість зон будинку.

1.4.1.2 NFC. Near Field Communication (NFC) — це технологія зв’язку малого радіуса дії, призначена для обміну різними даними (зображеннями, музичними файлами, контактами, ключами цифрової автентифікації) між двома пристроями з підтримкою NFC, що розташовані дуже близько один до одного. Носіями можуть бути смарткартки, портативні гаджети, а також RFID-зчитувачі; технологія може слугувати електронним ключем доступу до сервісів і даних [4].

Обмін можливий не лише в напрямку від активного пристрою до пасивного, а й між двома активними пристроями. NFC використовують для взаємодії з системами радіочастотної ідентифікації (RFID). Щоб забезпечити сумісність карток RFID і мобільних телефонів різних виробників, перевіряють відповідність цифрового протоколу та вимірюють ключові радіочастотні параметри: часові характеристики, чутливість і амплітуди приймача в активному режимі, а також несучу частоту й рівень сигналу.

Під час передавання від активного до пасивного пристрою застосовують амплітудну маніпуляцію (ASK). У режимі взаємного обміну обидва пристрої діють на рівних і мають власні джерела живлення, тому після завершення передачі несучу одразу вимикають (рис. 1.8).

Рис. 1.8. Принцип роботи NFC.

У разі індуктивного зв’язку між зчитувачем (активним пристроєм) і міткою/відповідачем (пасивним пристроєм) пасивний елемент впливає на активний, змінюючи своє навантаження. Це призводить до модифікації амплітуди або фази напруги на антені зчитувача. Після цього встановлюється сеанс взаємодії та відбувається обмін даними. Якщо відстань між пристроями перевищує приблизно 20 см, індуктивний зв’язок зникає, канал розривається і передавання інформації автоматично припиняється.

Фактично NFC є еволюційним продовженням уже добре відомої технології RFID [18]. RFID ми зустрічаємо повсюдно — у безконтактних картках і мітках. Водночас NFC здатна не лише зчитувати дані з пасивних тегів, а й забезпечувати двосторонній бездротовий обмін між двома активними пристроями.

1.4.1.3 RFID. Радіочастотна ідентифікація (Radio Frequency Identification, RFID) — це спосіб автоматичного розпізнавання об’єктів, за якого дані, що зберігаються у транспондері (RFID-мітці), зчитують або записують за допомогою радіосигналів [19]. Типова RFID-система складається із зчитувача (ридера) та транспондера (мітки, також уживають назву «RFID-тег») [5]. Більшість міток мають дві основні складові: інтегральну схему, що відповідає за зберігання й оброблення інформації та модуляцію/демодуляцію радіосигналу, і антену для прийому та

передавання. Для роботи системи потрібне програмне забезпечення, яке збирає й аналізує дані, одержані від міток.

За способом живлення розрізняють два типи:

- активні мітки містять власне джерело енергії, тому самі випромінюють сигнал і можуть зчитуватися з більшої відстані;
- пасивні мітки не мають батареї, активуються полем зчитувача і під час опитування повертають записану в них інформацію.

RFID-мітки (рис. 1.9) використовують для управління товарними запасами та, наприклад, для фіксації часу на спортивних змаганнях [5]. Вони не витісняють штрих-коди, а доповнюють їх можливістю дистанційного зчитування. Такі мітки застосовують для маркування великої рогатої худоби з метою обліку ветеринарних оглядів; у транспорті — для ідентифікації автомобілів навіть на високих швидкостях; авіакомпанії використовують їх для відстеження значних потоків багажу. RFID також інтегрують у біометричні паспорти та платіжні картки для безпечного доступу до захищених зон.

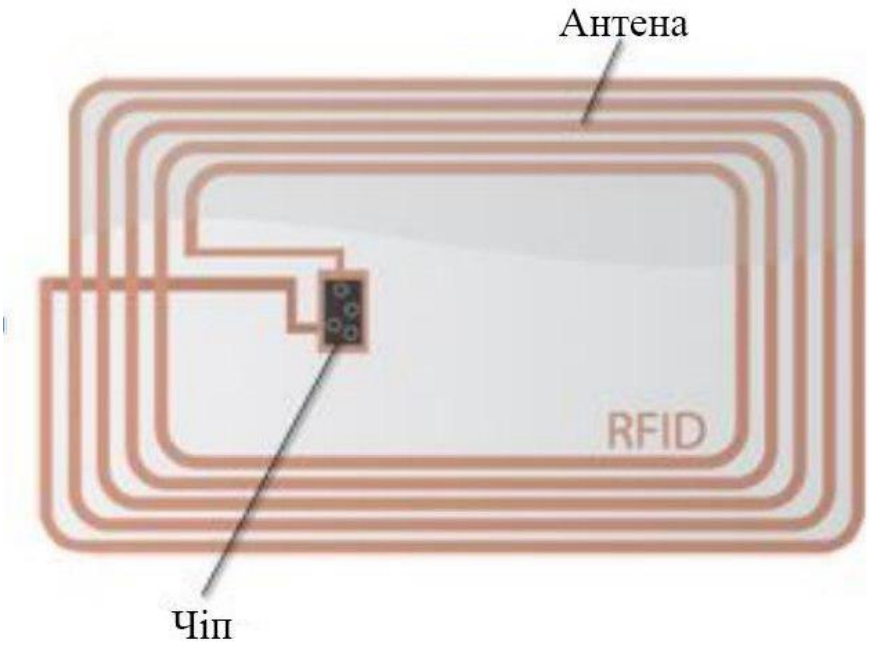


Рис. 1.9. Будова пасивної RFID мітки.

1.4.1.4 BLE, Bluetooth Low Energy. Бездротова технологія Bluetooth Low Energy (BLE) — це складова специфікації Bluetooth, що з’явилася, починаючи з версії 4.0, і далі розвивалася до Bluetooth 5.0 [6].

Пристрої з підтримкою BLE споживають набагато менше енергії, ніж рішення попередніх поколінь класичного Bluetooth. У багатьох випадках такі модулі здатні працювати понад рік від невеликої «таблетки» без підзарядки. Це відкриває змогу використовувати компактні датчики, які постійно активні та можуть взаємодіяти з іншими пристроями.

BLE орієнтовано на малогабаритні пристрої, для яких критичні компактність та економність живлення і де неможливо встановити повноцінний акумулятор чи батарею великого об’єму [6].

Порівняно з класичним Bluetooth, Bluetooth LE зазвичай споживає у 10–20 разів менше енергії, забезпечує істотно вищі швидкості передавання (у десятки разів) і може підтримувати дальність понад 100 метрів. До цього додаються підвищені показники безпеки та надійності, мала затримка під час встановлення з’єднання і низька споживча потужність.

Важливою рисою стандарту є адаптивне перестроювання робочої частоти: технологія швидко змінює канал, відшукуючи найменш зашумлену смугу, що знижує вплив завад, проблеми з перевантаженням та інтерференцію.

Специфікацію Bluetooth 5.0 спроєктовано з акцентом на потреби Інтернету речей, що фактично позиціонує її як один із базових стандартів для IoT. Порівняно з версією 4.0, зросла швидкість передавання даних — до рівня, співставного з ранніми реалізаціями HSPA/LTE, при збереженні низького енергоспоживання. Енергоефективність є ключовим параметром для мереж IoT. Попри те, що специфікація ще відносно нова і поширена не повсюдно, вона зберігає зворотну сумісність, тож імовірно протягом кількох років більшість мобільних пристроїв підтримуватимуть п’яте покоління стандарту, що є суттєвою перевагою порівняно з альтернативними технологіями.

1.4.1.5 Wi-Fi HaLow. Бездротовий протокол, затверджений у 2017 році як поправка до сімейства IEEE 802.11 [7], працює в неліцензованому субгігагерцовому діапазоні близько 900 МГц. Перенесення Wi-Fi у цей спектр забезпечує значно більшу зону покриття порівняно з типовими мережами на 2,4 та 5 ГГц. Ще одна ключова риса — низьке енергоспоживання, що дає змогу формувати великі кластери станцій або сенсорів, які взаємодіють і поширюють сигнал, підтримуючи сценарії Інтернету речей (IoT). За енергоефективністю протокол може конкурувати з Bluetooth, водночас пропонуючи вищі швидкості передавання даних і розширений радіус дії (рис. 1.10).

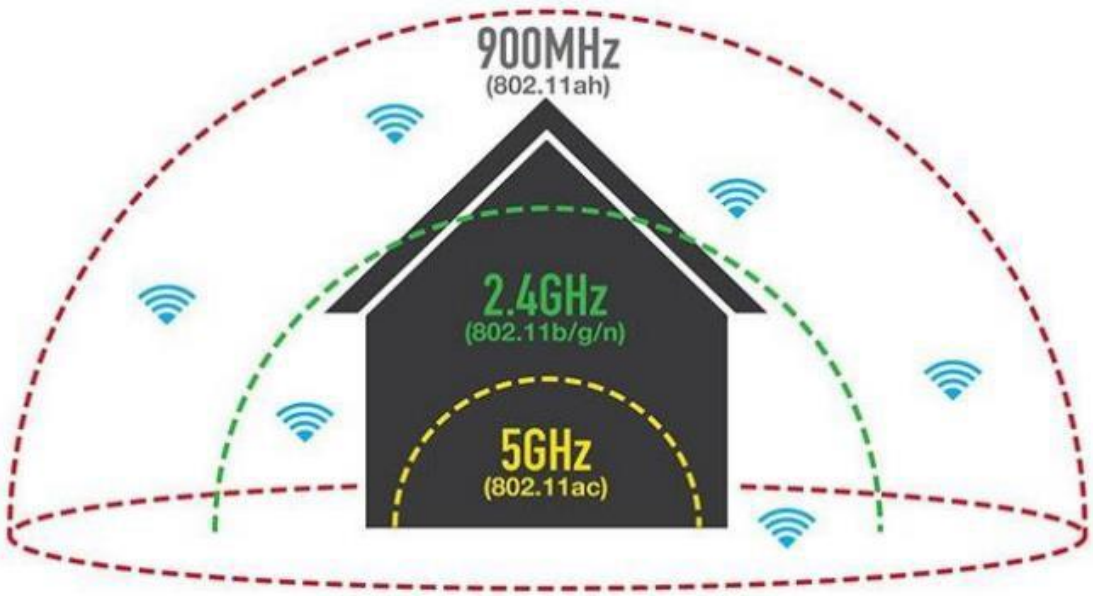


Рис. 1.10. Порівняння різних протоколів Wi-Fi за зоною покриття

Wi-Fi HaLow переносить технологію Wi-Fi у субгігагерцовий діапазон близько 900 МГц, завдяки чому стає можливим підключення малопотужних вузлів — сенсорів, носимих та портативних пристроїв. Стандарт зберігає ключові переваги попередніх поколінь: надійний захист даних, широку взаємну сумісність обладнання та простоту розгортання. Більшість пристроїв із підтримкою HaLow здатні працювати й у смугах 2,4 та 5 ГГц, тож

безперешкодно інтегруються в чинну багатомільярдну Wi-Fi-екосистему. Підтримка IP-стеку забезпечує прямий доступ до хмарних сервісів, що є критично важливим для IoT-сценаріїв. Крім того, одна точка доступу може обслуговувати приблизно до тисячі клієнтських вузлів.

1.4.1.6. Порівняльний огляд технологій і протоколів передавання даних на короткі відстані в IoT. Узагальнені характеристики мереж ближньої дії подано в табл. 1.1.

Таблиця 1.1

Порівняння основних технічних характеристик мереж з низькою дальністю дії

Характеристики	RFID	NFC	BLE	Z-Wave	Wi-Fi HaLow
Смуга частот	6/13.5/433/863-870/902-928 МГц 2.4/5-27 ГГц	13.56 МГц	2.4 ГГц	868/915 МГц	Піддіапазон 1 ГГц
Швидкість передачі даних	500 кбіт/с	106/212/424/848 кбіт/с	1 Мбіт/с	9.6,40 та 100 кбіт/с	До 4 Мбіт/с
Радіус дії	0.1 - 5 м	0.1 м	70 м	100 м	100 – 1000 м
Пропускна здатність на канал	10 МГц для 6 МГц 14 МГц для 13.5 МГц 1.74 МГц для 433 МГц 7 МГц для 800 МГц 8 МГц для 2.4 ГГц 5 – 27 ГГц сегментований	Змінна	40 каналів з шириною в 2 МГц	300,400 кГц	1/2/4/8/16 МГц
Модуляція	-	ASK, BPSK	GFSK	FSK/GFSK	BPSK, QPSK, 16-/64-/256-QAM, OFDM
Топологія	Point to Point Point to Multipoint	Peer-to-peer	Single-hop	Mesh	Star
Безпека	Шифрування	Шифрування	AES-128	AES-128	WPA

1.5 Протоколи для передачі повідомлень в мережі інтернету речей

Обсяг даних, що генерує окремий сенсорний вузол, зазвичай незначний, але більшість сервісів Інтернету речей спираються на оброблення повідомлень від великої кількості клієнтів/давачів. Це принципово відрізняє їх від типових архітектур класичних мереж.

Отже, маємо іншу модель взаємодії: багато джерел — багато одержувачів. До того ж трафік від одного вузла може коливатись від дуже малого до дуже великого. У таких умовах звичні прикладні підходи до обміну повідомленнями виявляються малопридатними.

Базовою топологією для передавання повідомлень у мережах IoT є «видавець — підписник» (publisher–subscriber, pub/sub) (див. рис. 1.11). У цій схемі видавець є джерелом даних, а підписник — їхнім отримувачем. Підписка — це дія, за якої учасник налаштовує отримання інформації від конкретного видавця і визначає параметри збору: періодичність, умови доставки та інші (залежно від реалізації) характеристики.

У розглянутому сценарії сенсорний вузол (Node) агрегує дані з кількох датчиків (наприклад, вимірювання вологості) і надсилає їх згідно з умовами підписки: за запитом або автоматично через задані інтервали. Самі датчики зазвичай прості й лише безперервно фіксують контрольовані параметри. Тому їх об’єднують у вузли на базі мікроконтролерів, які зчитують покази та передають їх далі на сервер за наперед визначеними алгоритмами.

