Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
Система віддаленого контролю та управління IoT-пристроями
08-54.БКР.009.00.000 ПЗ

Виконав:студент4курсу, групи 1КІ-23мс
спеціальності 123 —«Комп'ютерна інженерія»
освітня програма —«Комп'ютерна інженерія»
Зять М. В.
Керівник к.т.н., доц. каф. ОТ
Колесник І. С.
Рецензент к.т.н.. доцент, зав.каф. МБІС
Карпінець В. В.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

"13" 06 2025 р.

Вінниця ВНТУ– 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедраобчислювальної техніки
Освітньо—кваліфікаційний рівень бакалвр
Галузь знань — 12 Інформаційні технології
Спеціальність— 123 Комп'ютерна інженерія
Освітня програма— Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедриОТ
проф., д.т.н.
_____Азаров О. Д.
" 21 " березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУСТУДЕНТУ

Зятю Максиму Владиславовичу

1 Тема роботи: Система віддаленого контролю та керування IoT пристроями, керівникКолесник Ірина Серніївна, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу № 97 від « 20 » березня 2025р.

2 Строк подання студентом роботи 13.05.2025 р.

3 Вихідні дані до роботи: технічний опис архітектури та технологій IoT: платформа Raspberry Pi, Wi-Fi адаптер Auspi Wireless Adapteri, SSH-з'єднання, мова програмування Java та фреймворк Spring Boot.

4 Зміст розрахунково-пояснювальної записки:вступ, аналіз архітектури та технологій інтернету речей, апаратна частина та налаштування засобів віддаленого контролю й управління IO.

5 Перелік графічного матеріалу: Організація структури IoT.

Перелік ілюстративного матеріалу: Схема шини GPIO платформи RaspberryPi.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Колесник І.С.	21.03.2025р.	13.06.2025
Нормоконтроль	відповідальний за н.к. Швець С.І.	14.05.2025р.	30.05.2025

7 Дата видачі завданнязавдання 21.03.2025 р.

8 Календарний планприведений в таблиці2.

Таблиця 2— Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпи
1	Постановка задачі роботи	21.03.25	
2	Пошук матеріалів по технологіях IoT	21.03.25— 29.03.25	
3	Огляд існуючих рішень для моніторингу та керування IOT-пристроями	21.03.25— 29.03.25	
4	Встановлення та налаштування системи Domoticz на Raspberry Pi	30.03.25— 11.04.25	
5	Створення програмного забезпечення для збору, передачі та візуалізації даних	12.04.25— 26.04.25	
6	Тестування та демонстрація роботи системи	27.04.25— 07.05.25	
7	Оформлення пояснювальної записки та ілюстративного матеріалу	08.05.25	
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	14.05.25	

Студент Зять МАКСИМ

Керівник роботи к.т.н., доц.Ірина КОЛЕСНИК

АНОТАЦІЯ

УДК 004.3

Зять М. В.

Система віддаленого контролю та управління IoT-пристроями

Бакалаврська дипломна робота зі спеціальності 123 — комп’ютерна інженерія, освітня програма — системне програмування. Вінниця: ВНТУ, 2025, 93 с.

На укр.мові. Бібліогр.: 24 назв, рис. 43, табл. 3.

У бакалаврській кваліфікаційній роботі розроблено систему віддаленого контролю та управління IoT-пристроями з використанням сучасних апаратних і програмних засобів. У ході дослідження проаналізовано методи організації мереж Інтернету речей (IoT), особливості обміну даними між пристроями, а також визначено вимоги до створення ефективної системи моніторингу та управління.

В процесі роботи здійснено огляд популярних протоколів зв’язку, які використовуються в IoT-системах, таких як MQTT, HTTP та CoAP. На основі цього аналізу обрано оптимальні рішення для побудови безпечної та стабільної мережі. Для реалізації програмної частини використано мову програмування Java та фреймворк Spring Boot, що дозволило забезпечити надійний серверний функціонал.

Розроблена система інтегрована з платформою Domoticz та працює на мікрокомп'ютері Raspberry Pi, що забезпечує віддалене керування сенсорами та іншими IoT-пристроями. В рамках практичної частини проведено тестування працездатності системи, яке підтвердило її ефективність та можливість використання для автоматизованого контролю в розподілених IoT-мережах.

Ключові слова: Інтернет речей, віддалене управління, сенсори, IoT, Raspberry Pi, Domoticz, Java, Spring Boot, моніторинг.

ABSTRACT

Ziat M. V.

Remote Control and Management System for IoT Devices. Bachelor's thesis in specialty 123 - computer engineering, educational program - system programming. Vinnytsia: VNTU, 2025, 93 p.

In Ukrainian. Bibliography: 24 titles, fig. 43, table. 3.

Bachelor’s thesis is dedicated to the development of a remote control and management system for IoT devices using modern hardware and software solutions. During the research, methods of organizing Internet of Things (IoT) networks, features of data exchange between devices, and requirements for creating an effective monitoring and control system were analyzed.

As part of the work, a review of popular communication protocols used in IoT systems, such as MQTT, HTTP, and CoAP, was conducted. Based on this analysis, optimal solutions were selected to build a secure and stable network. The software component was implemented using the Java programming language and the Spring Boot framework, ensuring reliable server functionality.

The developed system is integrated with the Domoticz platform and operates on a Raspberry Pi microcomputer, enabling remote control of sensors and other IoT devices. In the practical part, system performance testing was carried out, confirming its efficiency and suitability for automated control in distributed IoT networks.

Keywords: Internet of Things, remote control, sensors, IoT, Raspberry Pi, Domoticz, Java, Spring Boot, monitoring.

ЗМІСТ

ВСТУП

Актуальність представленої роботи пов'язана з тим, що зі стрімким розвитком цифрових технологій, Інтернет речей (IoT) став однією з ключових складових сучасних інформаційних систем. Усе більше фізичних об'єктів отримують можливість підключення до мережі Інтернет, що дозволяє здійснювати моніторинг, збір та обробку даних у режимі реального часу. Такий підхід відкриває нові горизонти в автоматизації процесів у промисловості, побуті, аграрному секторі, медицині та інших сферах.

Одним з актуальних викликів сьогодення є забезпечення ефективного та безпечного керування пристроями, підключеними до мережі IoT. Це потребує створення гнучких та масштабованих рішень, здатних забезпечити надійний обмін даними, контроль стану пристроїв, а також оперативне реагування на зміну умов середовища. Саме тому дослідження та розробка системи для дистанційного керування IoT-пристроями є надзвичайно важливою та своєчасною задачею.

Метою роботи є розробка програмно-апаратного комплексу для віддаленого моніторингу та управління пристроями Інтернету речей.

Задачі дослідження бакалаврської дипломної роботи:

— аналіз проблем та вимог до вебзастосунків та telegram-ботів;

— проаналізувати сучасні підходи до побудови IoT-систем та їх архітектурні особливості;

— дослідити існуючі технології, протоколи та засоби для реалізації зв’язку між IoT-пристроями;

— підібрати оптимальні апаратні компоненти для реалізації IoT-рішення;

— створити програмне забезпечення для збору, передачі та візуалізації даних;

— протестувати систему в умовах, наближених до реального використання, та оцінити її ефективність.

Об’єкт дослідження—система моніторингу та управління пристроями Інтернету речей.

Предметом дослідженняє методи побудови програмно-апаратних рішень для дистанційного контролю IoT-пристроїв, а також технології обміну даними між компонентами таких систем.

Практичне значення отриманих результатів. У межах роботи було створено повнофункціональну систему, що може бути використана для впровадження у сферах, де необхідний віддалений контроль технічних пристроїв. Розроблене рішення може бути адаптоване для використання в аграрному секторі, енергетиці, системах «розумного будинку» тощо.

У роботі застосовуються такі м**етоди дослідження**, як методи аналізу і синтезу інформаційних систем, структурного та об'єктно-орієнтованого проектування, моделювання процесів у комп’ютерних мережах, а також експериментальні методи дослідження роботи розробленої системи.

1 АНАЛІЗ АРХІТЕКТУРИ ТА ТЕХНОЛОГІЙ ІНТЕРНЕТУ РЕЧЕЙ

Термін «Інтернет речей» був введений Кевіном Ештоном у 1997 році, коли він застосував технологію радіочастотної ідентифікації (RFID) для оптимізації системи постачання. Завдяки цій роботі його запросили до Масачусетського технологічного інституту, де разом з колегами він заснував Auto-ID Center — дослідницький консорціум. Відтоді концепція Інтернету речей еволюціонувала від простих RFID-міток до складних екосистем.

Одним з перших пристроїв, що стали частиною Інтернету речей, був автомат з газованими напоями, підключений до інтернету в університеті Карнегі-Меллон в 1982 році. Згодом інші об'єкти також почали отримувати підключення до мережі. У 1991 році компанія HP випустила перший мережевий принтер, а в 1993 році в Кембриджському університеті підключили першу камеру до Інтернету. У 1998 році створення організації Bluetooth SIG стало важливим кроком у розвитку IoT, оскільки технологія Bluetooth значно зменшила енергоспоживання, що дозволило збільшити автономність пристроїв.

У 2000 році компанія HP представила концепцію «Cooltown» — ідею всепроникної комп'ютеризації, що об'єднувала обчислювальні і комунікаційні технології для створення постійного підключення до Інтернету для людей, місць і об'єктів. У 2005 році Міжнародний союз електрозв'язку, спеціалізована установа ООН, опублікував звіт, у якому були визначені основні тенденції розвитку Інтернету речей. У 2008 році вдосконалення технології напівпровідникових світлодіодних ламп сприяло розвитку концепції «розумного» освітлення, що стало основою для створення «розумного» дому.

Сьогодні, у 2025 році, Інтернет речей продовжує розвиватися і охоплює всі сфери діяльності. Прогнозується, що в найближчі роки IoT проникне ще глибше в промисловість, бізнес, охорону здоров'я, транспорт і побут. Із розвитком технологій штучного інтелекту, великих даних та 5G, пристрої IoT стають все більш інтегрованими, забезпечуючи нові можливості для автоматизації і оптимізації. Вже зараз ми бачимо нові інновації в розумних

містах, де Інтернет речей використовують для управління енергоспоживанням, транспорту, безпекою і навіть для медичних послуг.

1.1 Структура IoT

Система Інтернету речей включає різні компоненти, сервіси та технології, які використовуються в цій сфері. Їх можна умовно поділити на нижче приведені категорії.

Сенсори та розумні датчики—це вбудовані системи, операційні системи реального часу, джерела безперебійного живлення, мікроелектромеханічні системи, що збирають і передають дані.

Системи зв'язку для датчиків — це є мережі з покриттям бездротових персональних мереж, які можуть охоплювати діапазон від 0 см до 100 метрів. Для обміну даними між датчиками використовуються малопотужні канали з низькою швидкістю, часто без використання IP-протоколів.

Локальні мережі (LAN),зазвичай це мережі, які використовують протокол IP, наприклад, Wi-Fi мережі на основі стандарту 802.11, що дозволяють забезпечити швидкий радіозв'язок. Ці мережі часто мають пірінгову або зіркоподібну структуру.

Агрегатори, маршрутизатори, шлюзи і пограничні пристрої— це ключові елементи для підключення і обміну даними між різними частинами Інтернету речей, включаючи вбудовані системи, процесори, оперативну пам'ять і системи зберігання даних, а також постачальників бездротових і стільникових технологій, міжплатформових рішень і інструментів для аналізу даних на периферії.