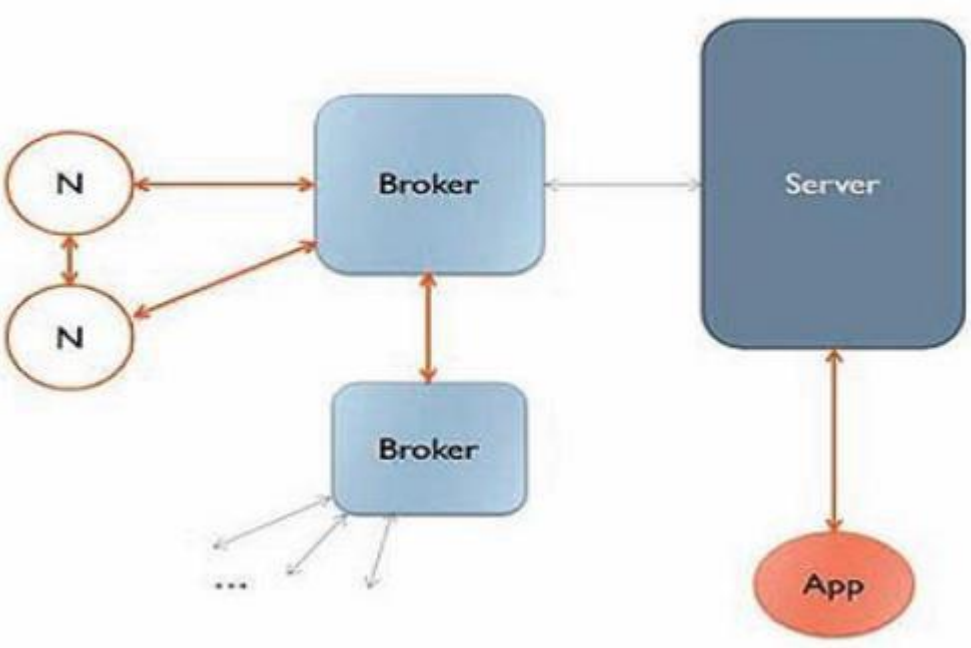


Рис. 1.11 Топологія системи, яка виконує передачу даних між різними вузлами мережі IoT

Для взаємодії користувача з системою зазвичай потрібен клієнтський застосунок, інстальований на особистому пристрої. Він наочно відображає телеметрію від датчиків або результати, уже оброблені сервером керування. У такій архітектурі передбачається наявність брокера. Брокер — це сервер, що приймає повідомлення від видавців і розподіляє їх відповідним підписникам. У складніших інсталяціях брокер може виконувати додаткові дії: попередній аналіз та перетворення даних, пріоритизацію каналів, формування черг. По суті, він відповідає за маршрутизацію, тимчасове зберігання й фільтрацію трафіку. Черга повідомлень — це буфер, у якому тимчасово утримуються повідомлення під час передавання; якщо пропускної здатності каналу бракує або одержувач недоступний у момент надсилання, повідомлення зберігається в черзі доти, доки його не буде доставлено.

У мережах IoT застосовують такі протоколи:

- DDS — прикладний M2M-протокол для систем реального часу.
- XMPP — протокол для обміну миттєвими повідомленнями та індикації присутності майже в реальному часі.
- CoAP — полегшений протокол для пристроїв і мереж з обмеженими

ресурсами.

- MQTT — надлегкий, відкритий pub/sub-протокол для передавання даних на віддалених або ненадійних ділянках мережі.
- STOMP — простий текстовий протокол взаємодії з брокером, підтримує модель запит–відповідь і роботу з чергами.
- SOAP — XML-орієнтований протокол обміну структурованими чи довільними повідомленнями у розподілених системах.

Далі розглянемо найпопулярніші протоколи.

1.5.1 DDS (Data Distribution Service). Це прикладний M2M-протокол для систем реального часу, що працює за моделлю publisher–subscriber (видавець–підписник). Його головна роль — забезпечити взаємодію між пристроями через шину обміну повідомленнями (див. рис. 1.12), причому система здатна синхронно доставляти мільйони повідомлень за секунду.

Поведінка пристроїв у таких мережах відрізняється від звичних IT-середовищ: вони працюють на дуже малих часових шкалах (до часток мікросекунд) і повинні зв’язуватися по складних маршрутах. Через це прості двоточкові TCP-потоки стають обмеженням для комунікації.

DDS натомість надає тонке керування параметрами якості обслуговування (QoS), підтримує багатоадресну доставку, налаштовувану надійність і механізми надмірності. Серед ключових переваг — ефективне «розгалуження» даних: протокол дозволяє фільтрувати й відбирати повідомлення за місцем призначення, обслуговуючи одночасно до тисяч одержувачів. Для малопотужних вузлів доступні полегшені реалізації DDS, пристосовані до обмежених ресурсів.

Щоб отримувати дані від пристроїв, топологія «зірка» виявляється неефективною. DDS реалізує прямий шиновий обмін між вузлами через так звану шину даних (DataBus) — мережевий аналог бази даних, спільний простір даних, у межах якого пристрої публікують і споживають повідомлення без жорсткої прив’язки до кінцевих точок.

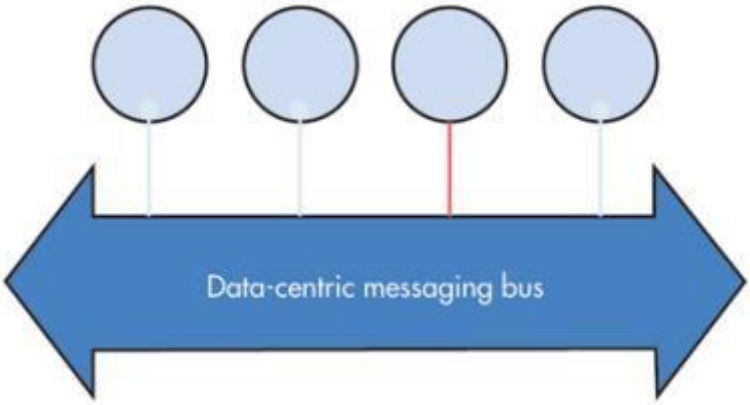


Рис. 1.12 Принцип з’єднання пристроїв за допомогою протоколу DDS

Протокол DDS застосовують там, де потрібні надвисока продуктивність і робота в режимі реального часу. Це одна з небагатьох технологій, що поєднує гнучкість, надійність і швидкодію, необхідні для створення складних систем. Серед типових сфер використання — оборонні та військові комплекси, вітрові електростанції, інтегровані лікарняні платформи, системи діагностичної візуалізації, рішення для моніторингу й супроводу активів, а також автомобільні комплекси випробувань і забезпечення безпеки. MQTT (Message Queue Telemetry Transport) — легковаговий, компактний і відкритий протокол обміну, спроєктований для передавання даних із віддалених майданчиків, де важливі мінімальний розмір клієнтського коду та обмежена пропускна здатність каналу. Такі властивості роблять його придатним для M2M-систем (машина–машина) [9–10].

Існує також варіант MQTT-SN (MQTT for Sensor Networks), раніше знаний як MQTT-S, орієнтований на вбудовані бездротові пристрої, що не використовують стек TCP/IP.

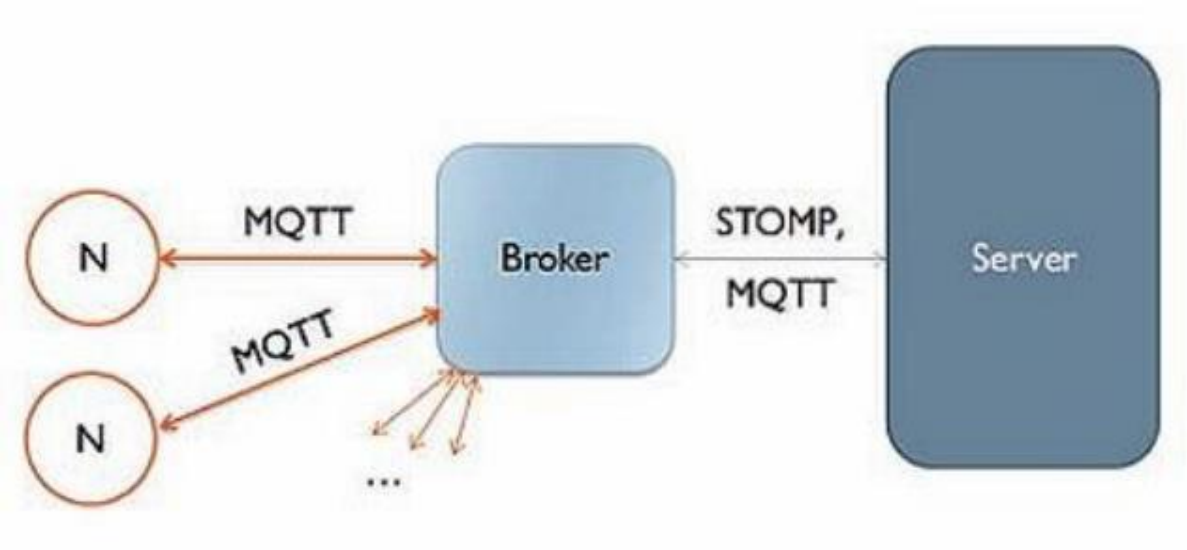
Схема роботи MQTT у спрощеному вигляді:
• Видавець публікує повідомлення з даними (наприклад, показники вологості) на брокері, зазначаючи тему (Topic), до якої належить повідомлення (скажімо, «вологість»).
• Брокер визначає, хто з підписників має активну підписку на цю тему.
• Усі підписники теми «вологість» отримують від брокера відповідне

повідомлення з оновленими значеннями.

Отже, велика кількість одержувачів може підписуватися на різні теми та отримувати потрібні дані без прямого звернення до видавця, що забезпечує розв'язування зв'язків між джерелами й споживачами інформації.

1.5.2 STOMP (Simple Text Oriented Message Protocol). Це простий текстовий протокол обміну повідомленнями, орієнтований на широку сумісність із різними мовами програмування, платформами та брокерами. Він підтримує шаблон «видавець — підписник» (див. рис. 1.13) і організовує взаємодію з брокером у логіці запит–відповідь, використовуючи такі команди, як SEND (надіслати), SUBSCRIBE (підписатися), UNSUBSCRIBE (скасувати підписку), BEGIN (розпочати), ABORT (перервати), ACK (підтвердити), NACK (не підтвердити), DISCONNECT (від'єднатися).

За стилем близький до HTTP: працює поверх TCP і має простий текстовий формат, завдяки чому клієнти STOMP можуть взаємодіяти з будь-



яким брокером, що реалізує цей протокол. Підхід створено для інтеграції систем, написаних різними мовами, і він підтримується великою кількістю клієнтських бібліотек.

Рис. 1.13. Сегмент мережі, де застосовують протоколи MQTT

STOMP

1.5.4. Огляд підтримуваних протоколами операцій і їхнього призначення показує, що кожен із них має власну нішу застосування. DDS

орієнтований на взаємодію між пристроями в режимі реального часу, коли потрібні висока пропускна здатність, робота з великими потоками даних, гнучкість і масштабованість. MQTT доцільний у перевантажених мережах із численними вузлами та брокерами, особливо за обмеженої смуги пропускання та вимоги до мінімального обсягу клієнтського коду. STOMP, своєю чергою, використовують тоді, коли важливо спростити підсистеми обміну повідомленнями та уніфікувати потоки з різнорідних протоколів.

Деталізоване зіставлення призначення й набору операцій для кожного з цих протоколів подано в табл. 1.2.

Таблиця 1.2

Призначення та операції, котрі виконуються аналізованими протоколами

Протокол	Призначення	Операції
DDS	Для розподілених систем, що потребують масштабування та балансування навантаження	Приймання, передавання й розповсюдження даних
MQTT	Для навантажених мереж із великою кількістю вузлів і наявним брокером	Процедури публікації/підписки (pub/sub) та доставляння повідомлень через брокера
STOMP	Для гетерогенних середовищ, де потрібен простий текстовий протокол обміну через брокера й інтеграція різних стеків	Публікація/підписка та транзакційні дії

1.5.5 Висновки після огляду технологій та протоколів передачі даних. Комунікацію в мережах Інтернету речей забезпечують різні технології та протоколи передавання даних, вибір яких залежить від умов експлуатації та середовища розгортання.

Для домашніх інсталяцій доцільно поєднувати Wi-Fi HaLow як

транспорт і MQTT як протокол обміну повідомленнями. Персональні мережі (PAN) раціонально будувати на BLE та RFID, які забезпечують взаємодію пристроїв на малій відстані, а для передавання команд і телеметрії використовувати MQTT і DDS.

STOMP надає уніфікований спосіб підключення до різнорідних пристроїв і абстрагує від протоколів нижчого рівня, спрощуючи доступ до сервісів.

1.6 Огляд існуючих засобів моніторингу та управління елементами інтернету речей

1.6.1 The Thing System. Створено платформу моніторингу та керування екосистемою Інтернету речей, розроблену командою інженерів зі США за участю Денні Гудмана, відомого автора посібників з JavaScript і HTML. Метою проєкту є централізоване об’єднання пристроїв «розумного дому», адже обладнання різних виробників часто несумісне й працює без узгодження. Щоб усунути цю фрагментацію, розробники створили програмне забезпечення, яке взаємодіє з кількома мережевими протоколами та слугує проміжним шаром між різнорідними гаджетами і клієнтськими застосунками. Список пристроїв, котрі підтримує проект, можна знайти на сайті проекту.

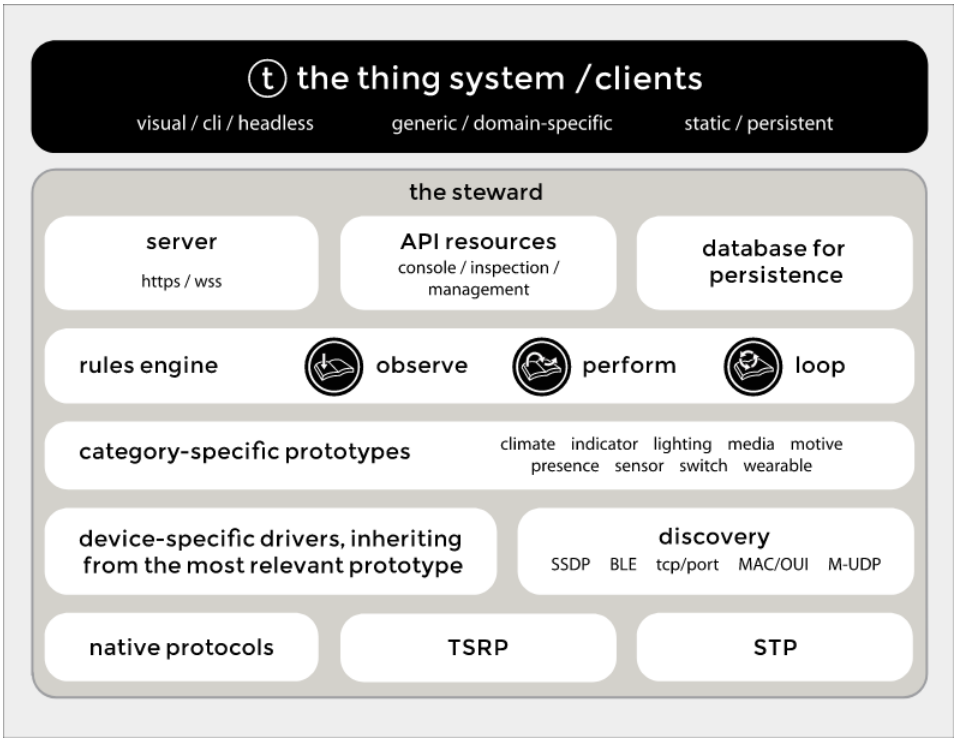


Рис. 1.14. Діаграма модулів додатку Steward

Програмне забезпечення Steward — основний компонент платформи the thing system — написане на Node.js, тож є кросплатформним і легко розширюваним. Його можна запускати як на ноутбуці, так і на віддаленому сервері чи одноплатнику Raspberry Pi.

Переваги:

- невеликий «розмір» і скромні системні вимоги;
- портативність і підтримка різних операційних систем;
- придатність для широкого кола IoT-пристроїв;
- простота встановлення та конфігурування.

Недоліки:

- орієнтація насамперед на сценарії «розумного дому»; під час масштабування до складніших мереж з’являються конфігураційні труднощі;
- можливі обмеження продуктивності при зростанні навантаження, зважаючи на вибраний стек (Node.js);

- порівняно невеликий перелік підтримуваного обладнання.

Висновок: рішення добре підходить для домашніх інсталяцій і невеликих локальних мереж, але за значного масштабування чи інтенсивних потоків даних можуть проявлятися проблеми з продуктивністю.

1.6.2 Domoticz. Платформа автоматизації «розумного дому» з відкритим вихідним кодом призначена для керування широким спектром пристроїв і для оброблення сигналів, що надходять від різноманітних датчиків. Вона сумісна з великою кількістю рішень від різних виробників. Основна частина коду написана мовою C++ і поширюється за ліцензією GNU General Public License. Користувацький інтерфейс реалізовано як HTML5 веб-застосунок із адаптивною верстkою, тож він коректно відображається на різних екранах [15].

Система є кросплатформною: працює у середовищах Windows, macOS (Apple), на платформах Cubieboard і Raspberry Pi, а також на більшості Unix-подібних операційних систем. Доступні мобільні застосунки для віддаленого доступу та керування. Користувач може задавати власну логіку оброблення подій і повідомлень, що надходять від пристроїв і сенсорів; це робиться у візуальному середовищі Blockly (див. рис. 1.15).

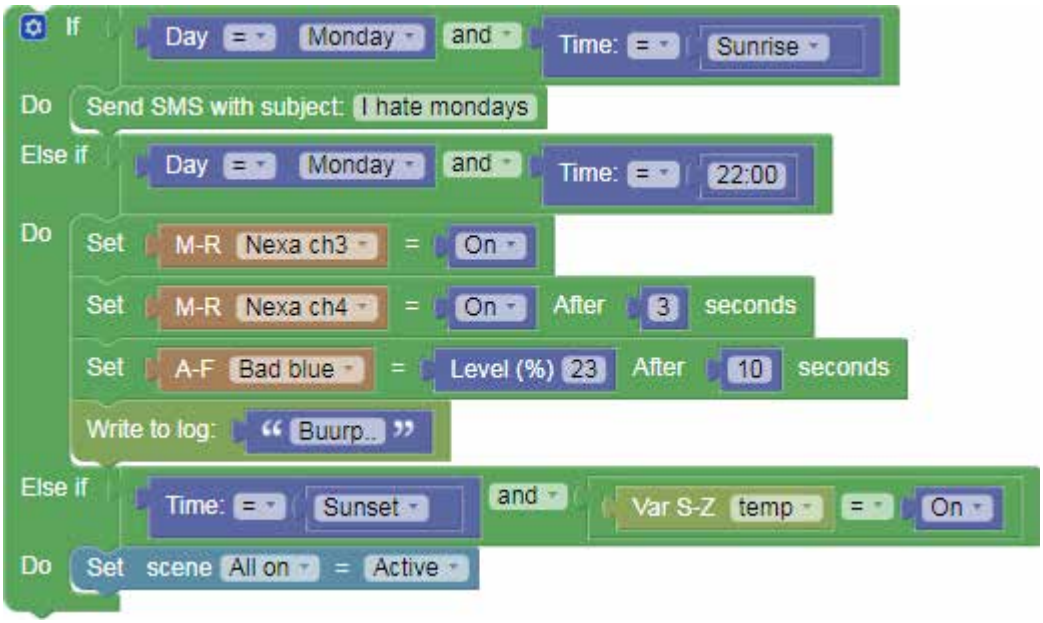


Рис. 1.15. Приклад візуального програмування з допомогою веб-додатку “Blockly”

Переваги системи:

- Працює на різних платформах.
- Має зручний вебінтерфейс.
- Підтримує широкий спектр готових пристроїв.
- Дозволяє створювати власні драйвери та скрипти для нестандартного обладнання.
- Проста у встановленні й конфігуруванні.
- Легко масштабується під різні сценарії використання.
- Дозволяє гнучко налаштовувати поведінку через логічні блоки/візуальне програмування.
- Відкритий вихідний код, тож можна самостійно вносити зміни й додавати функції.

Недоліки системи:

- Інтерфейс на HTML5 не сумісний із дуже старими браузерами.
- На пристроях із малою кількістю оперативної пам’яті можливі проблеми з продуктивністю.

Окрема перевага — керування через асинхронні запити у форматі JSON. Для всіх доступних запитів і методів наявна детальна документація, що спрощує інтеграцію та автоматизацію.

1.7 Висновок до розділу

У цьому розділі подано огляд і аналіз архітектури мереж Інтернету речей, а також підсумовано ключові протоколи, що застосовують під час їх побудови. Для ближнього радіуса взаємодії між пристроями доцільно обирати BLE та NFC, оскільки вони забезпечують оптимальний баланс між швидкістю обміну й енерговитратами. На середніх відстанях доцільним транспортом виступає Wi-Fi HaLow: за енергоспоживання, співставного з Bluetooth, він пропонує вищу пропускну здатність і більшу зону покриття.

Окремо розглянуто інструменти моніторингу та керування IoT-компонентами. Серед них виокремлюється Domoticz завдяки відкритому вихідному коду, широкій підтримці обладнання та наявності власного програмного інтерфейсу, що дає змогу організувати віддалене керування системою.

РОЗДІЛ 2. ОПИС І АНАЛІЗ АПАРАТНОЇ ЧАСТИНИ ТА НАЛАШТУВАННЯ ЗАСОБІВ МОНІТОРИНГУ ТА УПРАВЛІННЯ ЕЛЕМЕНТАМИ ГРАНИЧНИХ ПРИСТРОЇВ ІОТ

2.1 Опис вибраної платформи Raspberry Pi

Raspberry Pi Model B — це одноплатний мінікомп’ютер, який спершу створювали для шкільного навчання базових основ інформатики (рис. 2.1). Із розвитком мобільних процесорів він став однією з найпопулярніших платформ серед ентузіастів, що розробляють рішення для Інтернету речей. До ключових переваг цієї платформи належать невисока вартість, універсальність застосування та відкритість [11].

Рис. 2.1. Зовнішній вигляд плати RaspberryPi Вирізняльні риси платформи:

- Працює під керуванням Unix-подібних ОС.
- Спроможна відтворювати відео у форматі Full HD.
- Обсяг оперативної пам’яті — від 512 МБ.
- Габарити плати: 85,6 × 54 × 17 мм.

- Наявні композитний відеовихід RCA та HDMI для підключення дисплея, а також 3,5-мм аудіороз’єм для під’єднання звукових пристроїв.

Плату можна живити кількома способами. Варіант 1 — через роз’єм micro-USB із напругою 5 В. Варіант 2 — через універсальні контакти (піни). Варіант 3 — блок живлення на 5 В і 2 А. Перші два підходи можуть бути нестабільними за наявності значної кількості периферії та датчиків. Найбільш надійним є третій спосіб, оскільки він забезпечує достатній запас по струму для під’єднаних пристроїв [12].

Плата підтримує широкий набір інтерфейсів (рис. 2.2):

- UART;
- слот для SD/MMC/SDIO;
- HDMI;
- 3,5-мм стереовихід;
- композитний відеосигнал;
- 2 × USB 2.0;
- Ethernet RJ-45 10/100 Мбіт/с;
- Wi-Fi 802.11n;
- Bluetooth 4.1;
- 16 ліній вводу/виводу на 3,3 В;
- шини I2C та SPI;
- ARM JTAG;
- DSI;
- MIPI CSI-2.

Модулі бездротового зв’язку Wi-Fi 802.11n і Bluetooth 4.1 реалізуються окремим чипом Broadcom BCM43438; підключення здійснюється через інтерфейс розширення (Shield) або як зовнішні периферійні модулі.

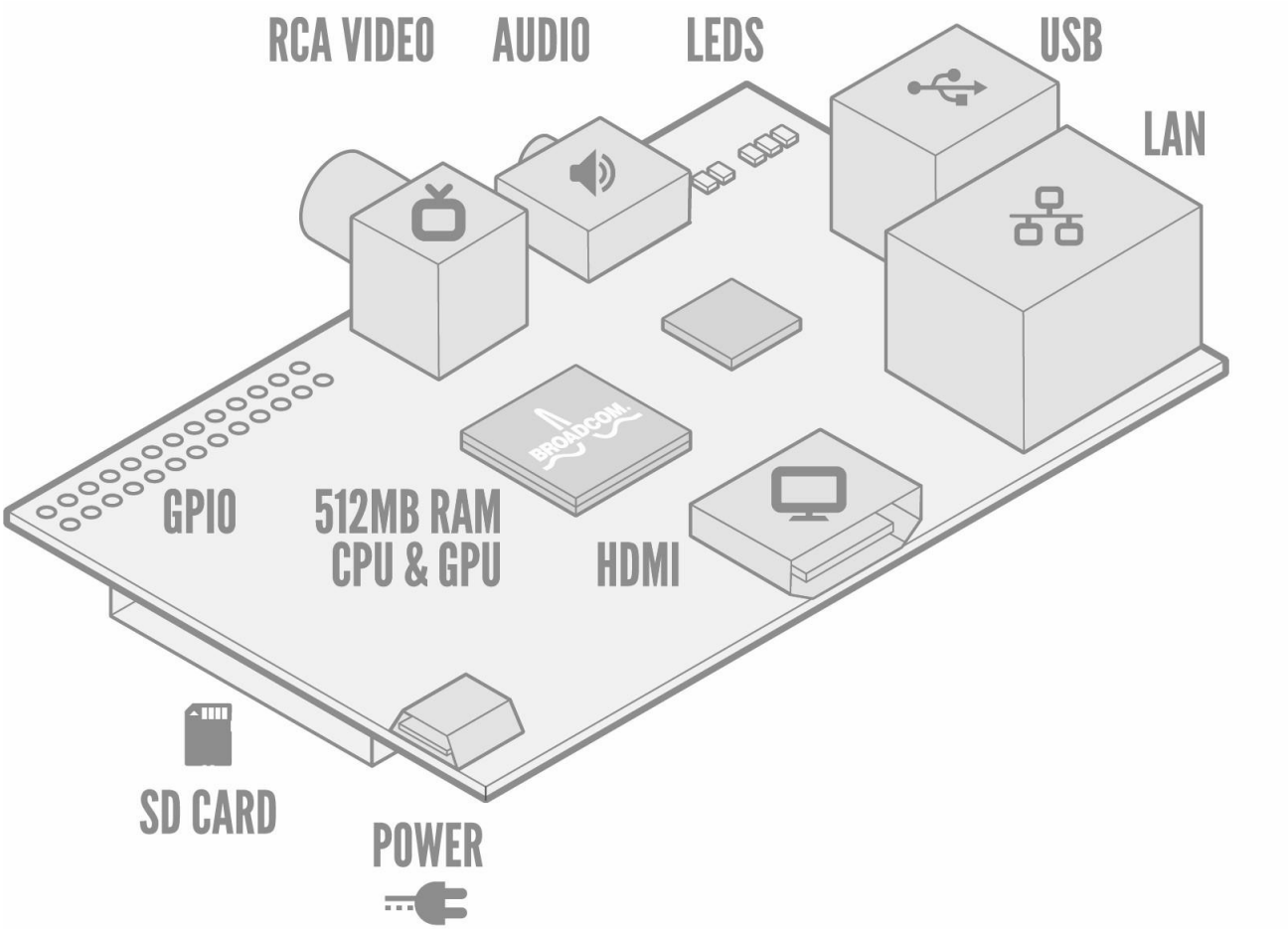


Рис. 2.2. Схема розміщення інтерфейсів на платі

Процесор. У Raspberry Pi B використано 32-бітний ARM Cortex-A7 із тактовою частотою 900 МГц. Об’єм кешу першого рівня становить 16 КБ, другого — 128 КБ; кеш L2 переважно задіює графічне ядро. Мікросхема SoC розміщена під чипом оперативної пам’яті, тому її не видно зовні.