Глобальні обчислювальні мережі— це інфраструктура, надається операторами стільникового, супутникового зв'язку та операторами мереж низької потужності (LPWAN), які забезпечують глобальне підключення і обмін даними.

Транспортні протоколи для IoT: для передачі даних між пристроями часто використовуються протоколи, такі як MQTT, CoAP та HTTP, що забезпечують ефективну комунікацію в IoT-мережах.

Хмара— це інфраструктура та платформи, що надають хмарні послуги, включаючи бази даних, поточну та пакетну обробку даних, аналітичні інструменти, сервіси машинного навчання, озера даних і програмно-визначені мережі.

Аналіз даних: великі обсяги інформації передаються в хмару для обробки. Оскільки ці дані потребують комплексного аналізу, зазвичай використовуються методи обробки подій, аналітики та машинного навчання для отримання корисної інформації.

Безпека: оскільки Інтернет речей об'єднує багато різних пристроїв і технологій, важливо забезпечити їх безпеку на кожному етапі. Це включає захист датчиків, процесорів, систем радіозв'язку і самих протоколів передачі даних від потенційних загроз і кібератак. Забезпечення безпеки є критично важливим, оскільки IoT може стати привабливою мішенню для хакерів.

Загалом, компоненти Інтернету речей можна розподілити на чотири основні категорії: сервіси, апаратні засоби, програмне забезпечення та набір правил. Ці категорії можна також деталізувати на підгрупи залежно від їх призначення (див. рис. 1.1).

Архітектура Інтернету речей може варіюватися залежно від її цілей та специфіки реалізації. Вона нагадує традиційну структуру автоматизованих систем управління технологічними процесами (АСУТП) (рис. 1.2).

Взаємодія з навколишнім середовищем здійснюється через датчики (sensors) та виконувані механізми (actuators), як це реалізовано в АСУТП для управління різними об'єктами. Ці датчики, разом з іншими компонентами інфраструктури, які забезпечують інтеграцію з системою обробки подій через Інтернет, утворюють так звану граничну зону (Edge).

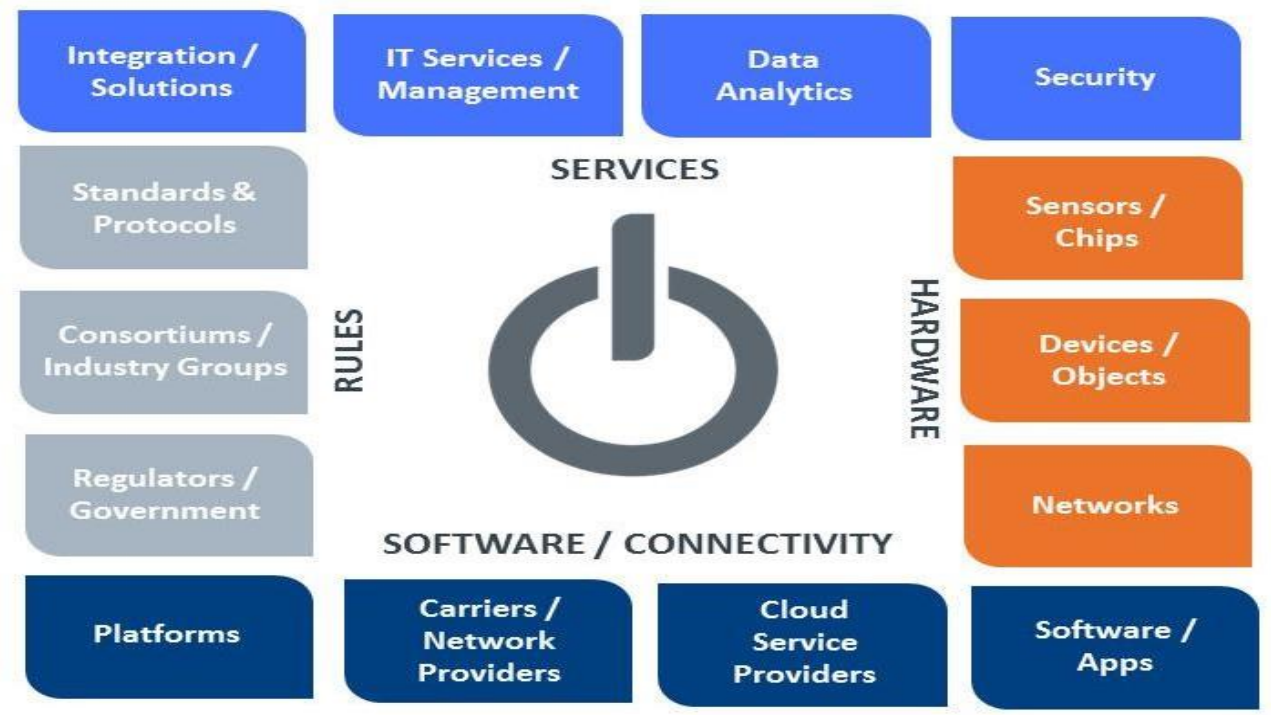


Рисунок 1.1 — Схема взаємодії засобів IoT

1.2 Розгляд архітектурних підходів до IoT

Дані, що надходять з граничної зони, зберігаються та обробляються в залежності від завдань, через платформу для обробки подій та аналітики (event processing, Platform).

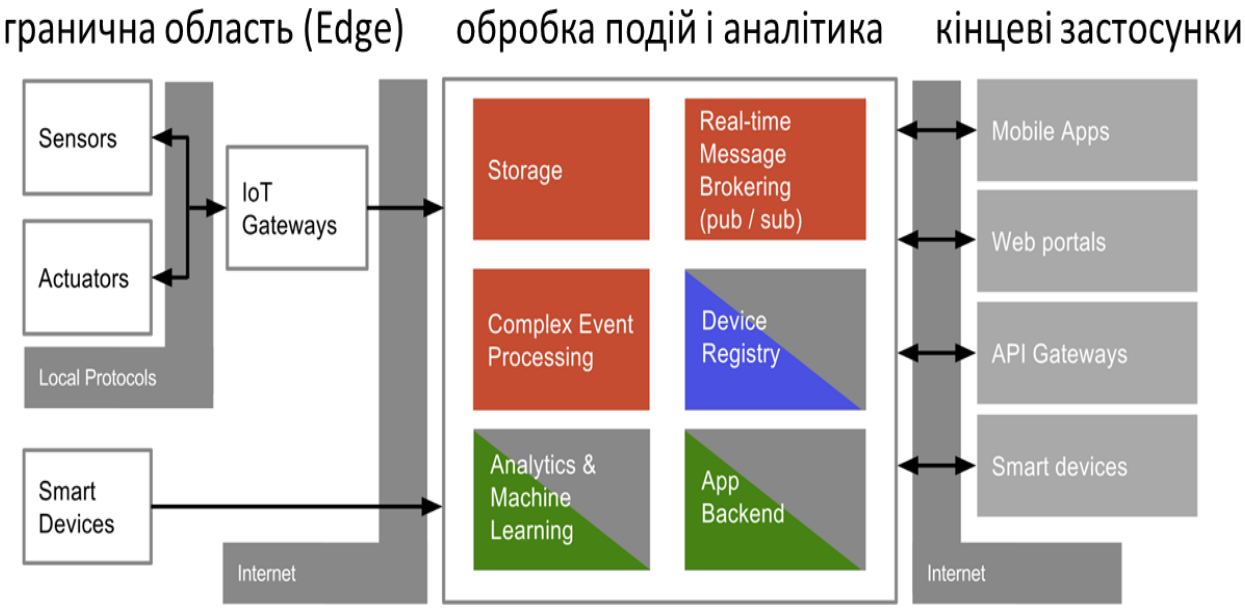


Рисунок 1.2 —Організація структури IoT

На цьому рівні відбуваються процеси збереження даних (storage), їх обробка (Event Processing), а також передача до необхідних додатків (Real-Time Message Brokering, Stream Processing). Крім того, на цьому етапі здійснюється адміністрування та управління пристроями, що знаходяться на межі (Device Registry, Edge Device Management). На основі отриманих подій проводиться машинне навчання (Machine Learning), що дозволяє робити висновки щодо об'єкта з використанням аналітичних сервісів (Analytics). Цей рівень зазвичай реалізується за допомогою хмарних (Cloud) або туманних (Fog) обчислень. У порівнянні з АСУТП, цей рівень відповідає контролерам та SCADA системам (без функцій HMI). Отримання результатів, контроль, віддалене управління та адміністрування системи здійснюються через кінцеві додатки з використанням Інтернету.

На рисунку 1.3 показана архітектура, подібна до описаної вище, але у вигляді сервісів. Тут область Edge представлена як датчики, також до неї віднесено Device Hub/Gateway (збір та маршрутизація даних) і Device Management (управління пристроями), що частково реалізуються як хмарні обчислення або на граничних пристроях. Функції збереження та первинної обробки подій (даних) зведені до Data Management. Усі інші функції обробки, включаючи аналітику, представлені як PaaS додатки, які взаємодіють з сервісами управління даними через API (Application Program Interface).

Ще один варіант архітектурної побудови IoT-системи, цього разу з деталізацією за модулями, представлено на рисунку 1.4. Зі схеми можна помітити, що всі розглянуті архітектури мають спільні характерні елементи: трирівневу структуру, схожі функціональні блоки, активне застосування хмарних технологій, а також використання мережі Інтернет як основної платформи інтеграції компонентів.

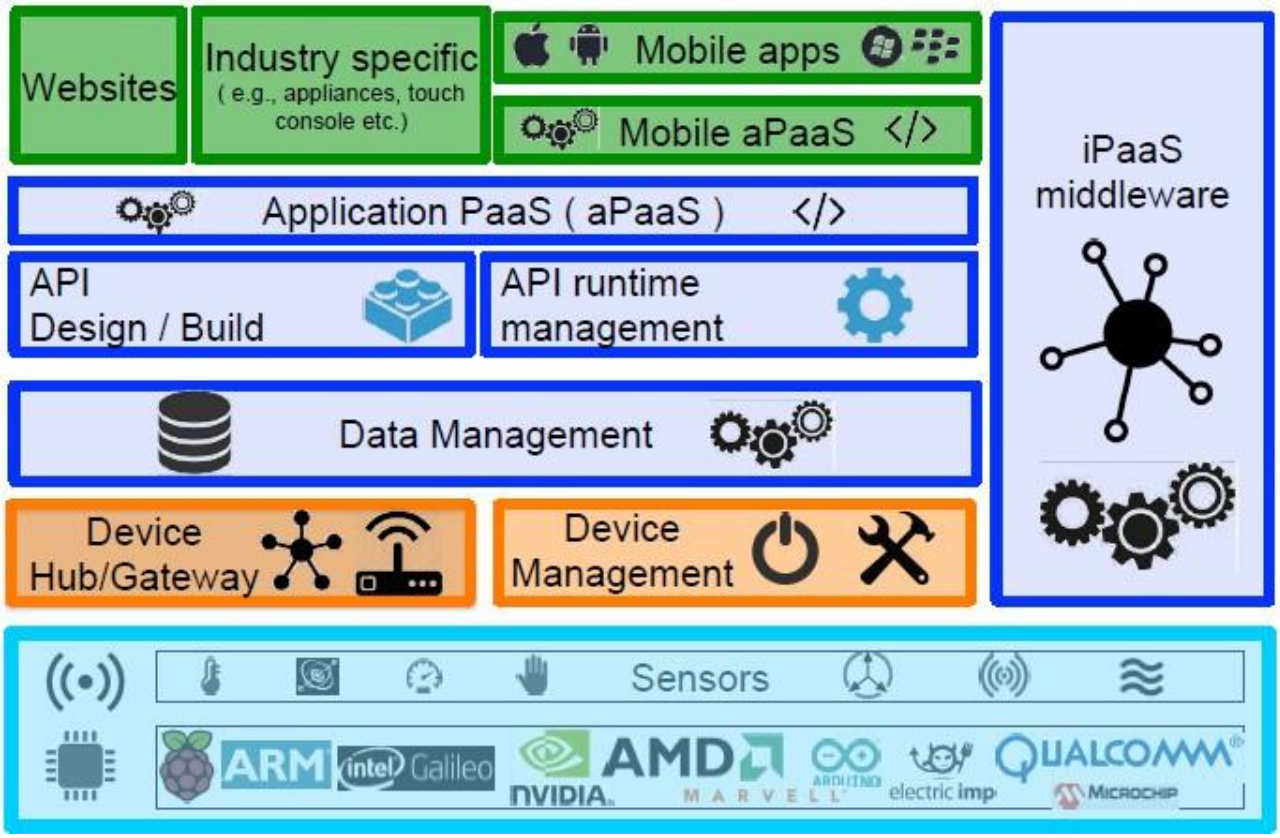


Рисунок 1.3 — Компонентна побудова IoT-системи в контексті сервісів

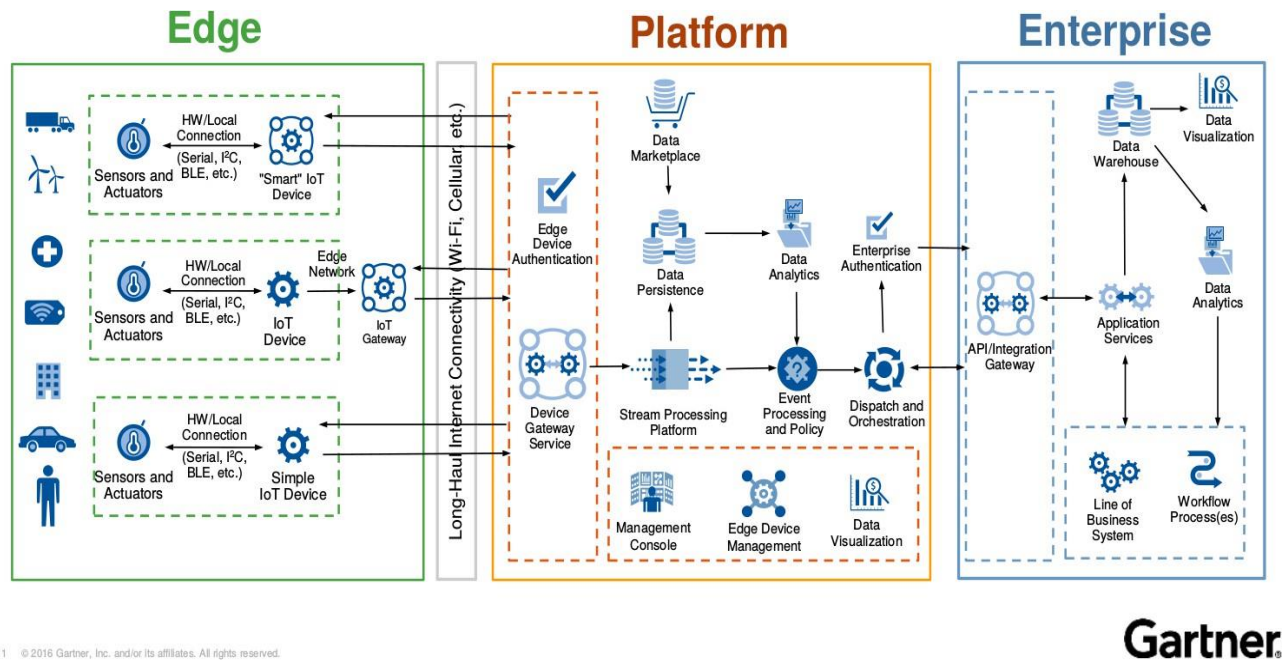


Рисунок 1.4 — Архітектурна модель IoT з розподілом на модулі

1.3 Основні компоненти архітектури IoT

До основних компонентів IoT, насамперед, можна віднести датчики та засоби для енергозабезпечення.

Початок або завершення взаємодії в системах Інтернету речей зазвичай пов’язане з певною фізичною подією: зміна температури, фіксація руху, натискання кнопки чи активація замка. На відміну від звичних IT-рішень, де основна увага приділяється обробці цифрових сигналів, IoT напряму пов'язаний із впливами з реального світу та реагуванням на них.

В деяких випадках навіть один датчик може продукувати великі обсяги інформації, — наприклад, при використанні акустичних сенсорів для моніторингу технічного стану обладнання. Водночас бувають ситуації, коли достатньо передати всього один біт даних, щоб повідомити про критичний параметр, як-от стан здоров’я людини (див. рис. 1.5).

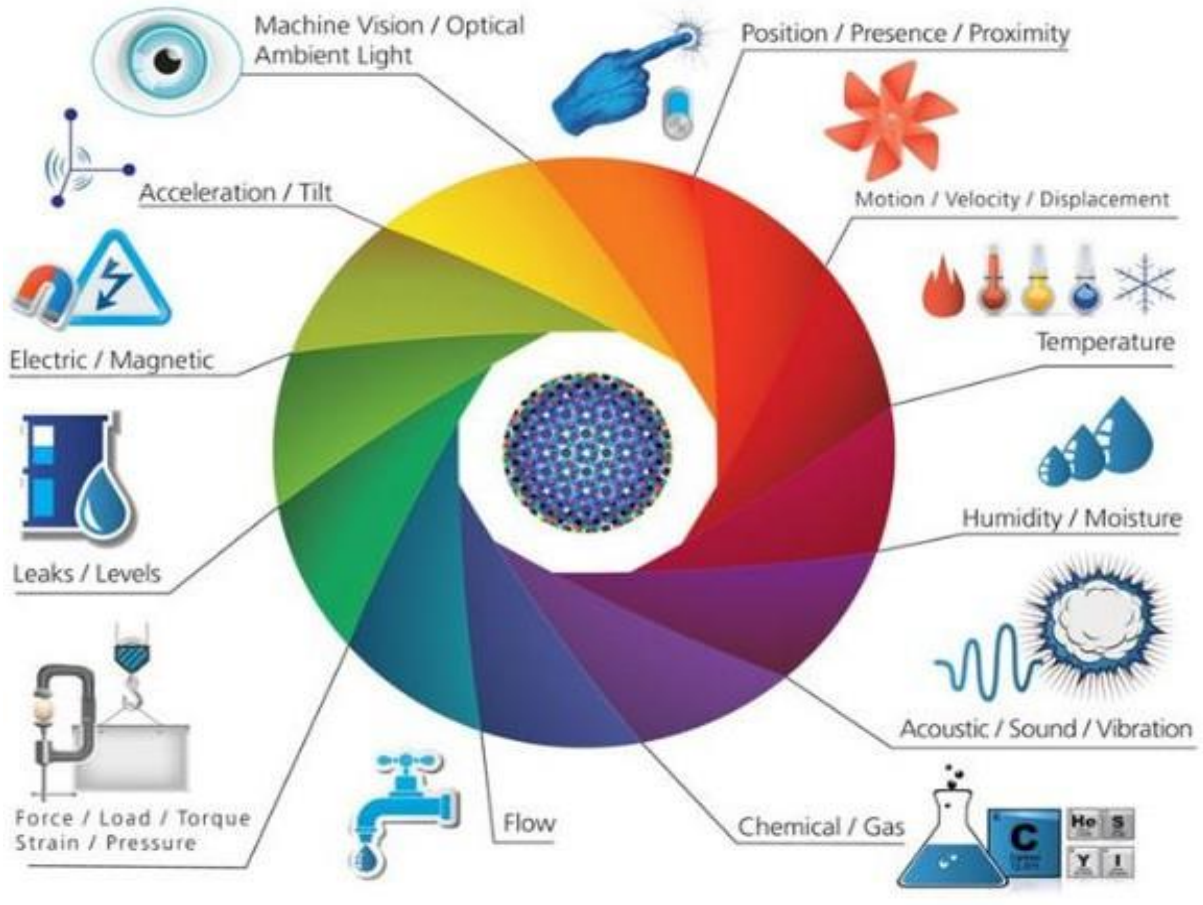


Рисунок 1.5 — Схема фізичних подій, що підлягають обробці засобами IoT

Завдяки технічному прогресу, сенсорні системи стали компактнішими, економнішими та доступнішими — ця тенденція відповідає закону Мура. Зменшення розмірів до нанометрового рівня та здешевлення компонентів робить можливим масштабне розгортання IoT-рішень. Саме на ці технічні досягнення спираються прогнози щодо підключення мільярдів пристроїв до Інтернету речей у найближчому майбутньому.

Отже, при аналізі Інтернету речей важливо брати до уваги мікроелектромеханічні системи, різноманітні сенсори, а також інші недорогі пристрої граничного рівня разом із їхніми електрофізичними характеристиками. Окрему увагу слід приділяти системам живлення та енергозабезпечення, які необхідні для стабільної роботи цих пристроїв. Не можна припускати, що живлення граничних компонентів забезпечується автоматично — навіть мільярди мініатюрних сенсорів вимагають суттєвих енергетичних ресурсів. Питання енергоспоживання тісно пов’язане також з особливостями побудови хмарної інфраструктури IoT-систем [1].

Наступний аспект, на який слід звернути увагу , це передача інформації.

Одним із ключових аспектів розробки систем Інтернету речей є забезпечення стабільного мережевого з’єднання та ефективної передачі даних. Без надійних засобів комунікації, здатних транспортувати інформацію з віддалених або складних для доступу місць до потужних центрів обробки даних компаній на кшталт Google, Amazon, Microsoft чи IBM, концепція IoT була б просто неможливою.

Оскільки в самому терміні «Інтернет речей» закладено слово «Інтернет», то логічно, що вивчення мережевих технологій, протоколів обміну даними та навіть основ теорії сигналів є надзвичайно важливим. Суть IoT полягає не лише у зборі даних через сенсори або обробці через програми — ключовим чинником є можливість забезпечити підключення.

Передача даних у таких системах зазвичай реалізується через мережі малого радіусу дії (PAN — Personal Area Network), які часто функціонують поза

межами стандартного IP-протоколу. Це можуть бути як дротові, так і бездротові з'єднання. Серед бездротових технологій, які активно використовуються в IoT, можна виокремити Bluetooth, сіткові (mesh) мережі, Zigbee, Z-Wave (див. рис. 1.6), а в промисловому сегменті — IIoT (Industrial Internet of Things) — такі протоколи, як WirelessHART та ISA100 [2].