Продуктивність. Працюючи на 700 МГц, плата демонструє близько 0,041 GFLOPS реальної обчислювальної потужності. Рівень CPU приблизно відповідає Pentium II 300 МГц (кінець 1990-х). Графічний процесор здатен обробляти орієнтовно 1 Gpixel/s або 1,5 Gtexel/s, що еквівалентно приблизно 24 GFLOPS у задачах загального призначення.

Програмне забезпечення. Типові ОС для Raspberry Pi — це дистрибутиви на ядрі Linux; можливе також встановлення Windows 10 IoT (існують збірки з ліцензійною версією). ARM11 базується на архітектурі

ARMv6 і традиційно підтримує різновиди Linux. Для інсталяції систем застосовують NOOBS або інші утиліти запису образів на носії, щоб розгорнути сумісні з Raspberry Pi операційні системи на карті пам’яті.

2.2 Установлення та налаштування Raspberry Pi

2.2.1 Підготовка карти пам’яті

Оскільки Raspberry Pi не має вбудованого постійного накопичувача, перед запуском необхідно підготувати карту пам’яті — записати на неї образ обраної операційної системи [12–13]. Потрібно мати:

1. карту SD (MMC/SDIO) на 8–32 ГБ із класом швидкості не нижче 8;
2. кардридер для під’єднання карти до ПК;
3. програму Win32DiskImager;
4. файл образу ОС (у роботі використовується Raspbian).

Далі під’єднайте карту через кардридер до комп’ютера та за допомогою Win32DiskImager запишіть на неї образ операційної системи.

2.2.2 Перше підключення та початкове налаштування

Для старту, окрім самої плати та підготовленої карти, знадобляться USB-клавіатура, дисплей із HDMI-входом або телевізор/тюнер із роз’ємом RCA і відповідним кабелем, а також блок живлення 5 В мінімум на 0,5 А [13]. Після підключення всіх компонентів і подачі живлення система автоматично почне завантаження (рис. 2.3).

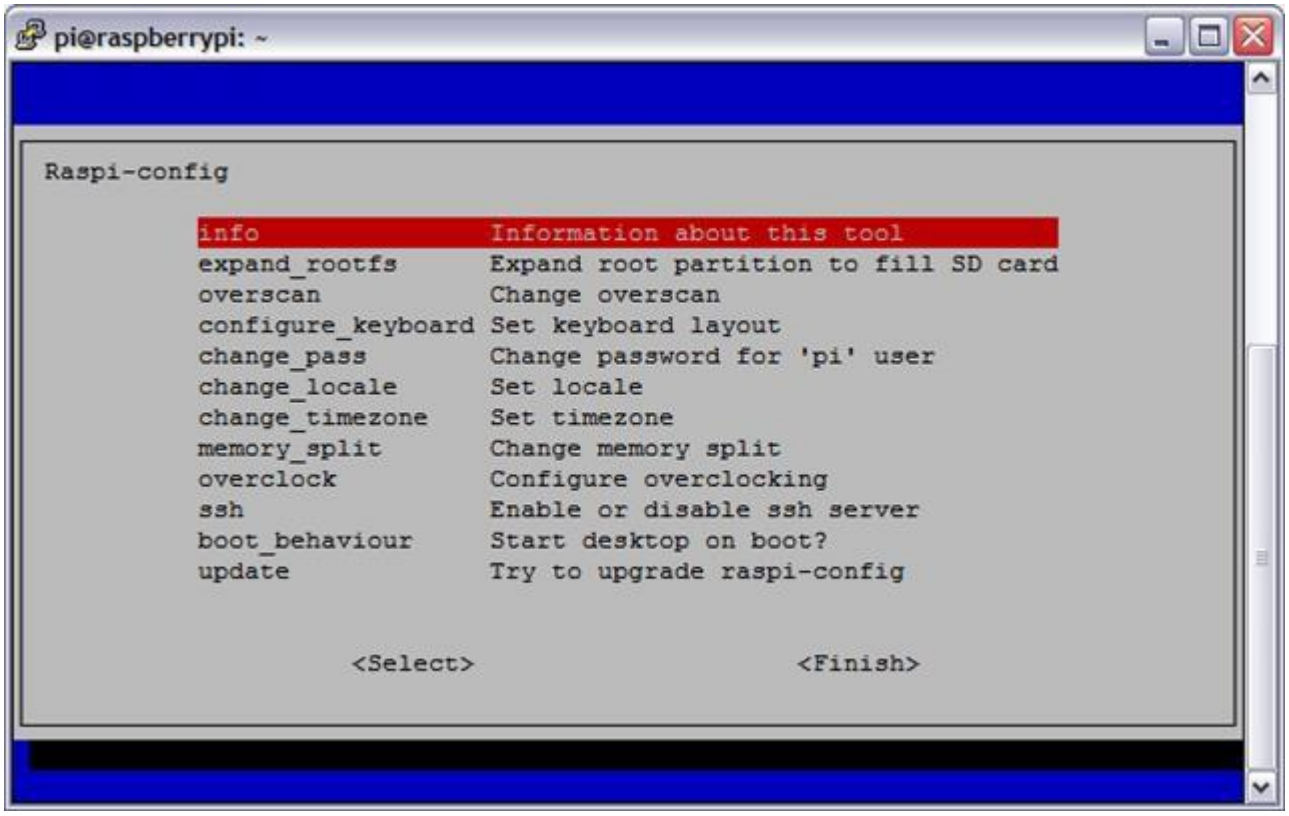


Рис. 2.3 меню первинної конфігурації Raspberry Pi

Нижче подано основні пункти меню, які використовують під час первинного налаштування:

- expand_rootfs — розгортає кореневий розділ на весь доступний обсяг карти пам’яті.
- configure_keyboard — вибір драйвера та параметрів клавіатури, якщо варіант за замовчуванням не підходить.
- change_pass — зміна пароля користувача “pi”; під час введення символи не відображаються, пароль слід увести двічі.
- change_locale — налаштування мовних параметрів системи (локалі).
- change_timezone — встановлення часового поясу; плата не має власної бази поясів і завантажує її з Інтернету.
- memory_split — розподіл оперативної пам’яті між центральним і графічним процесорами.
- overclock — параметри розгону процесора.

- ssh — вмикання/вимикання SSH-сервера; рекомендується активувати для подальшого віддаленого доступу.
- boot_behaviour — вибір режиму старту: одразу запускати графічну оболонку чи ні.

Після завершення налаштувань виконайте перезавантаження: натисніть комбінацію Ctrl+F та підтвердьте дію — плата перезапуститься.

Далі, після завантаження системи, задайте пароль для користувача “root”. У терміналі введіть команду sudo passwd root і двічі наберіть новий пароль. Для перегляду поточних процесів скористайтеся командою top (рис. 2.4).

2.3 Підключення зовнішнього диска та модуля Wi-Fi

Для організації бездротового доступу буде використано Wi-Fi-адаптер Auspi Wireless Adaptor (802.11 b/g/n) для Raspberry Pi. До його переваг належать підтримка стандартів b, g, n, сумісність із USB-портом і відсутність потреби в окремому живленні [13]. Зовнішній накопичувач може бути будь-якого типу; головна вимога — можливість підключення через USB.

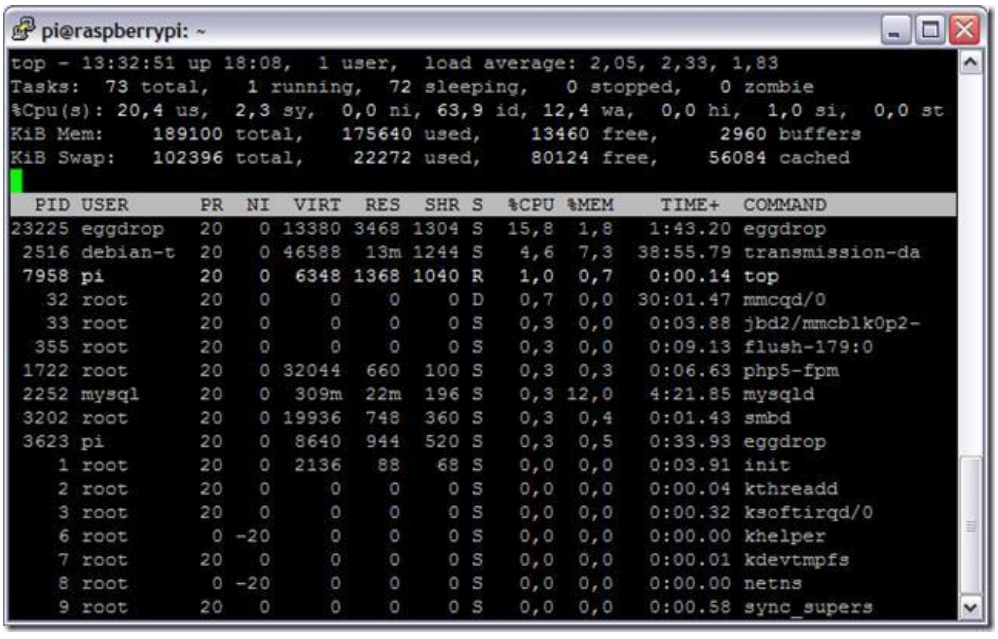


Рис. 2.4 Виконання команди top

Після під’єднання обладнання спершу переконайтеся, що система його розпізнала. Оскільки обидва пристрої працюють через USB, виконайте команду lsusb. Якщо пристрої відображаються у списку, перейдіть до налаштування бездротового інтерфейсу wlan0 у файлі /etc/network/interfaces. Відкрийте його, наприклад, командою sudo nano /etc/network/interfaces, і внесіть потрібні параметри (рис. 2.5).

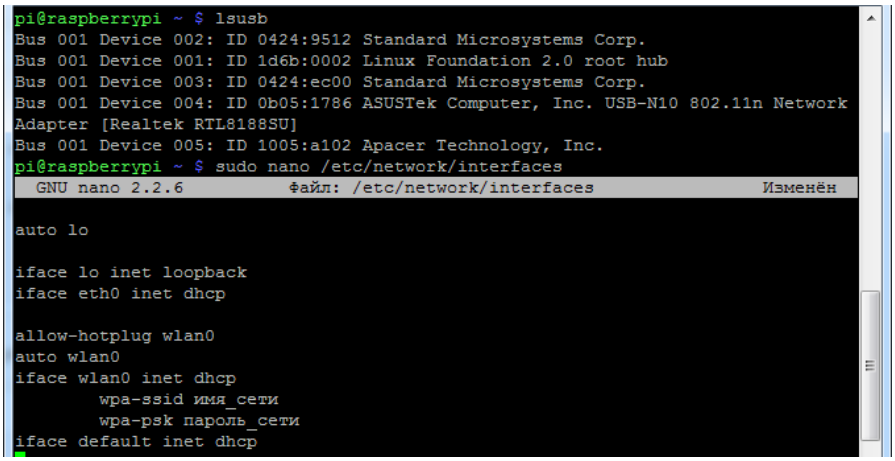


Рис 2.5 виконання команди lsusb та внесення змін у кофігурацію мережі.

Далі перезапустіть мережеві служби командою service networking restart, а потім оновіть систему, послідовно виконавши apt-get update та apt-get upgrade.

Наступний крок — під’єднання зовнішнього жорсткого диска. Спершу у каталозі /media створіть директорію data, яка слугуватиме точкою монтування для файлових розділів.

У файлі /etc/fstab необхідно прописати UUID потрібного розділу та шлях до точки монтування (див. рис. №). Після збереження змін змонтуйте розділ командою sudo mount /media/data. Для кожного додаткового розділу повторіть цю процедуру.

На завершення перезавантажте Raspberry Pi і від’єднайте клавіатуру з монітором, оскільки подальша робота виконуватиметься через SSH.

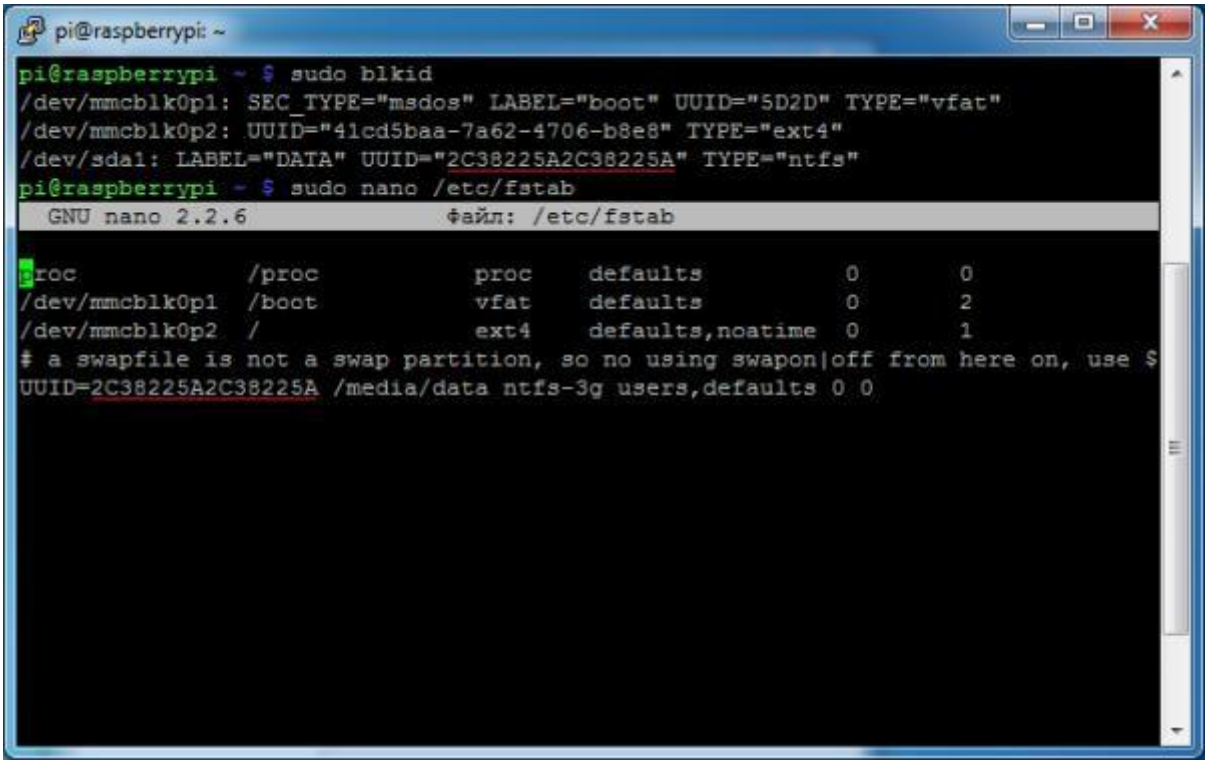


Рис. 2.6 Пошук зовнішнього диску та конфігурування файлу /etc/fstab

Далі в налаштуваннях роутера потрібно присвоїти Raspberry Pi статичну IP-адрессу і прописати параметри провайдера DDNS. На цьому налаштування плати завершене, і її можна використовувати як сервер, та подальші дії проводити з допомогою ssh з'єднання.