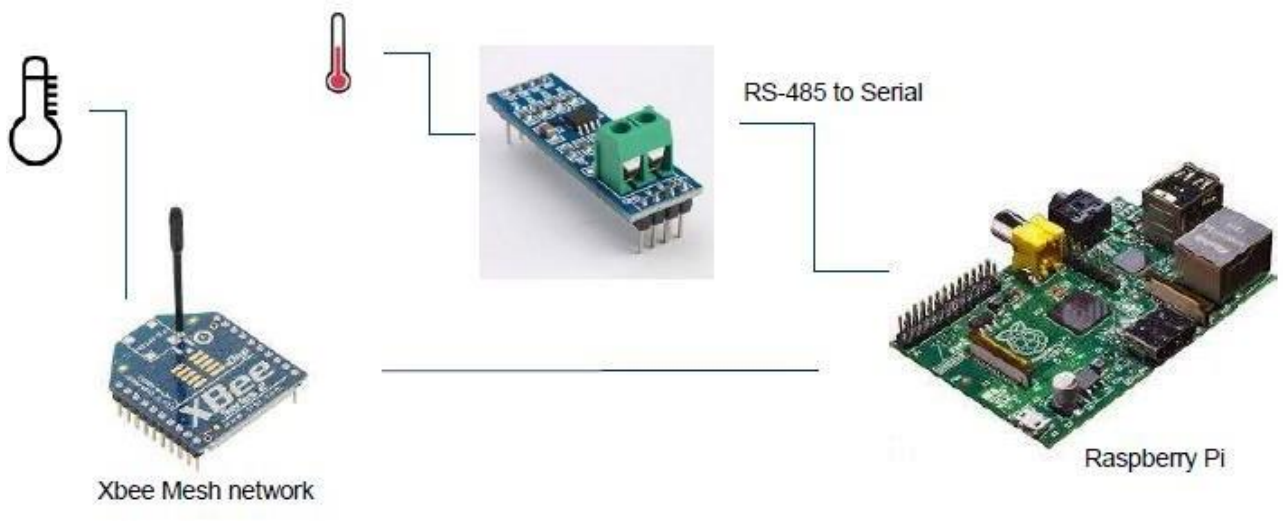


Рисунок 1.6 — Пристрої комунікації в мережах ближнього радіусу дії (PAN)

Окрім персональних мереж (PAN), в системах Інтернету речей активно застосовуються бездротові локальні мережі (WLAN) та інші комунікаційні рішення, що базуються на IP-протоколі. Зокрема, широко використовуються Wi-Fi технології, які відповідають стандартам IEEE 802.11, а також інноваційні протоколи, як-от 6LoWPAN та Thread.

Для забезпечення зв'язку в умовах великого покриття часто залучаються телекомунікаційні технології, що базуються на стільникових мережах — таких як 3G та 4G LTE. Також активно впроваджуються новітні рішення, створені спеціально для IoT та машинної взаємодії (M2M), зокрема Cat-1, NB-IoT (Cat-NB) і низка вузькоспеціалізованих протоколів, таких як LoRaWAN та Sigfox. Останні відзначаються низьким енергоспоживанням і здатністю передавати

дані на великі відстані, що робить їх ідеальними для IoT-пристроїв (див. рис. 1.7).

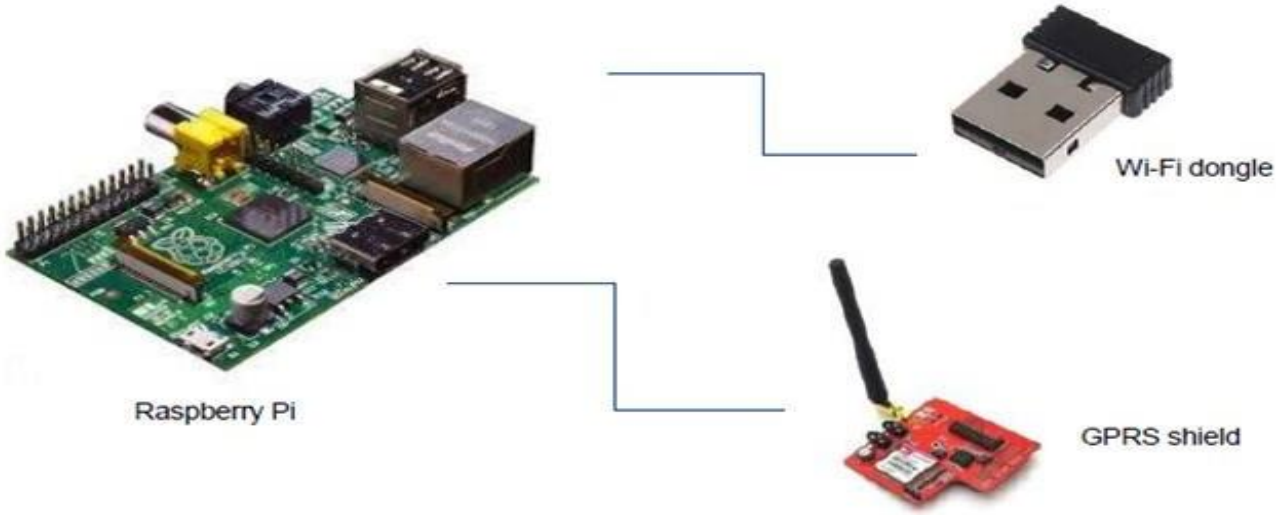


Рисунок 1.7 —Обладнання для обміну даними у бездротових LAN-мережах

Далі мова йтиме про маршрутизацію.Для ефективної передачі даних з датчиків до Інтернет-простору необхідні дві основні технології: шлюзи-маршрутизатори та відповідні інтернет-протоколи, які забезпечують надійний і швидкий обмін даними. Маршрутизатор відіграє важливу роль у таких аспектах, як безпека, управління трафіком і напрямки даних. Граничні маршрутизатори (Edge routers) забезпечують моніторинг та управління mesh-мережами, а також відповідають за покращення та підтримку якості переданих даних.

Особливу увагу слід приділяти захисту та конфіденційності переданої інформації. Маршрутизатор є ключовим компонентом у створенні віртуальних приватних мереж (VPN), віртуальних локальних мереж (VLAN) та програмно-налаштовуваних глобальних мереж (SD-WAN). Вони можуть підтримувати тисячі вузлів, які обслуговуються одним граничним маршрутизатором, що виступає як своєрідний міст для з'єднання з хмарними сервісами (edge device) [2].

На цьому рівні використовується ряд протоколів для забезпечення обміну даними між вузлами, маршрутизаторами та хмарними сервісами в межах IoT-

системи. Інтернет речей сприяв появі нових спеціалізованих протоколів, що працюють на рівні традиційних технологій, таких як HTTP і SNMP, які використовуються вже десятки років. Для ефективної передачі IoT-даних необхідні протоколи з низьким енергоспоживанням і мінімальною затримкою, здатні безпечно і ефективно передавати дані між пристроями та хмарними сервісами. Серед таких протоколів часто застосовуються MQTT, AMQP та CoAP, які оптимізовані для роботи в умовах Інтернету речей.

На цьому етапі важливо визначити, як обробляти потік даних, що надходять від граничного пристрою (Edge Device) у хмарний сервіс. Для того, щоб правильно оцінити, як система буде розвиватися, необхідно детально розглянути архітектуру хмарних систем та вплив затримок на IoT-систему. Крім того, не всі дані варто відправляти в хмару, оскільки пересилання всіх IoT-даних може бути надто дорогим. Замість цього обробка даних на рівні граничних обчислень (Edge Computing) або в межах туманних обчислень (Fog Computing) є більш економічно вигідним варіантом. Туманні обчислення також стандартизуються, і одним із прикладів є архітектура OpenFog.

Дані, що зібрані шляхом перетворення фізичних впливів у цифрові сигнали, можуть бути дуже об'ємними. На цьому етапі важливу роль відіграють аналітичні інструменти та процесори правил IoT-систем. Складність впровадження IoT-системи залежить від проектованого рішення. У деяких випадках завдання може бути досить простим: наприклад, встановлення простого процесора правил на граничному маршрутизаторі для моніторингу аномальних змін температури. В інших випадках, коли в реальному часі передаються великі обсяги структурованих та неструктурованих даних в хмарне сховище, потрібні високі швидкості обробки для прогнозної аналітики та довгострокового прогнозування, що потребує використання складних моделей машинного навчання, таких як рекурентні нейронні мережі для аналізу сигналів з кореляцією в часі. У таких ситуаціях виникають певні труднощі, які

можна вирішити за допомогою складних обробників подій, байесівських мереж і створення нейронних мереж.

Загрози та безпека в IoT:багато IoT-систем не обмежуватимуться лише безпечними зонами, такими як будинки або офіси. Вони можуть бути розташовані в громадських місцях, віддалених районах, на транспортних засобах, а також вбудовані в тіло людини. Інтернет речей є привабливою мішенню для всіх видів хакерських атак. Уже зафіксовано численні організовані зломи, атаки, спрямовані на IoT-пристрої, а також вразливості, які можуть мати національні масштаби. Розробники рішень для Інтернету речей повинні розуміти специфіку таких вразливостей та способи їх усунення, а також застосовувати стандартні заходи для забезпечення безпеки систем та компонентів мережі.

1.4 Аналіз технологій та протоколів обміну даними в IoT-мережах

Оскільки концепція Інтернету речей передбачає, що пристрої будуть розміщені на невеликій відстані один від одного і не завжди зможуть бути підключені до постійного джерела живлення, необхідні спеціалізовані технології і протоколи для передачі даних. Ось кілька популярних технологій:

— Z-Wave — радіотехнологія, що працює на частотах до 1 ГГц, ідеально підходить для передачі простих команд з низькою затримкою;

— NFC (Near Field Communication) — дозволяє передавати дані на відстані до 20 см.;

— RFID (Radio Frequency Identification) — технологія для ідентифікації об'єктів за допомогою радіосигналів;

— BLE (Bluetooth Low Energy) — варіант Bluetooth з низьким енергоспоживанням;

— Wi-Fi HaLow — працює на частоті 900 МГц, має енергоспоживання на рівні Bluetooth, але з вищою швидкістю передачі даних у порівнянні з Bluetooth.

Далі розглянемо ці технології детальніше і визначимо їх призначення.

Z-Wave — це бездротова радіотехнологія, що відзначається низьким енергоспоживанням. Z-Wave працює в діапазоні частот до 1 ГГц, і вона оптимізована для передачі простих команд управління з мінімальними затримками, таких як перемикання каналів або вимикання пристроїв. Частотний діапазон вибраний спеціально для зниження перешкод від інших технологій, що використовуються широким колом людей, наприклад, Wi-Fi, який працює на частотах 2,4 ГГц і 5 ГГц.

Попри свою складність, система Z-Wave надзвичайно проста у використанні. Вона також є енергозберігаючою та допомагає заощадити час користувачів. Система, що функціонує за допомогою дистанційного керування і використовує радіохвилі малої потужності, є високонадійною, оскільки забезпечує покриття всіх зон будинку. Радіохвилі здатні проходити через стіни, поверхи та меблі, що забезпечує стабільну роботу мережі.

Near Field Communication (NFC) — це технологія зв'язку на малих відстанях, призначена для обміну різною інформацією, такою як зображення, музичні файли, номери телефонів або ключі цифрової авторизації, між двома пристроями, що підтримують NFC. Це можуть бути смарт-картки, портативні пристрої або зчитувальні пристрої RFID. Технологія також дозволяє використовувати NFC як ключ доступу до даних або послуг.

NFC дає змогу здійснювати передачу даних не тільки від активного пристрою до пасивного, а й між двома активними пристроями. Вона застосовується для взаємодії з пристроями радіочастотної ідентифікації RFID. Для забезпечення сумісності між RFID-картками та мобільними телефонами різних виробників перевіряють цифровий протокол і вимірюють важливі параметри радіочастотного сигналу, такі як тимчасові характеристики, чутливість і амплітуда приймача в активному режимі, а також частота несучої амплітуди сигналу.

При передачі інформації від активного пристрою до пасивного використовують амплітудну маніпуляцію (ASK). У процесі обміну даними

обидва пристрої мають рівні права. Кожен пристрій має власне джерело живлення, тому сигнал несучої вимикається одразу після завершення передачі (див. рис.1.8).

Рисунок 1.8 —Принцип роботи технології NFC

Завдяки індуктивному зв'язку між активним та пасивним пристроєм, пасивний пристрій впливає на активний. Зміна опору в пристрої, який слухає, викликає зміну амплітуди або фази напруги на антени опитуваного пристрою. Після цього відбувається з'єднання двох пристроїв, і передача даних стає можливою. Якщо пристрої віддаляються більше ніж на 20 см, електромагнітне з'єднання розривається, і передача даних припиняється автоматично.

Технологія NFC є фактично розширенням технології RFID, що широко використовується у безконтактних картках і мітках. Проте NFC має можливість не тільки зчитувати інформацію з пасивних електронних міток, але й забезпечувати двосторонній бездротовий зв'язок між пристроями.

RFID (Radio Frequency Identification) — це метод автоматичної ідентифікації об'єктів за допомогою радіосигналів, який дозволяє зчитувати або записувати дані, що зберігаються у транспондерах або RFID-мітках. Будь-яка RFID-система складається з зчитувального пристрою (рідер) та транспондера (RFID-мітка або RFID-тег).

RFID-мітки зазвичай складаються з двох основних частин. Перша — це інтегральна схема, яка відповідає за обробку та зберігання інформації, демодулювання і модуляцію радіочастотного сигналу, а також виконання інших

функцій. Друга частина — антена, яка забезпечує прийом та передачу сигналу. Крім того, для функціонування таких міток необхідне спеціальне програмне забезпечення, яке дозволяє збирати та аналізувати інформацію, отриману з RFID-міток.

RFID-мітки бувають двох типів:

— активні мітки, що мають власне джерело живлення і можуть самостійно посилати сигнали, що дозволяє зчитувати їх на значній відстані;

— пасивні мітки, які не мають власного джерела енергії і активізуються тільки після отримання сигналу від пристрою зчитування, після чого передають збережену інформацію.

RFID-мітки (рис. 1.9) використовуються для управління товарними запасами, відстеження часу на спортивних змаганнях, маркування великої рогатої худоби для запису інформації про ветеринарний огляд.

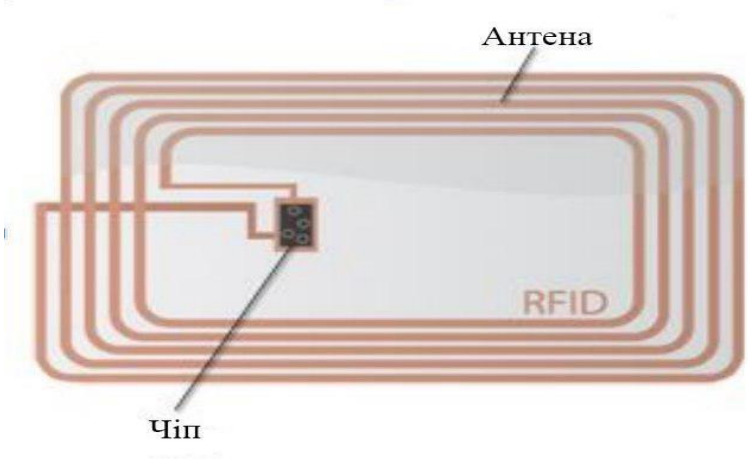


Рисунок 1.9 — Внутрішній вміст RFID-мітки

Вони також доповнюють штрих-коди, забезпечуючи можливість дистанційного зчитування. Вони можуть бути використані для ідентифікації автомобілів, навіть коли вони рухаються на високій швидкості, або для ефективного відстеження багажу в авіації. Крім того, RFID інтегрується в біометричні паспорти та кредитні картки для забезпечення безпечного доступу в певні зони.

Bluetooth Low Energy (BLE)— це бездротова технологія з низьким енергоспоживанням, яка є частиною специфікації Bluetooth, починаючи з версії Bluetooth 4.0 і до Bluetooth 5.0. Пристрої, що використовують BLE, споживають набагато менше енергії в порівнянні з попередніми поколіннями Bluetooth. Це дозволяє їм працювати довше, навіть до року на одній невеликій батарейці типу "таблетка". Така особливість дозволяє використовувати компактні датчики, які можуть постійно функціонувати та взаємодіяти з іншими пристроями.

BLE оптимізовано для малих пристроїв, де важлива компактність, і неможливо встановити великий акумулятор або батарею. Технологія витрачає в 10-20 разів менше енергії, здатна передавати дані в 50 і більше разів швидше і на значно більші відстані (понад 100 метрів) порівняно з традиційними рішеннями Bluetooth. Крім того, BLE забезпечує високу безпеку, надійність, низьку затримку підключення і низьке споживання енергії.

Ще одна ключова особливість BLE полягає в адаптивному переналаштуванні частоти. Це дає змогу технології мінімізувати перешкоди, помилки передачі та інтерференцію, швидко обираючи оптимальну частоту для забезпечення більш стабільної і безпечної комунікації.

Специфікація Bluetooth 5.0 була розроблена з орієнтацією на Інтернет речей (IoT). Це вказує на прагнення стандарта захопити ринок пристроїв, що підтримують IoT. В порівнянні з версією 4.0, швидкість передачі даних в Bluetooth 5.0 значно збільшена, наближаючись до швидкостей HSPA і LTE, в той час як енергоспоживання залишається на колишньому рівні. Одним з найважливіших аспектів для розвитку мереж IoT є енергоефективність. Хоча Bluetooth 5.0 ще не став широко розповсюдженим через свою нещодавню появу, він має зворотну сумісність з попередніми версіями. Це означає, що в майбутньому майже всі мобільні пристрої будуть підтримувати п’яту версію цього стандарту, що є однією з основних переваг цієї технології.

Wi-Fi HaLow — це протокол бездротової мережі, представлений у 2017 році як доповнення до стандарту IEEE 802.11. Він працює на частоті 900 МГц,

яка не потребує ліцензування, що дозволяє забезпечити більший діапазон покриття порівняно з традиційними Wi-Fi мережами, які функціонують у діапазонах 2,4 ГГц і 5 ГГц. Однією з основних переваг Wi-Fi HaLow є низьке споживання енергії, що дозволяє створювати великі мережі датчиків або станцій, які взаємодіють між собою, підтримуючи концепцію Інтернету речей (IoT). Витрата енергії протоколом Wi-Fi HaLow є конкурентною з Bluetooth, але має додаткові переваги, зокрема вищі швидкості передачі даних та більший діапазон покриття (див.рис. 1.10).

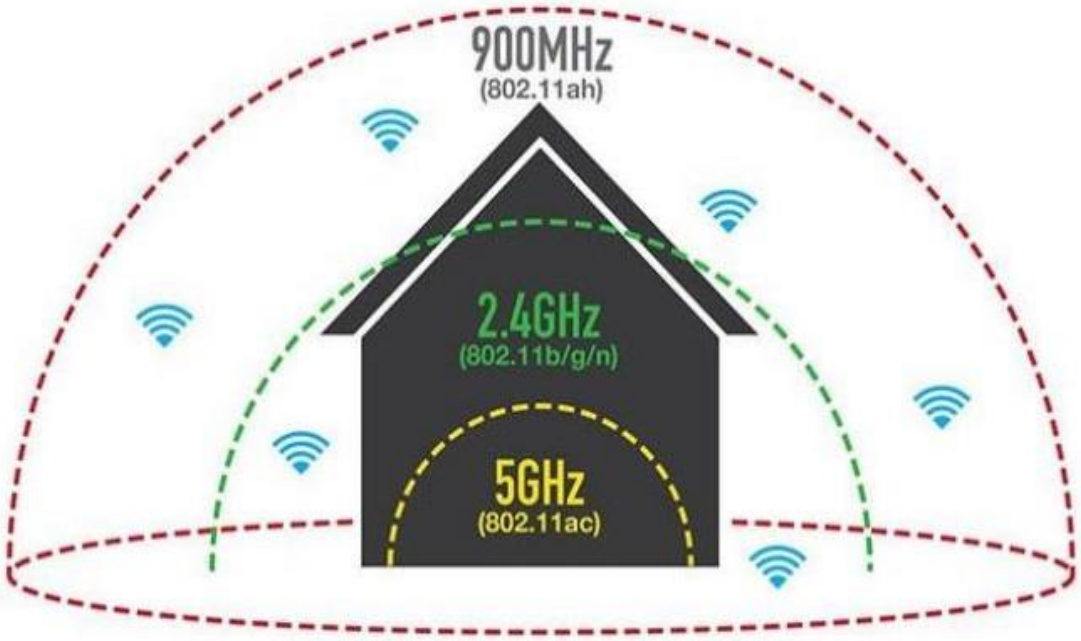


Рисунок 1.10 — Порівняння зон покриття різних протоколів Wi-Fi

Wi-Fi HaLow розширює можливості традиційного Wi-Fi, працюючи в діапазоні 900 МГц. Це дозволяє підключати пристрої з низьким енергоспоживанням, такі як датчики та портативні комп'ютери. Протокол зберігає важливі переваги попередніх версій, зокрема високий рівень захисту даних, сумісність з різними типами обладнання та зручність у налаштуванні. Пристрої, які підтримують Wi-Fi HaLow, можуть також працювати в діапазонах 2,4 та 5 ГГц, що дає змогу інтегрувати їх у глобальну екосистему з більш ніж 7 мільярдами пристроїв. Крім того, протокол підтримує підключення через IP, що

дозволяє взаємодіяти з хмарними технологіями, що є важливим для IoT. Wi-Fi HaLow також забезпечує можливість підключення до 1000 пристроїв до однієї точки доступу.

Порівняльна характеристика технологій та протоколів передачі даних на короткі відстані в IoT представлено в таблиці 1.1.

Таблиця 1.1 —Порівняння основних технічних характеристик мереж з малим радіусом 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 передачі повідомлень в IoT-середовищі

Обсяг інформації, що генерується одним сенсорним вузлом, зазвичай невеликий. Проте більшість сервісів Інтернету речей (IoT) побудовані на принципі обробки даних, отриманих від великої кількості пристроїв або датчиків. Це суттєво відрізняється від архітектур традиційних мереж.

Отже, ми стикаємося з новою архітектурою: багато джерел - багато отримувачів. Водночас обсяг трафіку від сенсорного вузла може варіюватися від дуже маленького до дуже великого. У таких умовах традиційні протоколи передачі даних не можуть бути застосовані.

Основною топологією для передачі даних в IoT є модель "видавець-підписник" (Publisher-Subscriber, або pub/sub) (див. рис. 1.11). У цій схемі є два ключових компоненти: видавець — джерело інформації, та підписник — отримувач інформації. Поняття "передплата" означає, що учасники здійснюють операцію підписки для отримання даних від конкретного видавця, а також визначають параметри, як-от періодичність отримання або інші характеристики (залежно від реалізації).

У цьому контексті розглядається ситуація, коли сенсорний вузол (Node) агрегує інформацію від численних датчиків (наприклад, дані про вологість повітря) та передає її відповідно до параметрів підписки, або за запитом, або через певні інтервали часу. Зазвичай самі датчики є простими, і їх основна задача полягає в постійному зборі даних про контролювані параметри. Це створює необхідність у об'єднанні датчиків у вузли з мікроконтролерами, які відповідають за зчитування вимірюваних даних та їх передачу на сервер за заздалегідь визначеними алгоритмами.