2.4 Підключення до Raspberry Pi з допомогою ssh. Налаштування з'єднання та встановлення Domoticz

2.4.1 Підключення до Raspberry з допомогою ssh з'єднання. Для встановлення SSH-з'єднання з ОС Windows зручно використати безплатний пакет PuTTY [14]. Із сайту слід завантажити такі компоненти:

- PuTTY.exe — основний SSH-клієнт;
- PuTTYgen.exe — утиліта для створення пари відкритого й закритого ключів;
- Pageant.exe — агент автентифікації для зберігання ключів;
- PSCP.exe — консольний інструмент захищеного копіювання;

- PSFTP.exe — клієнт для безпечної передачі файлів за протоколом SFTP.

Щоб інструменти коректно запускалися у середовищі Windows, додайте теку з виконуваними файлами до змінної середовища PATH (рис. 2.7):

Variable name: PATH

Variable value: C:\BIN;%PATH%

Це потрібно, аби система могла знаходити всі необхідні утиліти під час роботи.

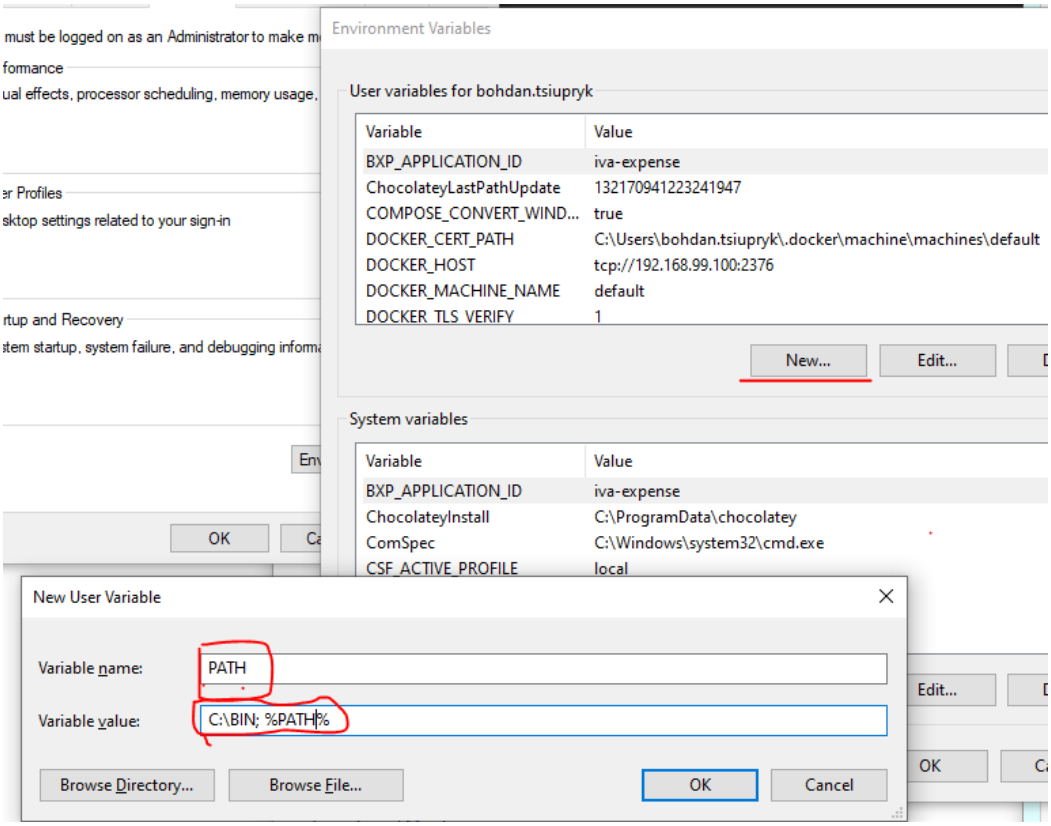


Рис. 2.7 Встановлення змінних середовища, для коректної роботи PuTTY

Далі розглянемо роботу з PuTTY, її базові налаштування та порядок встановлення з’єднання [14]. На рис. 2.8 наведено приклад конфігурації сесії.

Основні параметри:

1. Host Name (or IP address) — вкажіть доменне ім’я або IP того вузла, до якого планується під’єднання.

2. Port — типовий порт для SSH-з’єднань: 22.

3. Connection type — для захищеного каналу оберіть SSH.

4. Saved Sessions — дає можливість зберегти профіль підключення, щоб надалі не вводити параметри вручну; задайте назву профілю й натисніть Save.

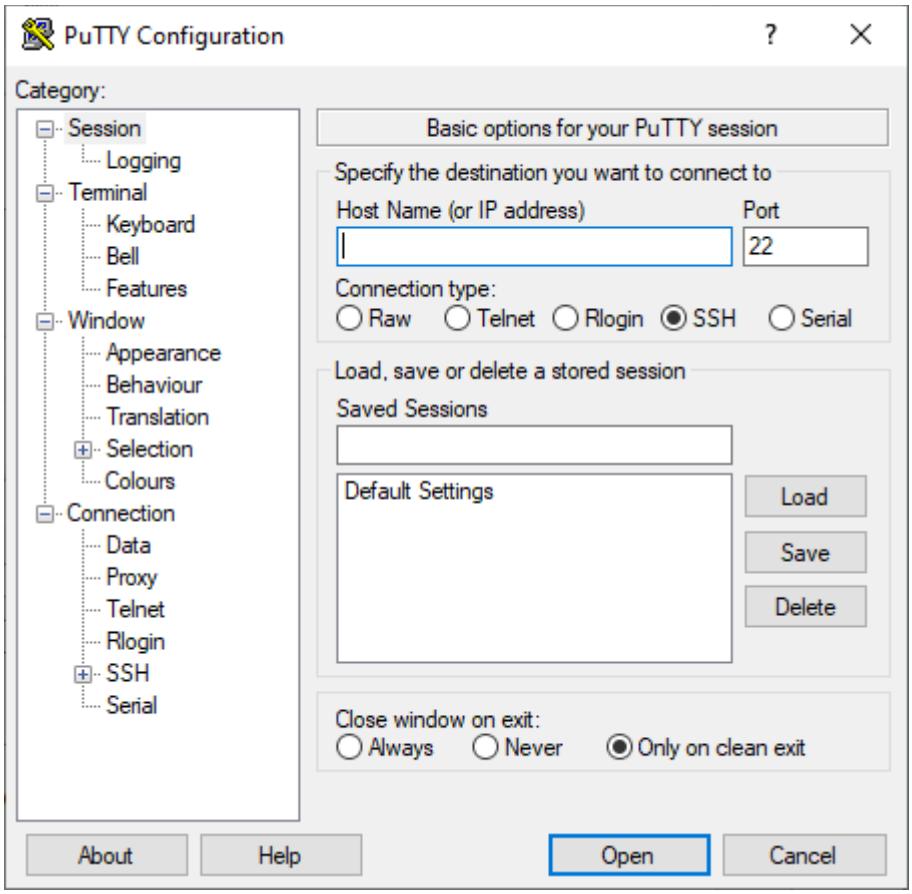


Рис. 2.8 Програма PuTTY. Вкладка session

Після заповнення всіх полів натисніть кнопку Open — з’явиться термінальне вікно. У ньому введіть логін і пароль, задані під час первинного налаштування Raspberry Pi. Після успішної автентифікації буде встановлено з’єднання з платою.

2.4.2 Встановлення та кофігурація Domoticz. Для інсталяції спершу перейдіть у домашній каталог командою cd /home/, після чого завантажте й запустіть інсталятор: curl -L https://install.domoticz.com | bash [15]. Коли встановлення завершиться, відкрийте браузер і перейдіть за адресою IP-адреса-Raspberry-Pi:8080 — з’явиться стартова сторінка Domoticz (рис. 2.9). Якщо пристрої ще не підключені, ця сторінка буде порожньою.

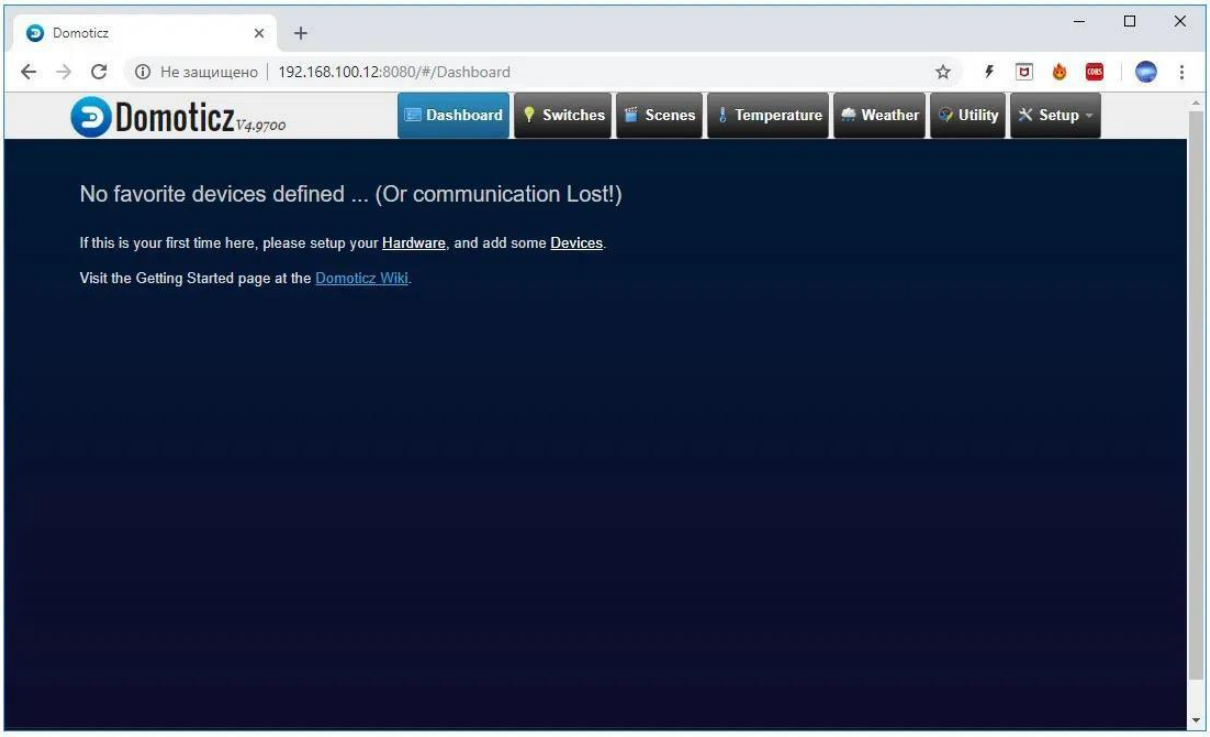


Рис. 2.9 Домашня сторінка програми Domoticz

Для початкового налаштування перейдіть до розділу Setup → Setting (рис. 2.10) й задайте потрібні параметри: мову, тему оформлення, геолокацію, налаштування безпеки сайту тощо. Щоб зберегти зміни, натисніть Apply Setting.

Після цього базове конфігурування завершено, можна додати кілька датчиків для перевірки роботи.

Далі підключимо два термодатчики DS18B20 через інтерфейс 1-Wire, датчик температури та атмосферного тиску BMP180, а також датчик освітленості TSL2561 по I²C. Схему виводів GPIO плати Raspberry Pi наведено в додатку Б.

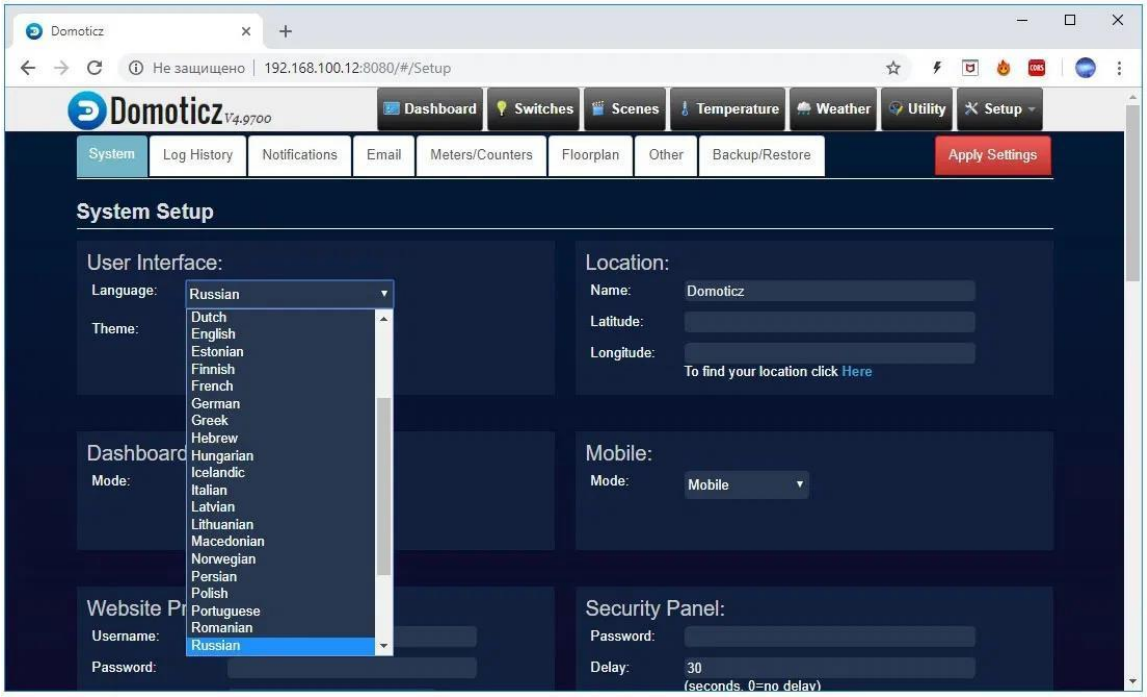


Рис. 2.10. панель налаштувань Domoticz

Додавання датчиків у системі здійснюється через веб-інтерфейс. Перейдіть до вкладки Setup і виберіть Hardware. У відкритому вікні відображається перелік уже підключеного обладнання та форма для додавання нових пристроїв (рис. 2.11). У полі Name вкажіть, наприклад, “DS1820 #1”. У полі Type встановіть “1-Wire (System)”, а поле OWFS Path очистіть. Решту параметрів залиште за замовчуванням і натисніть Add, щоб зберегти. Другий датчик DS18B20 підключається й налаштовується аналогічно [18].

Далі налаштуйте датчик тиску й температури BMP180: у полі Type виберіть I2C sensors, а у SubType — I2C sensor BMP085/180 Temp+Baro (рис. 2.12). За тим самим підходом додається датчик освітленості TSL2561, лише в SubType потрібно обрати I2C sensor TSL2561 Illuminance [19].

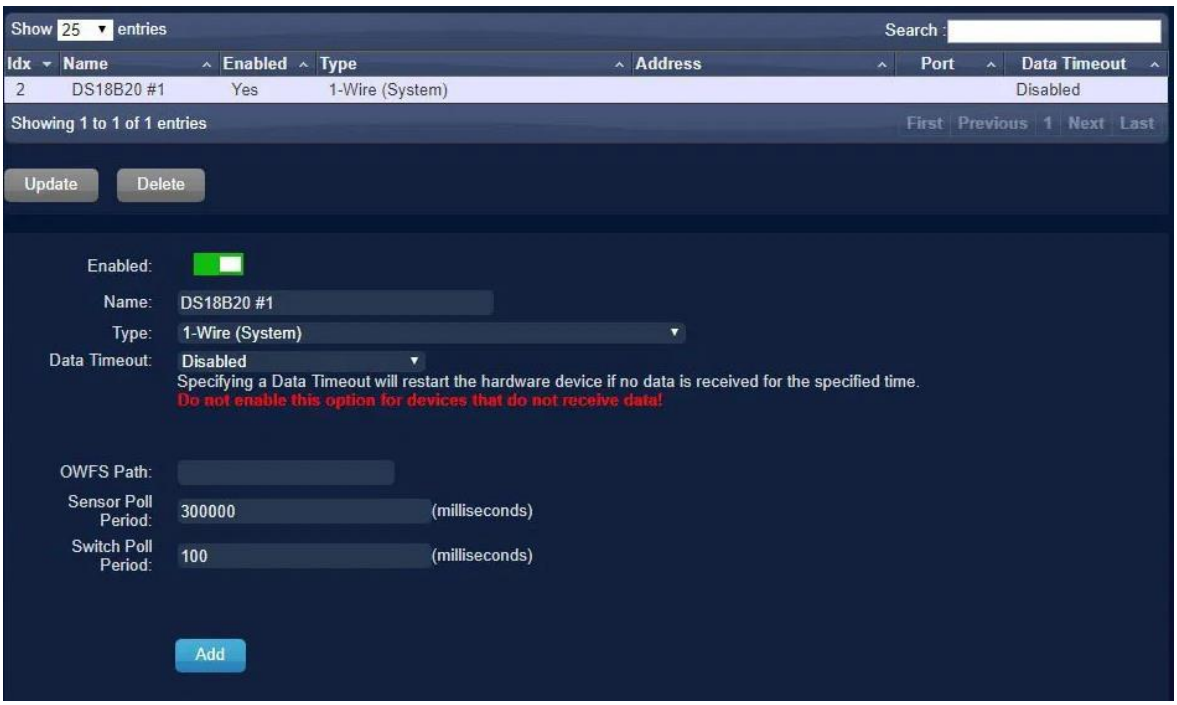


Рис. 2.11. Список підключених пристроїв та вікно додавання нового пристрою

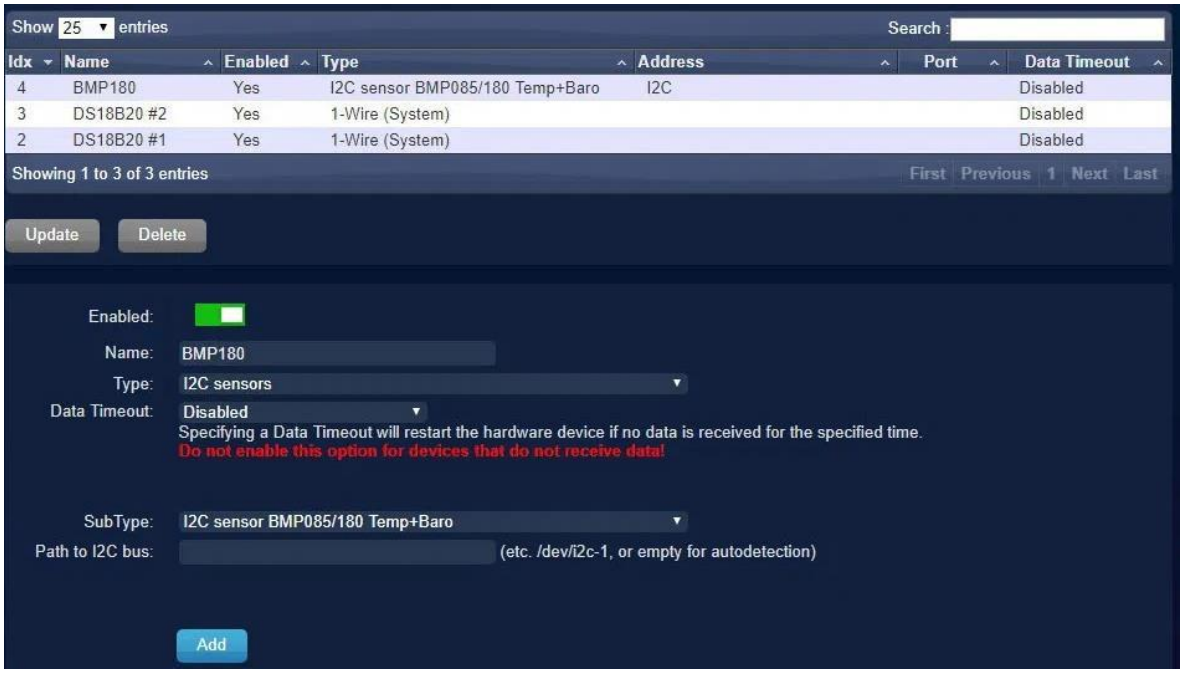


Рис. 2.12. Налаштування датчика атмосферного тиску та температури BMP180

Наразі датчики додані, але залишаються неактивними. Щоб їх увімкнути, відкрийте розділ Setup → Settings і активуйте потрібні елементи:

натисніть зелене кільце навпроти відповідного датчика та задайте йому назву — саме вона відображатиметься в інтерфейсі.

У системі сенсори автоматично групуються за основною функцією. Показники температури від DS18B20 та температурний канал BMP180 з’являться на вкладці Temperature, вимірювання атмосферного тиску BMP180 — у вкладці Weather, а датчик освітленості TSL2561 — у розділі Utility.

Таке групування особливо зручне, коли датчиків багато, але найуживаніші можна винести на головну панель Dashboard (рис. 2.13). Для цього натисніть значок зірки на картці пристрою: піктограма стане жовтою, і відповідний елемент з’явиться на Dashboard.

Рис. 2.13. Панель Dashboard з усіма обраними датчиками

У Domoticz реалізовано журналювання даних. Завдяки цьому можна стежити за змінами показників сенсорів у різні проміжки часу. Щоб відкрити графік конкретного давача, оберіть його у списку і клацніть по його піктограмі. Як приклад, на рис. 2.14 наведено добовий графік температури з датчика DS18B20.

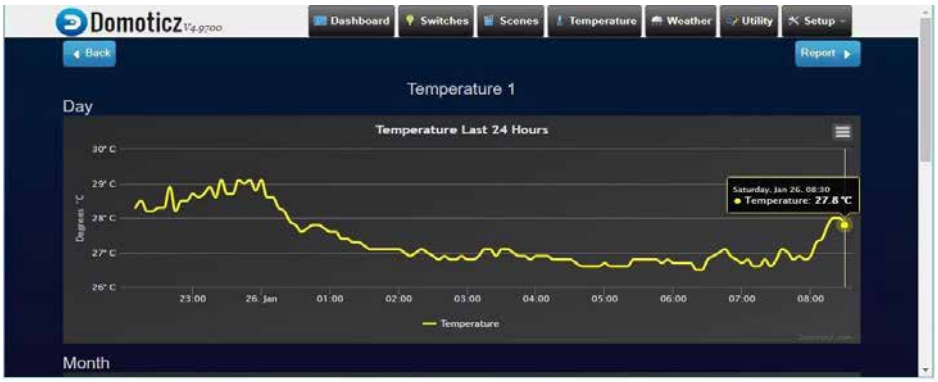


Рис. 2.14. Графік зміни температури у датчика DS18B20

2.5 Аналіз Domoticz API, як способу управління

Domoticz підтримує різні способи взаємодії з усіма під'єднаними перемикачами та сенсорами: через JSON-інтерфейс, безпосередньо у веб-браузері в інтерактивному режимі або програмно за допомогою скриптів [20].

У цьому розділі розглянуто одноплатний комп'ютер Raspberry Pi як платформу, що доцільна для розгортання системи моніторингу й керування IoT-пристроями. Виконано початкове налаштування плати, інсталяцію Domoticz та базову конфігурацію. Окрему увагу приділено процесу налаштування і можливостям користувацького програмованого інтерфейсу (API), які були проаналізовані під час роботи системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАСОБІВ МОНІТОРИНГУ ТА УПРАВЛІННЯ ЕЛЕМЕНТАМИ ГРАНИЧНИХ ПРИСТРОЇВ ІОТ

3.1 Опис архітектури та інструментів розробки

Обрано клієнт–серверну модель. Клієнтом виступає застосунок, що працює на тій самій машині або в локальній мережі разом із Domoticz; сервер, своєю чергою, може бути розгорнутий на віддаленому вузлі чи хостингу. Такий підхід забезпечує автономність сервера від клієнтів, а клієнтам — простий механізм підключення та відсутність взаємозалежностей між собою.

Застосунок реалізовано мовою Java 8 із використанням фреймворку Spring Boot. Зберіганням даних опікується СКБД PostgreSQL; доступ до БД організовано через ORM Hibernate. Фреймворк має власну об’єктно-орієнтовану мову запитів HQL і дає змогу уникнути написання більшості «ручних» SQL-операторів, оскільки стандартні дії вже інкапсульовані у методах бібліотеки. Веб-інтерфейс створено за допомогою шаблонізатора FreeMarker.

Середовищем розробки було обрано Intellij IDEA від компанії JetBrains. У нього є безплатна версія з потужними плагінами та інструментами розробки, що полегшує розробку на мові Java.

3.1.1 Архітектура серверного додатку.

Застосунок реалізовано за шаблоном MVC (model–view–controller). Такий підхід розв’язує модулі між собою, спрощуючи підтримку й заміну окремих частин [23–25].

Коротко про складові:

- Model — робота з базою даних, валідація та опрацювання інформації.
- View — шар подання, що приймає вхідні дані користувача та відображає результати обробки.

- Controller — бізнес-логіка: координує операції з даними та викликає потрібні дії.

Model інкапсулює структури даних і алгоритми їх опрацювання та не залежить від каналів введення/виведення. View може складатися з кількох взаємопов'язаних областей (наприклад, таблиці та форми), у яких показуються актуальні значення.

Завдання Controller — реагувати на події, що виникають унаслідок дій користувача, і впорядковувати код через групування споріднених операцій в окремі класи. Типовий приклад — контролер користувача з методами реєстрації, автентифікації, редагування профілю та зміни пароля.

Зареєстровані події перетворюються на запити до компонентів моделі або елементів подання. Розділення моделі й уявлення дає можливість незалежно змінювати засоби відображення. Тому, якщо через контролер відбувається зміна стану моделі, візуальні компоненти автоматично оновлюють показані дані.

3.1.2 Опис архітектури клієнтського додатку. Клієнтський застосунок реалізовано як моноліт. Такий вибір зумовлений тим, що програмі не потрібні ані підключення до СКБД, ані веб-інтерфейс [25]. Головна функція — підтримувати з'єднання з віддаленим сервером і здійснювати передавання повідомлень, тож перехід на мікросервіси чи використання шаблону MVC у цьому випадку був би зайвим ускладненням.

3.1.3 Опис мови програмування Java та фреймворку Spring Boot. Java — об'єктноорієнтована мова програмування, представлена у 1995 році. Синтаксис наближений до сімейства C. Її ключова особливість — Java Virtual Machine (JVM), що виконує байткод і забезпечує переносимість: один і той самий код працює на різних платформах, що прискорює розробку та зменшує її вартість. Байт-код зазвичай генерується з вихідних файлів Java, хоча це не є обов'язковим.