Для взаємодії користувача з IoT-системою зазвичай необхідне спеціальне клієнтське програмне забезпечення (Application), яке встановлюється на персональний пристрій. Це забезпечує зручне візуальне відображення інформації, отриманої безпосередньо з датчиків або вже обробленої на сервері

управління системою. Така архітектура передбачає наявність посередника — так званого брокера (Broker).

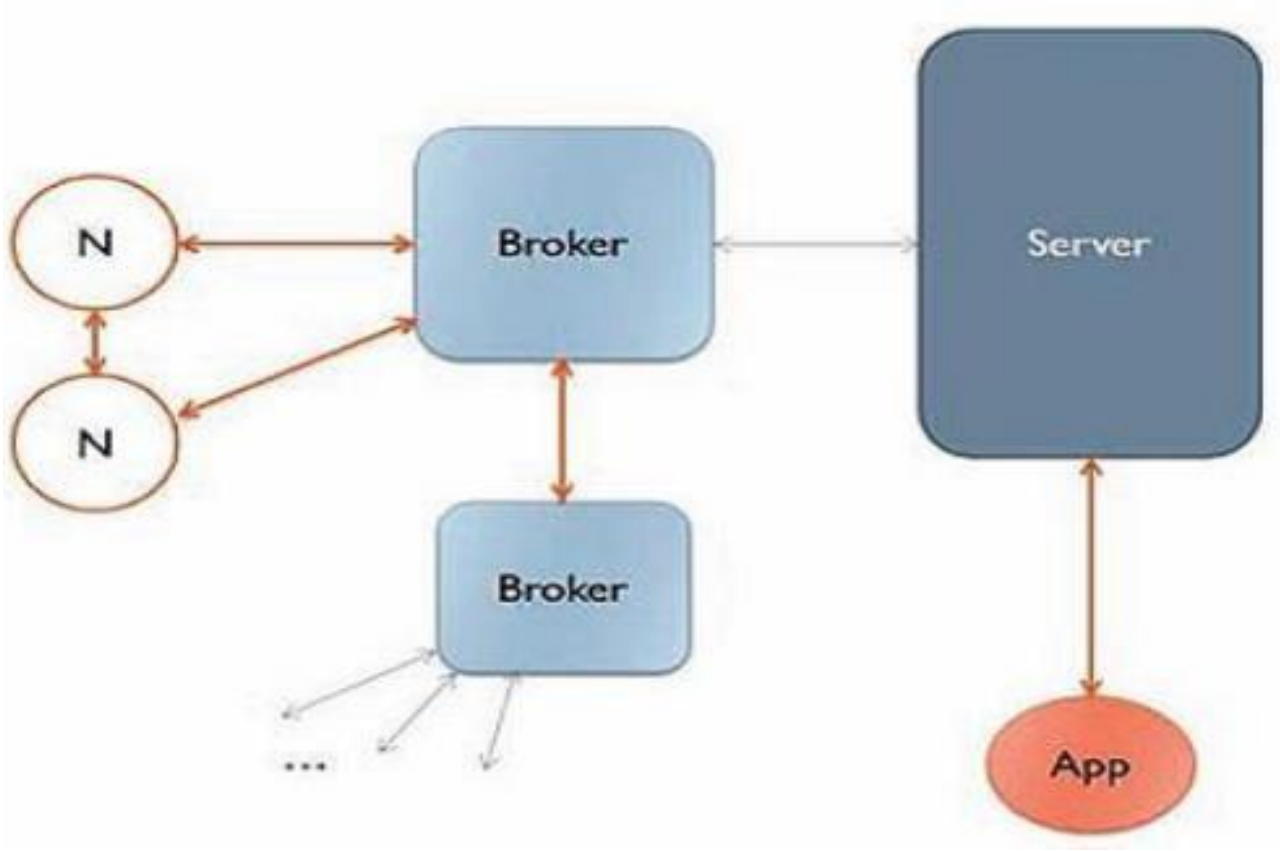


Рисунок 1.11 —Топологія передачі даних у мережі IoT

Брокер виконує функцію посередницького сервера, який отримує повідомлення від джерел даних (видавців) і передає їх відповідним отримувачам (передплатникам). У більш складних IoT-мережах брокер може не лише пересилати дані, але й виконувати їхню попередню обробку, аналіз, встановлення пріоритетів, а також формувати черги повідомлень. Це дозволяє впорядковувати трафік та оптимізувати доставку інформації до адресатів.

Повідомлення зберігаються в чергах — спеціальних буферах, які тимчасово утримують дані в разі перевантаження каналу або відсутності доступу до приймача. Після відновлення доступу або появи вільного ресурсу, повідомлення надсилаються далі.

У мережах Інтернету речей найчастіше використовуються такі протоколи:

— DDS— протокол прикладного рівня для систем із високими вимогами до часу реакції (реального часу);

— XMPP— застосовується для обміну повідомленнями в режимі, близькому до реального часу, з можливістю передачі статусів присутності;

— CoAP— розроблений спеціально для пристроїв з обмеженими обчислювальними та енергетичними ресурсами;

— MQTT— легкий і ефективний протокол для передачі даних на великі відстані з низьким споживанням ресурсів;

— STOMP— реалізує обмін повідомленнями з брокером на основі принципу "запит-відповідь";

— SOAP— призначений для передачі структурованих XML-повідомлень у розподілених обчислювальних середовищах.

Далі буде розглянуто найпоширеніші з цих протоколів.

DDS (Data Distribution Service) — це прикладний протокол, орієнтований на M2M-взаємодію в системах реального часу, який базується на архітектурі «видавець–підписник». Його основне завдання — організувати обмін даними між пристроями через спеціалізовану комунікаційну шину (див. рис. 1.12). DDS здатен надсилати мільйони повідомлень щосекунди, підтримуючи високу швидкість і синхронність.

На відміну від традиційних IT-мереж, де взаємодія здійснюється за класичним запитом до сервера, пристрої в реальному часі мають інші вимоги. Зокрема, затримки мають вимірюватись у мікросекундах, а з'єднання мають бути адаптивними, враховуючи складну маршрутизацію. Тому прості TCP-з’єднання з обмеженням на двоточкову комунікацію не підходять для таких сценаріїв.

DDS, натомість, пропонує гнучке керування параметрами якості обслуговування (QoS), підтримує мультитрансляцію, високу надійність та можливість масштабування. Його ключова перевага — можливість ефективного розповсюдження даних серед тисяч одержувачів із можливістю глибокого

фільтрування за критеріями. Для пристроїв із низькими ресурсами також передбачені спрощені версії DDS, що дозволяють використовувати протокол в обмежених умовах.

Звичайна топологія типу "зірка" не є ефективною для таких систем. DDS застосовує модель обміну даними через спільну інформаційну шину (DataBus), яка виступає мережевим аналогом бази даних, дозволяючи пристроям безпосередньо взаємодіяти між собою.

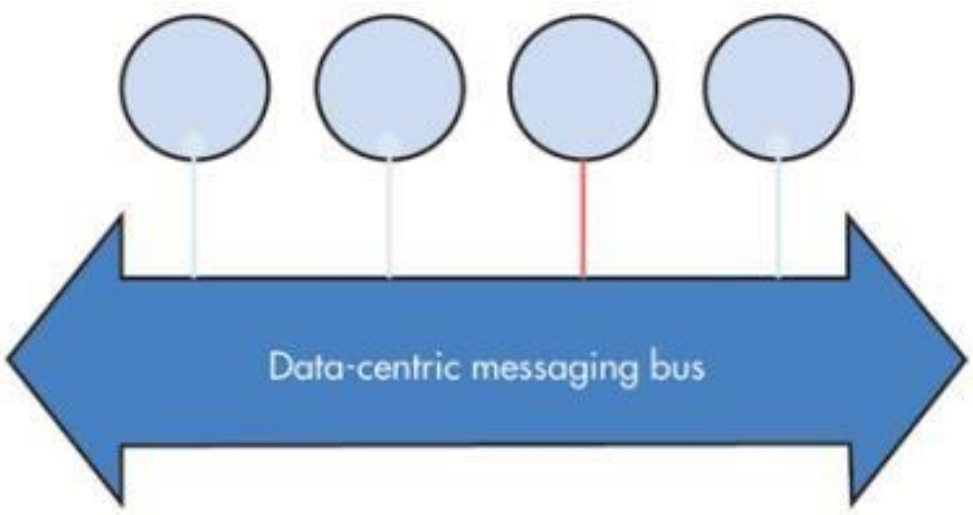


Рисунок 1.12 — Механізм взаємодії пристроїв з використанням DDS

MQTT (Message Queue Telemetry Transport) — це відкритий, легковаговий протокол, призначений для передачі даних у віддалених точках, де обмежені ресурси пристроїв та пропускна здатність каналу. Завдяки своїй компактності та ефективності, MQTT ідеально підходить для систем типу "машина-машина" (M2M) [9–10].

Для мереж, що складаються з бездротових сенсорних пристроїв, розроблено окрему версію цього протоколу — MQTT-SN (Sensor Networks), яка раніше була відома як MQTT-S. Цей варіант призначений для пристроїв, що не використовують TCP/IP.

Нижче наведено короткий опис роботи MQTT у спрощеному вигляді.

Пристрій-видавець надсилає дані (наприклад, з сенсорів вологості) на сервер-брокер, вказуючи відповідну тему — наприклад, "вологість", далі брокер перевіряє, які підписники оформили підписку на цю тему.Усі передплатники, що слідкують за темою "вологість", отримують повідомлення з відповідними даними.

Такий підхід дозволяє ефективно розповсюджувати інформацію серед багатьох клієнтів без потреби в прямому з’єднанні між відправником і кожним з одержувачів.

STOMP (Simple Text Oriented Messaging Protocol) — це легкий у використанні текстовий протокол для передачі повідомлень, який підтримує взаємодію між різними мовами програмування, операційними системами та брокерами повідомлень.

STOMP добре підходить для реалізації шаблону "видавець-підписник" (див. рис. 1.13) і забезпечує обмін повідомленнями з брокером через типові команди, такі як SEND (відправлення), SUBSCRIBE (підписка), UNSUBSCRIBE (відписка), BEGIN (початок транзакції), ABORT (переривання), ACK (підтвердження), NACK (непідтвердження) та DISCONNECT (від'єднання).