Підстави обрати Java для цього проєкту:

- простий, об'єктноорієнтований синтаксис;

- безпечна робота з пам’яттю та відмовостійкість;
- платформна й архітектурна незалежність;
- добра продуктивність.

Spring Boot — фреймворк, що радикально спрощує розробку Java-застосунків: він надає готові реалізації типових функцій, які в «чистій» Java довелося б писати вручну. Дозволяє швидко підняти готові до продакшену Spring-додатки, фактично «просто запустивши» їх [22]. Платформа містить інтеграції зі сторонніми бібліотеками, тож старт потребує мінімальних налаштувань у більшості випадків.

3.1.4 Можливості Spring Boot:

1 Створення самодостатніх застосунків, які не потребують зовнішніх допоміжних програм.

2 Наявність вбудованих контейнерів сервлетів (наприклад, Tomcat або Jetty).

3 Автоконфігурування компонентів там, де це можливо.

4 Підтримка моніторингу стану, розширених параметрів налаштування та метрик.

5 Конфігурація без генерування коду і без XML-файлів (на відміну від класичного Spring).

Для створення вебзастосунку у Spring Boot слід додати залежність для роботи з веб-інтерфейсами. Оскільки фреймворк використовує систему збирання Maven, підключення потрібної бібліотеки зводиться до кількох рядків у файлі pom.xml (рис. 3.1).

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Рис. 3.1 Підключення залежностей для web розробки для Spring Boot

PostgreSQL та Hibernate. PostgreSQL — реляційна об'єктно-орієнтована СКБД з відкритим кодом, що розвивається спільнотою й не належить жодній окремій компанії. Вона підтримує широкий спектр вбудованих типів даних і дає змогу визначати власні користувацькі типи. Ключові можливості PostgreSQL:

1. дотримання властивостей ACID;
2. підтримка складних запитів і підзапитів (JOIN, UNION тощо);
3. механізм послідовностей;
4. реплікація;
5. засоби контролю цілісності;
6. рекурсивні запити та загальні табличні вирази (CTE).

Hibernate — ORM-фреймворк, що відображає класи Java на таблиці бази даних і надає зручні інтерфейси для роботи з даними. Його завдання — зменшити обсяг ручного коду за рахунок готових операцій доступу, автоматично підтримувати зв'язки між сутностями й відповідними таблицями та спрощувати їх поєднання. Фреймворк має власну мову запитів HQL — спрощений, об'єктно-орієнтований аналог SQL, зручний для роботи зі сутностями, а не з таблицями.

3.1.5 Шаблонізатор FreeMarker. FreeMarker — легкий засіб

шаблонізації, що дає можливість створювати гнучкий інтерфейс користувача без глибокого занурення в JavaScript чи TypeScript. Важлива перевага — підтримка макросів, завдяки яким можна перевикористовувати повторювані фрагменти і суттєво скорочувати обсяг шаблонного коду.

Реалізація системи автоматизації збору даних

Реалізація моделей і бази даних. У проєкті визначено такі сутності:

- Device — пристрій, під’єднаний до системи.
- DeviceDTO — транспортне подання пристрою для серіалізації та передавання без зайвих полів.
- DeviceState — стан/телеметрія пристрою.
- User — обліковий запис користувача.
- UserRole — роль, що визначає права користувача.
- Diagram — модель для відображення графіків/серій даних.
- Client — клієнтський вузол системи.
- Command — команда, яку можна надсилати пристроям і зберігати в історії.

Роботу з цими сутностями організовано через репозиторії (Repository) — інтерфейси фреймворку Hibernate; приклад їхньої реалізації наведено на рис. 3.2. Такий підхід мінімізує дублювання: не потрібно для кожної моделі окремо писати стандартні операції доступу до БД, адже вони надаються інфраструктурою репозиторіїв [24].

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface DeviceRepo extends JpaRepository<Device, Long> {
}
```

Рис. 3.2. Приклад наслідування інтерфейсів, для уникнення великої кількості коду при доступі до БД

Інтерфейси JpaRepository та CrudRepository надають лише базові CRUD-операції: отримати всі записи, знайти за ідентифікатором, видалити за ідентифікатором, оновити дані конкретного об’єкта. Коли потрібна розширена логіка доступу, в JPA передбачено механізм «похідних запитів за назвою методу»: необхідну вибірку можна описати, додавши метод із правильно сформованою назвою.

Наприклад, потрібно отримати всі пристрої зі станом OFF. Базові інтерфейси дозволяють шукати лише за id, тому оголошуємо власний метод. Оскільки результатів може бути кілька, очікуємо колекцію: List<Device>. Далі використовуємо префікс пошуку find, вказуємо, що потрібні всі записи (findAll), і додаємо критерій за полем стану — ByStatus. У параметрах методу передаємо значення статусу (див. рис. 3.3).

За таким самим принципом визначаються репозиторії й методи доступу для інших доменних сутностей.

```
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface DeviceRepo extends JpaRepository<Device, Long> {
    List<Device> findAllByStatus(DeviceStatus status);
}
```

Рис. 3.3. Приклад сформованого запиту на пошук пристрою по статусу

Реалізація бізнес-логіки додатку. Бізнес-логіку зосереджено в пакеті service. Призначення застосунку полягає в агрегуванні даних із кількох серверів із установленим Domoticz, їхньому впорядкуванні в єдиному місці, формуванні статистики та, за потреби, здійсненні керування.

У Domoticz доступний API, що приймає запити у форматі JSON: через нього можна отримувати потрібні дані або надсилати керувальні команди. Для виконання HTTP-запитів використовується стандартний інструмент

Spring Boot — RestTemplate, який дає змогу передавати й отримувати дані без додаткових перетворень. Обмін інкапсульовано в сервісному класі HttpUtil; він успадкований від абстрактного класу ClosableHttpUtil (рис. 3.4).

```
public static HttpClientResponse executeRequest(HttpUriRequest request, CloseableHttpClient closeableHttpClient) {
    String jsonText = null;
    int statusCode;

    try (CloseableHttpResponse response = closeableHttpClient.execute(request)) {
        statusCode = response.getStatusLine().getStatusCode();
        log.debug("Status code: {}", statusCode);
        HttpEntity entity = response.getEntity();
        if (entity != null) {
            jsonText = extractJsonStringFromHttpResponse(entity);
            EntityUtils.consume(entity);
        }
```

Рис. 3.4. Приклад методу для надсилання запитів

Основні сервіси системи:

- HttpUtil — утилітний сервіс для формування та надсилання HTTP-запитів.
- DeviceService — відповідає за операції з пристроями (облік, отримання станів, керування).
- CommandService — створює та обробляє команди, забезпечує їх доставку й фіксацію.
- UserService — управління користувачами: реєстрація, автентифікація, зміна параметрів облікового запису.
- DiagramService — обчислює та трансформує набори даних у структури, придатні для відображення графіків.
- MailService — надсилання електронних листів зі статистикою та іншими службовими повідомленнями.
- StatisticService — збирає метрики, агрегує та аналізує статистику щодо пристроїв і роботи застосунку.
- InitializationService — встановлює з’єднання з серверами та виконує початкове конфігурування системи.

Реалізація інтерфейсу користувача та обробки інформації, що приходить від користувачів. Користувацький інтерфейс реалізовано з використанням шаблонізатора FreeMarker. Його суттєва перевага — макроси, що дають змогу багаторазово застосовувати один і той самий фрагмент розмітки, змінюючи лише окремі параметри. Наприклад, для побудови діаграм достатньо один раз описати шаблон відображення, а самі дані підставляти динамічно.

Для візуалізації графіків обрано Google Charts — легку у використанні бібліотеку: потрібно підключити скрипт, визначити тип діаграми та передати набір даних (рис. 3.5). Оформлення інтерфейсу виконано за допомогою Bootstrap 4, який вирізняється простотою опанування та невеликою «вагою».

Оброблення інформації, що надходить із інтерфейсу, здійснюють контролери: для них задаються маршрути отримання запитів; далі вхідні дані опрацьовуються й передаються до сервісного шару для подальших дій.

```
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
  google.charts.load('current', {'packages':['corechart']});
  google.charts.setOnLoadCallback(drawChart);

  function drawChart() {
    var data = google.visualization.arrayToDataTable([
      ['Year', 'Sales', 'Expenses'],
      ['2004',  1000,      400],
      ['2005',  1170,      460],
      ['2006',  660,       1120],
      ['2007',  1030,      540]
    ]);
```

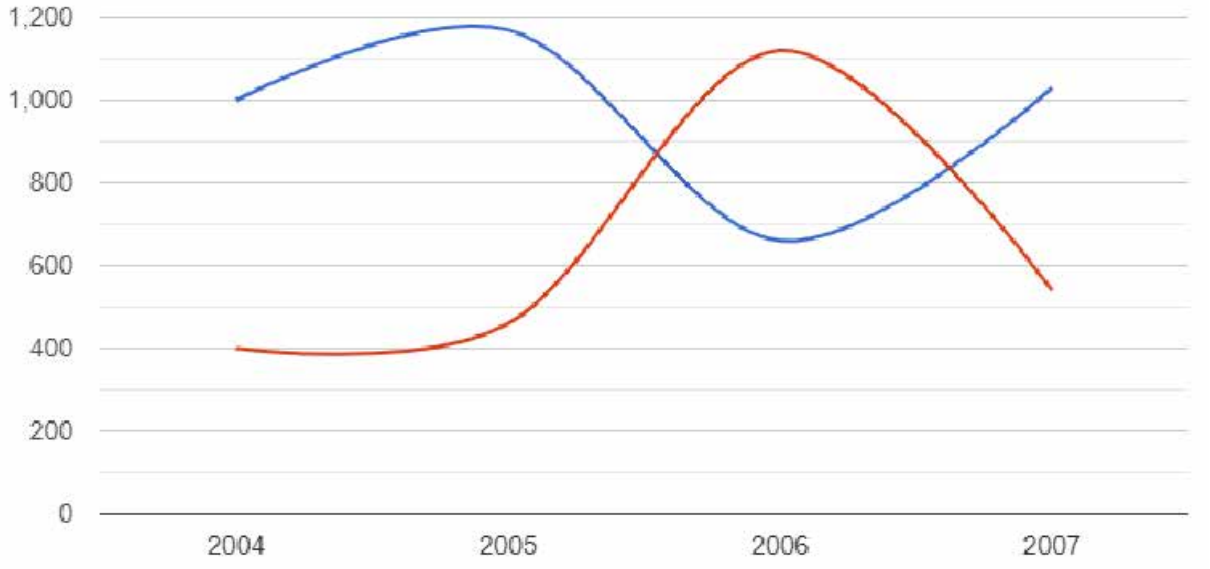


Рис. 3.5. Приклад коду діаграм, та їх вигляд після відображення

Оголошення контроллера відбувається за допомогою анотації @Controller або @RestController (рис 3.6). А шлях який буде обробляти контроллер задається з допомогою анотації @RequestMapping(path). Причому, методи контроллера, можуть мати свої шляхи, які будуть з'єднуватись з шляхом контроллера. Наприклад, якщо у контроллера вказаний шлях "/device", а в метода додана анотація @GetMapping("all"), то в результаті такий метод буде обробляти шлях "/device/all" з HTTP методом GET.

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/device")
public class DeviceController {
    private final DeviceService deviceService;

    @GetMapping("all")
    public ResponseEntity<List<Device>> getAll() {
        List<Device> all = deviceService.getAll();
        return ResponseEntity.ok(all);
    }
```

Рис. 3.6. Приклад оголошення шляху, який обробляє HTTP метод GET

3.2 Налаштування системи

Конфігурація серверної частини системи. Налаштування доступу до бази даних здійснюється у конфігураційному файлі db.properties (рис. 3.7). Після будь-яких змін необхідно перезапустити застосунок, оскільки параметри підтягуються під час старту.

Основні параметри:

- datasource.url — адреса (шлях) до БД.
- datasource.username — облікове ім'я користувача БД.
- datasource.password — пароль для підключення до БД.

```
datasource.url=jdbc:postgresql://localhost/iot-control-system
datasource.username=postgres
datasource.password=123
```

Рис. 3.7. Приклад конфігурації доступу до бази даних

Налаштування клієнтського модуля. Відкрийте файл application.properties і вкажіть такі параметри:

- remote-server.ip — IP-адреса віддаленого сервера, куди передаватимуться дані.
- remote-server.port — необов'язковий параметр; заповнюйте лише якщо сервер працює не на порту 8080.
- remote-server.key — секретний ключ доступу; без нього запити будуть відхилені.
- remote-server.local-id — унікальний ідентифікатор клієнта.
- domoticz.login — облікове ім'я для під'єднання до Domoticz.
- domoticz.password — пароль до Domoticz.
- domoticz.port — необов'язковий; зазначте, якщо Domoticz використовує порт, відмінний від 8080.

Після збереження параметрів запустіть клієнтську частину. Вона автоматично встановить з'єднання з Domoticz, збере перелік під'єднаних пристроїв і, за умови успішних попередніх кроків, під'єднається до віддаленого сервера. Якщо сервер тимчасово недоступний, застосунок перейде в режим очікування та періодично намагатиметься відновити підключення.

3.3 Демонстрація роботи

3.3.1 Інсталяція та запуск. Для розгортання системи на цільовій машині має бути встановлена JVM версії не нижче 8.

3.3.1.1 Клієнтський застосунок. Скопіюйте файл IoT-control-system-client.jar на вузол, де працює Domoticz, і запустіть його командою:

java -jar IoT-control-system-client.jar

3.3.1.2 Серверний застосунок. Попередньо необхідно мати розгорнуту базу даних і створеного користувача. Далі перенесіть файл IoT-control-system-server.jar на сервер і виконайте запуск:

java -jar IoT-control-system-server.jar

3.3.2 Демонстрація роботи. Користуватися системою можна з ПК, ноутбуків і мобільних пристроїв; головна вимога — наявність сучасного веббраузера. Для входу перейдіть за адресою:

https://<IP-адреса-сервера>:8080

Після цього відкриється сторінка авторизації (рис. 3.8).

Рис. 3.8. Форма входу в систему

Якщо це перший запуск, потрібно увійти під стандартними креденціалами “admin:admin”. Після чого змінити пароль, та додати інших користувачів за необхідності. Для цього в системі додана панель керування

користувачами (рис 3.9). На панелі можна редагувати інформацію та права користувачів, а також додавати нових.

Рис. 3.9. Панель адміністратора

Щоб під’єднати клієнтські застосунки до системи, їх спершу реєструють у розділі створення клієнтів (рис. 3.10). У формі потрібно вказати назву клієнта, яка слугуватиме системним ідентифікатором, і згенерувати секретний ключ безпеки. Після збереження запису клієнт може автентифікуватися та встановити з’єднання.

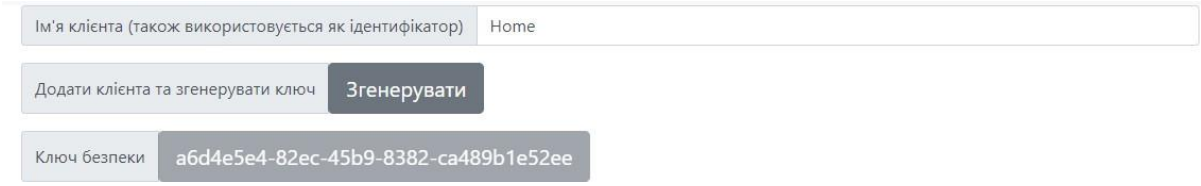


Рис 3.10. Приклад додавання нового клієнта

Після створення запису клієнта його слід налаштувати й запустити. У конфігурації потрібно зазначити ідентифікатор та секретний ключ, згенеровані під час реєстрації. Після успішного встановлення з’єднання клієнт відобразиться в переліку зі статусом «активний» (рис. 3.11). Якщо з’єднання ще не створено — статус буде «новий». Коли сеанс уже існував, але перервався або пристрій вимкнено, відображатиметься «неактивний».

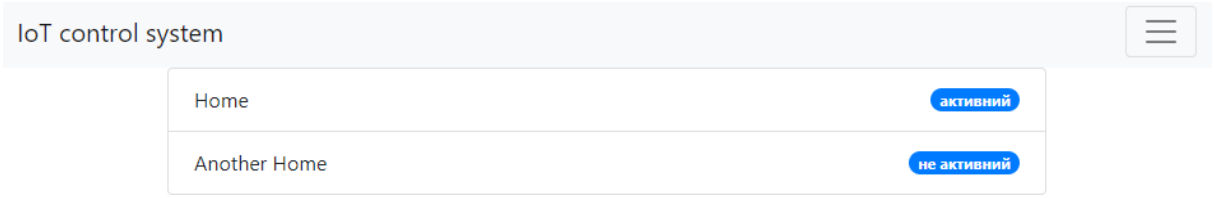


Рис. 3.11. Список клієнтів та їх статус

Щоб переглянути відомості про клієнта, оберіть його у списку. Відкриється сторінка з детальною інформацією: назва пристрою, його IP-адреса, час безперервної роботи (uptime), кількість під'єднаних елементів із можливістю відкрити дані кожного окремо, а також інші корисні параметри.

Список обладнання доступний на вкладці «Домашня», де відображаються всі пристрої, під'єднані на поточний момент (рис. 3.12). Перегляд можна впорядкувати за клієнтом або за типом пристрою (наприклад, датчик температури, лампа тощо).

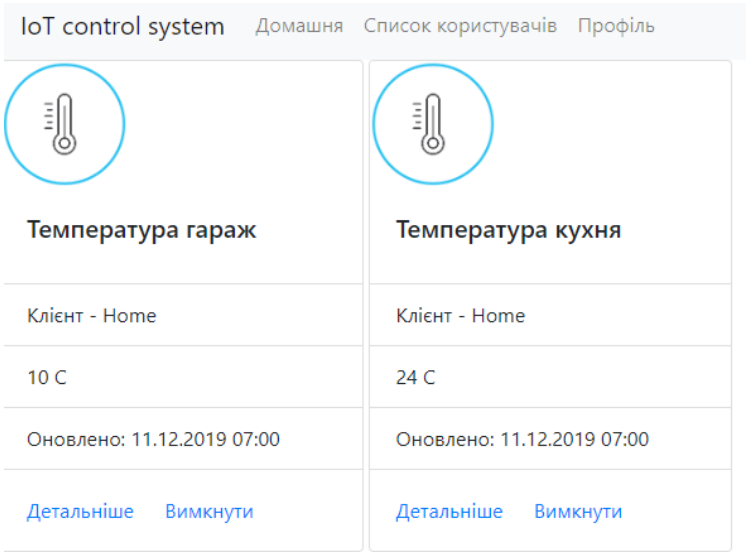


Рис. 3.12. Відображення на домашній сторінці датчиків температури

З головної сторінки можна перейти до розширеного перегляду пристроїв, а також вимкнути окремий елемент. Вимкнені пристрої на домашній панелі не показуються; щоб знову їх увімкнути, слід зайти в налаштування відповідного клієнта.

Натиснувши «Детальніше» на картці пристрою, відкриєте сторінку з повним описом: назва, поточні показники, перелік доступних команд для надсилання. Якщо це, наприклад, температурний датчик, додатково відображається графік змін температури за останні 24 години (рис. 3.12).

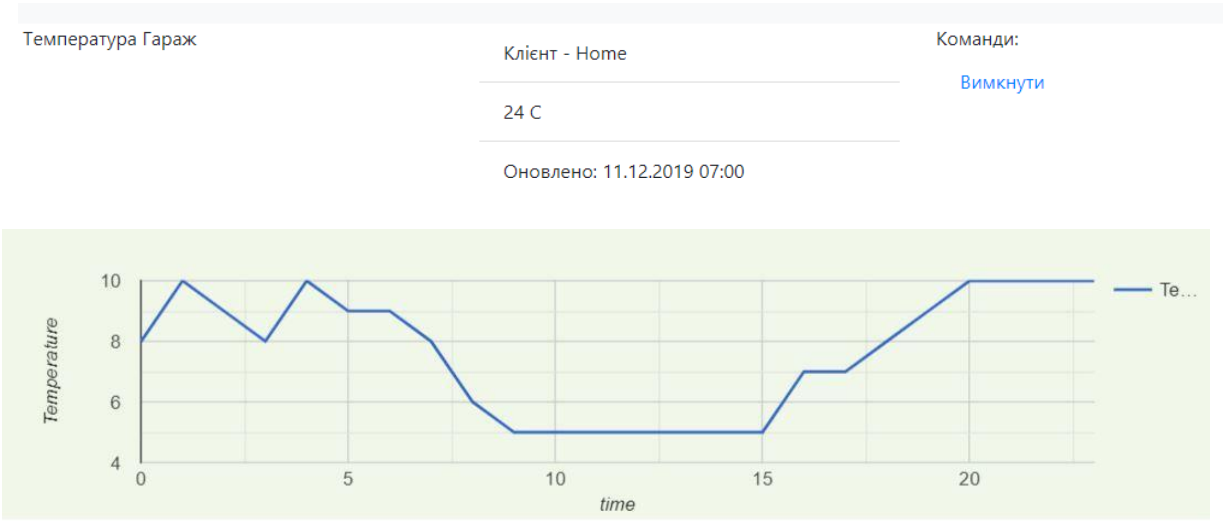


Рис. 3.12. Детальна інформація про пристрій та графік зміни температури

3.4 Висновки до розділу

У цьому розділі подано опис інструментарію, зокрема мови Java та фреймворку Spring Boot, за допомогою яких реалізовано систему, що працює поверх Domoticz і розширює його функціональність. Застосовано незалежну архітектуру, яка дає змогу без змін у базовому коді оперативно підключати додаткові сервери-клієнти. Також продемонстровано підхід до роботи з клієнтами: процедуру їх додавання, конфігурацію та подальше налаштування.

ВИСНОВКИ

У межах цієї магістерської роботи виконано цілісне дослідження моніторингу та керування периферійними компонентами IoT з акцентом на спрощення інтеграції різнорідних пристроїв і підвищення керованості розподілених мереж. Запропоновано архітектуру, що агрегує телеметрію з кількох географічно рознесених вузлів Domoticz, нормалізує дані, забезпечує віддалений контроль і масштабується без змін базового коду.

Ключові результати:

- проведено порівняльний аналіз транспортів і протоколів (BLE, NFC, Wi-Fi HaLow; MQTT, DDS, STOMP, CoAP, XMPP, SOAP) за енергоефективністю, QoS та затримками, обґрунтовано вибір MQTT для обмежених каналів і DDS для сценаріїв реального часу;
- спроєктовано й реалізовано надбудову на Java 8/Spring Boot з використанням PostgreSQL/Hibernate, REST-інтерфейсів і механізмів ключової автентифікації; веб-подання створено на FreeMarker/Bootstrap;
- розгорнуто експериментальний стенд (DS18B20, BMP180, TSL2561), виконано конфігурацію, журналювання та візуалізацію, оцінено стабільність, затримки й ресурсні витрати;
- продемонстровано централізоване керування з кількох майданчиків і швидке підключення нових клієнтів без переписування системного коду.

Практична цінність полягає у зменшенні інтеграційних витрат, підвищенні прозорості експлуатації та придатності рішення для побутових і малих промислових сценаріїв. Перспективи подальших робіт: наскрізне шифрування й взаємна автентифікація (TLS/mTLS), MQTT over WebSocket, потокова аналітика (CEP) та ML на периферії, OTA-оновлення, контейнеризація й розширення підтримки драйверів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Интернет вещей в научных исследованиях. 2017. URL: https://cyberleninka.ru/article/v/internet-veschey-v-nauchnyh-issledovaniyah (дата звернення: 01.09.2019)

2. Обзор и сравнительний анализ технологий LPWAN сетей. 2016. URL: www.sut.ru/doci/nauka/review/20164/33-48.pdf (дата звернення: 01.10.2019)

3. Z-Wave Technical Basics 2017 URL: https://www.domotiga.nl/attachments/download/1075/Z-Wave%20Technical%20Basics-small.pdf (дата звернення: 01.10.2019)

4. Технология NFC принципы работы и преимущества. 2016. URL: http://www.fotokomok.ru/texnologiya-nfc-principy-raboty-i-preimushhestva/ (дата звернення: 01.10.2019)

5. RFID-технології та магнітні мітки. 2015. URL: http://allta.com.ua/what-is-rfid (дата звернення: 01.10.2019)

6. The Basics of Bluetooth Low Energy (BLE). 2010. URL: https://www.novelbits.io/basics-bluetooth-low-energy/ (дата звернення: 01.10.2019)

7. Wi-Fi HaLow (IEEE 802.11ah) — дальнобойное беспроводное подключение с низким энергопотреблением для интернета вещей. 2016. URL: https://www.ixbt.com/news/2016/01/05/wi-fi-halow-ieee-802-11ah.html (дата звернення: 01.10.2019)

8. LoRa Mesh Networking with Simple Arduino-Based Modules. 2018. URL: https://github.com/nootropicdesign/lora-mesh (дата звернення: 15.10.2019).

9. Bryan Boyd et al. Building Real-time Mobile Solutions with MQTT and IBM MessageSight. IBM Redbooks, 2014

10. IBM Podcast: Piper, Diaz, Nipper – MQTT. 2011. URL: http://www.ibm.com/podcasts/software/websphere/connectivity/piper_diaz_nipper_mq_tt_1118201 1.pdf.

11. Raspberry Pi Documentation. Official resource. URL: https://www.raspberrypi.org/documentation/(Last accessed: 14.10.2019).

12. THE OFFICIAL Raspberry Pi Beginner's Guide. URL: https://www.raspberrypi.org/magpi-issues/Beginners_Guide_v1.pdf (Last accessed: 14.10.2019).

13. Questions regarding armhf port for Raspberry Pi. URL: https://lists.debian.org/debian-arm/2012/03/msg00002.html (Last accessed: 14.10.2019).

14. PuTTY Documentation Page. URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/docs.html (Last accessed: 14.10.2019).

15. Domoticz. Частина Друга. webhomepi. URL: https://whp.home.blog/2019/02/02/domoticz-%d1%87%d0%b0%d1%81%d1%82%d1%8c-%d0%b2%d1%82%d0%be%d1%80%d0%b0%d1%8f/

16. Domoticz. Open Source Home Automation System. URL: https://www.domoticz.com/DomoticzManual.pdf

17. Domoticz. Official GitHub Page. URL: https://github.com/domoticz/domoticz

18. DS18B20. Programmable Resolution 1-Wire Digital Thermometer. URL: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

19. BMP180 Datasheet (HTML) - Texas Instruments. URL: https://www.alldatasheet.com/datasheet-pdf/pdf/514264/TI1/BMP180.html

20. Domoticz API/JSON URL's. Wiki page. URL: https://www.domoticz.com/wiki/Domoticz_API/JSON_URL%27s

21. The Java Language Environment. 1997 Sun Microsystems, Inc. URL: https://www.oracle.com/technetwork/java/intro-141325.html

22. Rob Harrop, Clarence Ho. Pro Spring 3. United Kingdom. 2012 944p

23. Joshua Bloch. Effective Java. Addison-Wesley Professional. 2018 412p

24. Андрій Будай. Дизайн-патерни — просто, як двері. 2012 90p

25. Alex Berson. Client/Server Architecture (McGraw-Hill Computer Communications Series). 1996 569p

26. Christian Bauer, Gary Gregory, Gavin King. Java Persistence with Hibernate Second Edition. 2015 608p

27. MQTT Essentials. URL: https://www.hivemq.com/mqtt-essentials/

28. Robert C. Martin. Clean Code. 2008 464p

29. Brian Goetz, Tim Peierls, Joshua Bloch. Java Concurrency in Practice. 2006 432p

30. Eric Freeman, Elizabeth Robson. Head First Design Patterns. 2004 694 p.