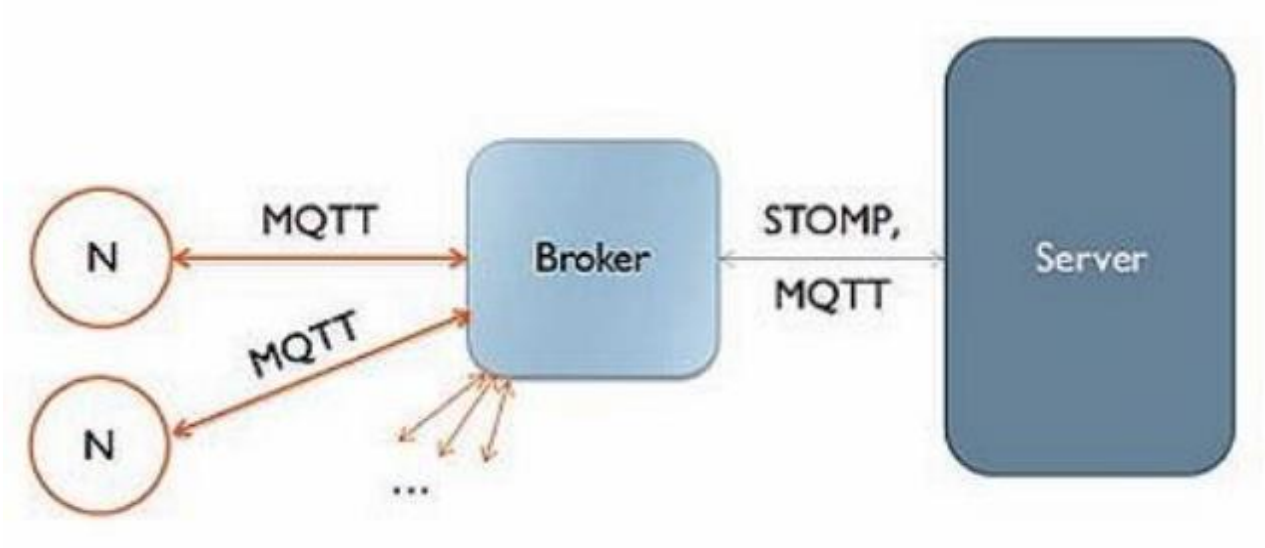


Рисунок 1.13 — Фрагмент мережевої структури з використанням MQTT і STOMP

Цей протокол нагадує HTTP і працює поверх TCP-з'єднання, дозволяючи клієнтам STOMP взаємодіяти з будь-якими брокерами повідомлень, які його підтримують. Така структура дає змогу легко інтегрувати компоненти, створені на різних мовах програмування, у єдину систему. Протокол підтримується великою кількістю бібліотек для різних платформ, що робить його універсальним інструментом для створення мультиплатформених рішень.

Кожен із розглянутих протоколів має своє призначення і використовується в залежності від специфіки завдань та особливостей середовища. Наприклад, DDS найкраще підходить для систем, що вимагають високої швидкості передачі великого обсягу даних, а також гнучкої архітектури та можливості масштабування. MQTT ідеально підходить для середовищ із великою кількістю пристроїв, де обмежена пропускна здатність мережі, а обсяг коду має бути мінімальним. Протокол STOMP, у свою чергу, доцільно використовувати для спрощення обміну повідомленнями та уніфікації комунікацій між різними мовами програмування і платформами.

Більш детальну інформацію про призначення і функціональні можливості кожного протоколу наведено у таблиці 1.2.

Таблиця 1.2—Протоколи: цільове використання та підтримувані операції

Протокол	Призначення	Операції
DDS	Длямереж,щопотребують розподіленого навантаження	Процедури отримання та відправки даних
MQTT	Для завантажених мереж з великою кількістю пристроїв та брокером	Процедуриобробки публікацій/передплати
STOMP	Для мереж, в яких є можливість застосовувати декілька комбінацій різних протоколів, що потребують простий протокол передачі повідомлень через брокера	Процедурипублікацій/передплати. Операції з транзакціями

У системах інтернету речей для організації обміну інформацією використовують різноманітні технології та протоколи, вибір яких залежить від особливостей середовища та конкретних вимог до побудови мережі.

Для реалізації локальних IoT-рішень, наприклад, у межах одного житлового або офісного приміщення, оптимальним варіантом є використання бездротової технології Wi-Fi HaLow, яка забезпечує розширене покриття при низькому енергоспоживанні, у поєднанні з протоколом MQTT, що є ефективним для передавання невеликих обсягів даних з великої кількості пристроїв.

У випадку побудови персональних мереж ближньої дії (PAN), доцільно застосовувати BLE або RFID— технології, призначені для комунікації між пристроями, що знаходяться на невеликих відстанях. Для обміну командами та даними в таких мережах доцільно використовувати MQTT або DDS, які підтримують масштабованість і високу швидкодію.

Протокол STOMP відіграє важливу роль у забезпеченні взаємодії між різнорідними системами, оскільки дозволяє уніфікувати обмін повідомленнями незалежно від внутрішніх механізмів передачі, що спрощує інтеграцію пристроїв з різними платформами.

1.6 Огляд існуючих рішень для віддаленого контролю та керування IOT-пристроями

The Thing System — це система моніторингу для екосистеми інтернету речей, розроблена групою американських інженерів разом із Денні Гудманом, автором книг з JavaScript і HTML.

Основною метою проекту є об'єднання різноманітних гаджетів та елементів інтернету речей, що використовуються в системах розумного будинку з централізованим управлінням. Пристрої різних виробників часто не можуть взаємодіяти один з одним і працюють не синхронізовано. Для вирішення цієї проблеми автори розробили програмне забезпечення, яке здатне

працювати з різними мережевими протоколами, гаджетами і клієнтськими додатками (див. рис. 1.14).

Список пристроїв, що підтримуються проектом, можна знайти на офіційному сайті проекту.

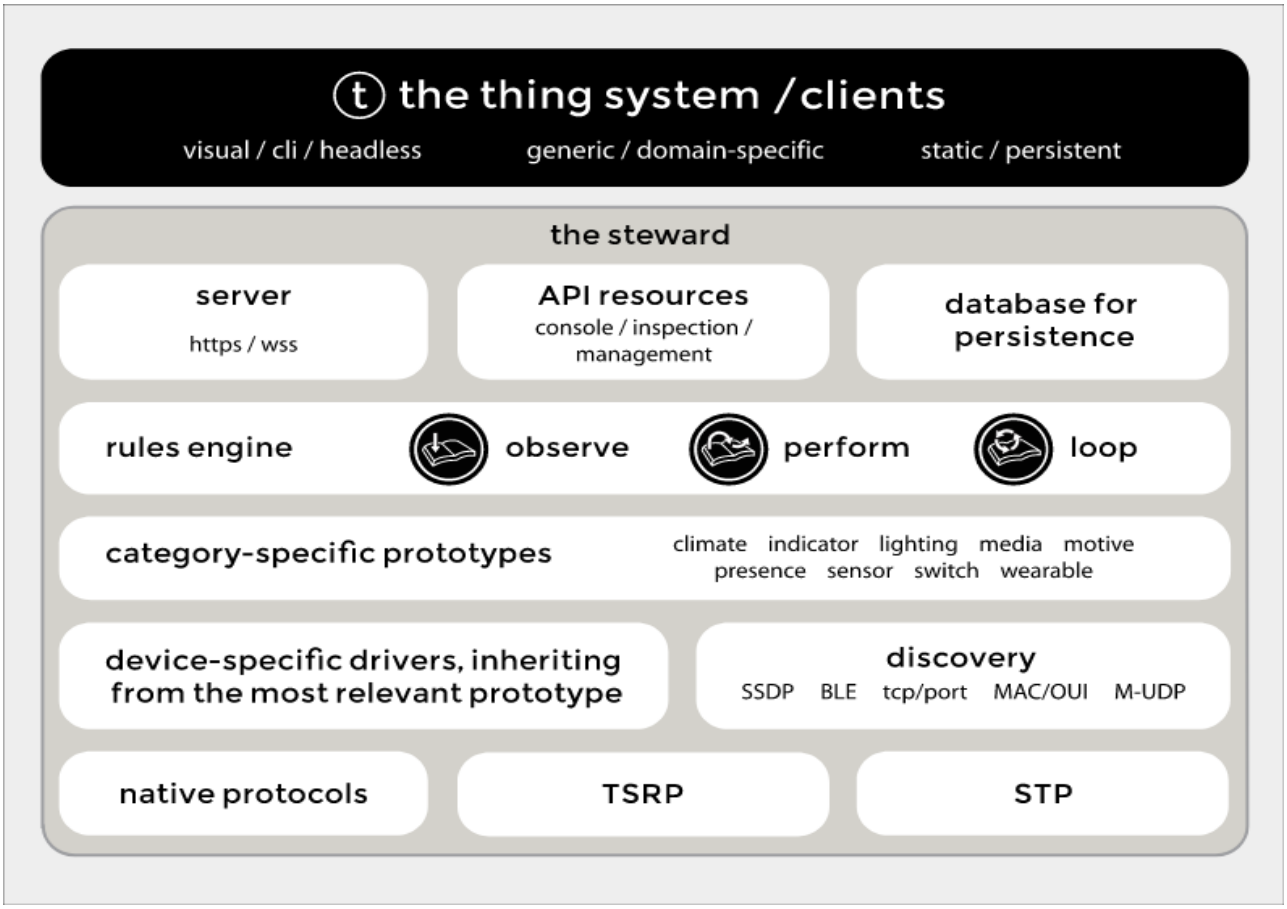


Рисунок 1.14 — Схема компонентів додатку Steward

Програмне забезпечення Steward, яке є основним ПЗ для системи The Thing System, розроблене на платформі Node.js, що забезпечує йому високу портативність та можливість простого розширення. Воно може працювати як на персональному комп'ютері, так і на віддаленому сервері або одноплатному комп'ютері, наприклад, Raspberry Pi.

Переваги цього ПЗ:

— компактний розмір;

— портативність і підтримка багатьох платформ;

— сумісність з різними пристроями в рамках мережі інтернету речей;

— легкість налаштування та інсталяції.

Недоліки ПЗ:

— система оптимізована для пристроїв розумного будинку, і при розширенні на великі мережі можуть виникнути проблеми з конфігурацією;

— обмежена продуктивність при масштабуванні, через те, що платформа побудована на Node.js;

— підтримка обмеженої кількості пристроїв.

Таким чином, система є оптимальною для використання в розумному будинку та простих локальних мережах. Проте при масштабуванні або обробці великих обсягів даних можуть виникнути проблеми з продуктивністю.

Domoticz — це система автоматизації розумного будинку з відкритим кодом, призначена для контролю різних пристроїв і обробки вхідних сигналів від різноманітних сенсорів. Вона підтримує велику кількість пристроїв від різних виробників. Система має відкритий код, основна частина якого написана на C++ та поширюється під ліцензією "GNU General Public License". Користувацький інтерфейс реалізований у вигляді адаптивного HTML5 веб-інтерфейсу, що автоматично підлаштовується під різні пристрої.

Domoticz працює на різних операційних системах, таких як Windows, Apple, Cubieboard, Raspberry Pi, а також на всіх Unix-подібних системах. Для мобільних платформ також доступні додатки, які дозволяють керувати системою віддалено. Користувачі можуть програмувати обробку подій і повідомлень, що надходять від пристроїв і датчиків. Логіку програмування можна налаштувати візуально за допомогою додатку Blockly (див. рис. 1.15).

Переваги системи:

— підтримка багатьох платформ;

— зручний веб-інтерфейс;

— сумісність з великою кількістю вже готових пристроїв;

— можливість розробки власних драйверів та скриптів для пристроїв;

— простота встановлення та налаштування;

— можливість масштабування для виконання різних завдань;

— гнучке налаштування поведінки пристроїв за допомогою програмування з використанням логічних блоків;

— відкритий код, що дозволяє самостійно змінювати і розширювати функціональність системи.

Недоліки системи:

— інтерфейс на базі HTML5 не підтримується старими браузерами;

— проблеми з продуктивністю при встановленні на платформи з обмеженим обсягом оперативної пам'яті.

Особливістю цієї системи є можливість керування через асинхронні запити, що містять дані у форматі JSON. Система також надає детальну документацію щодо всіх доступних запитів для управління.

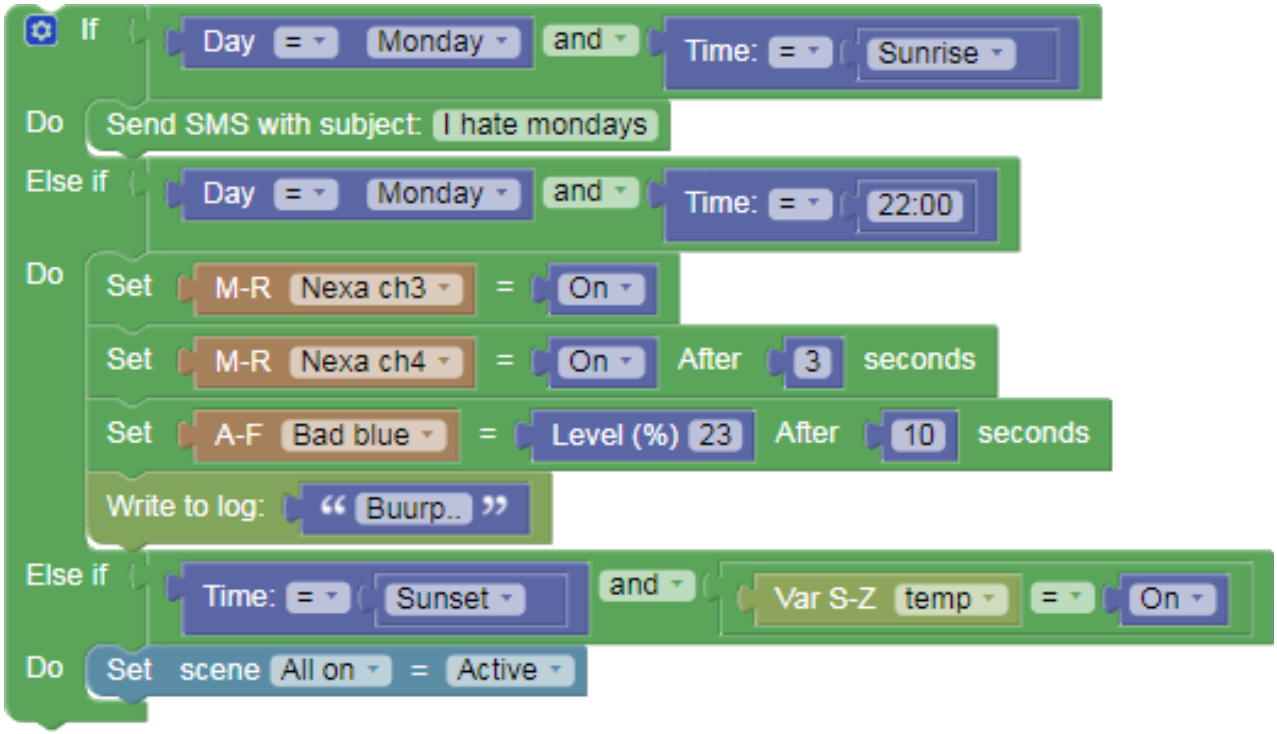


Рисунок 1.15 —Приклад створення програм за допомогою візуального інтерфейсу веб-додатку «Blockly»

Отже, можна зробити підсумок, а саме: проаналізовано архітектуру мережі Інтернету речей та основні протоколи, що використовуються для їх побудови. Для комунікації на малих відстанях найефективнішими є технології BLE та NFC, оскільки вони забезпечують оптимальне співвідношення між швидкістю передачі даних і енергоспоживанням. Для передачі даних на середніх відстанях найкращим вибором є Wi-Fi HaLow, оскільки, за рівного споживання енергії з Bluetooth, він забезпечує вищу швидкість передачі та більший радіус дії.

Також було розглянуто засоби моніторингу та управління елементами Інтернету речей. Серед них варто виділити Domoticz, який має відкриту кодову базу, підтримує велику кількість пристроїв і надає власний програмний інтерфейс для віддаленого управління системою.

2 АПАРАТНА ЧАСТИНА ТА НАЛАШТУВАННЯ ЗАСОБІВ ВІДДАЛЕНОГО КОНТРОЛЮ Й УПРАВЛІННЯ ІОТ

2.1 Характеристика платформи Raspberry Pi

Raspberry Pi Model B — одноплатний комп’ютер, який спочатку був розроблений для навчання основам комп’ютерних наук у школах. Завдяки розвитку мобільних процесорів ця платформа стала дуже популярною серед ентузіастів, які займаються розробкою додатків для Інтернету речей. Однією з головних переваг Raspberry Pi є її доступність, універсальність та відкритість, що робить платформу ідеальною для численних проектів, від навчання до промислових застосувань.

Основні переваги платформи:

— низька вартість, що робить комп’ютер доступним для широкого кола користувачів;

— універсальність, завдяки великій кількості інтерфейсів і підтримці різноманітних операційних систем;

— відкритість, що дозволяє налаштовувати систему під конкретні потреби користувача, включаючи програмування драйверів і скриптів для додаткових пристроїв.

Особливості платформи Raspberry Pi:

— операційна система: платформа може працювати на основі Unix-подібних систем, таких як Raspbian, а також на Windows 10 IoT.

— виведення відео: підтримка FullHD відео, що дозволяє використовувати плату для мультимедійних завдань.

— розмір плати: компактні розміри — 85,6 × 54 × 17 мм, що робить пристрій зручним для вбудовування в різні проекти.

— інтерфейси: підтримка різноманітних інтерфейсів, таких як UART, HDMI, USB 2.0, Ethernet, Wi-Fi 802.11n, Bluetooth 4.1, а також GPIO порти для підключення датчиків і інших пристроїв.

— живлення: Raspberry Pi можна живити через micro-USB (5V) або через універсальні піни. Однак для стабільної роботи при підключенні великої кількості периферійних пристроїв рекомендується використовувати адаптер живлення на 5V/2A.

Рисунок 2.1 —Вигляд платформи Raspberry Pi

Інтерфейси Raspberry Pi:

— UART — для серійного зв'язку;

— слот для карт пам'яті SD, MMC, SDIO — для зберігання операційної системи та даних;

— HDMI — для підключення монітора;

— 3.5 мм стерео — для підключення пристроїв відтворення звуку;

— композитний відеовихід — для старих телевізорів або моніторів;

— USB 2.0 — для підключення периферійних пристрої;.

— Ethernet 10/100Mb RJ45 — для підключення до мережі;

— Wi-Fi 802.11n і Bluetooth 4.1 — забезпечуються окремою мікросхемою Broadcom BCM43438, що підключається через інтерфейс Shield або за допомогою зовнішніх модулів.

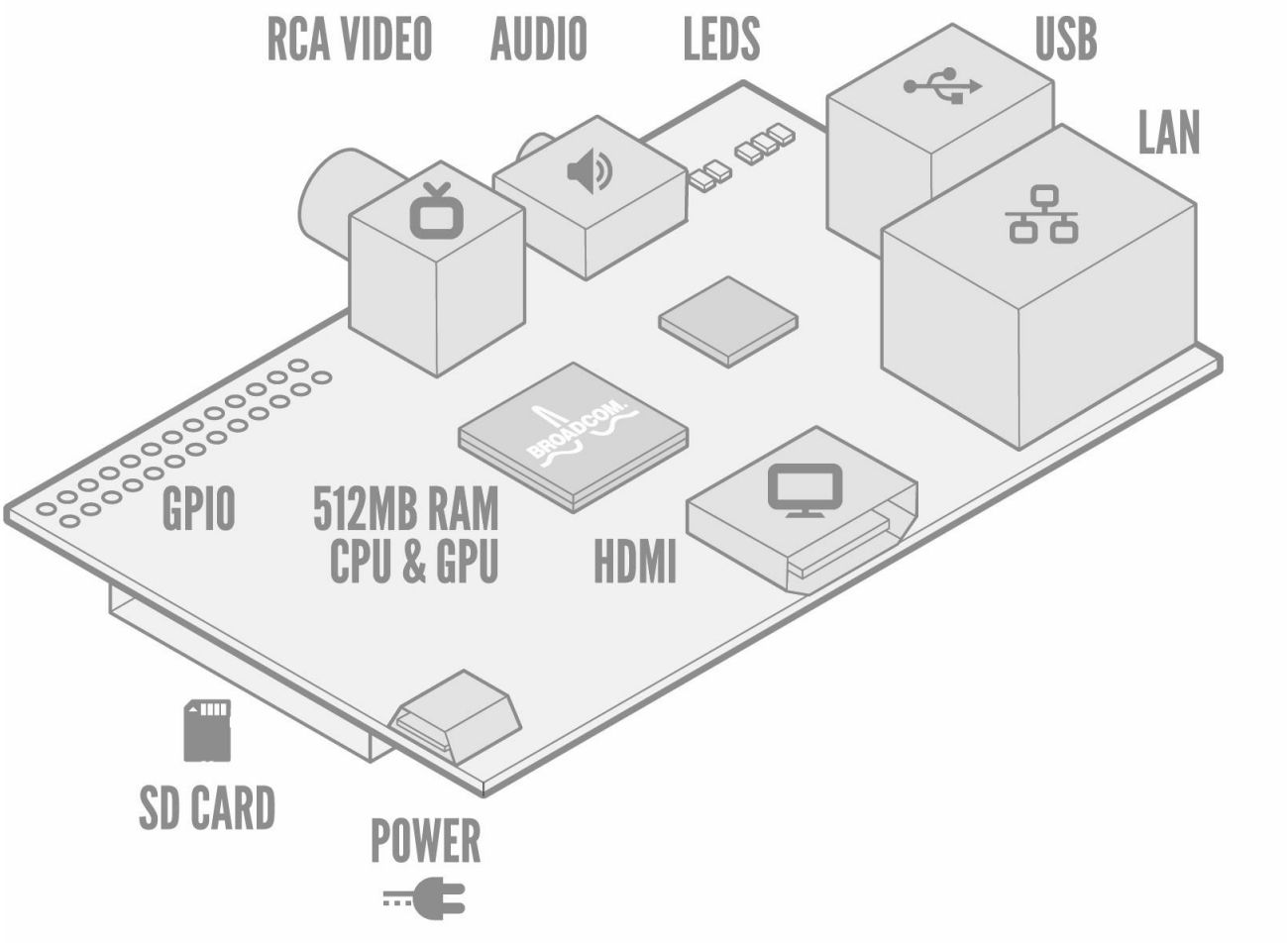


Рисунок 2.2 — Інтерфейсиплатформи Raspberry Pi

Процесор: Raspberry Pi Model B оснащений 32-розрядним процесором ARM Cortex-A7 з тактовою частотою 900 МГц. Він має кеш пам'яті L1 об'ємом 16 КБ і кеш L2 — 128 КБ. Кеш пам'яті другого рівня в основному використовується для обробки графіки.

Продуктивність: при роботі на частоті 700 МГц, продуктивність Raspberry Pi приблизно еквівалентна 0,041 GFLOPS, що порівнянно з процесором Pentium II 300 МГц. Графічний процесор здатний обробляти до 1

Gpixel/s або 1,5 Gtexel/s, що дозволяє обробляти відео і графіку з хорошою швидкістю.

Програмне забезпечення: Raspberry Pi підтримує різноманітні операційні системи, засновані на ядрі Linux, зокрема Raspbian. Крім того, є можливість встановлення Windows 10 IoT, що розширює можливості платформи. Для встановлення операційних систем можна використовувати інструмент NOOBS, який дозволяє записувати образи ОС на картку пам'яті для подальшої установки.

Ця плата є потужним інструментом для реалізації різноманітних проектів, від навчальних до промислових додатків, завдяки своїй універсальності, підтримці великої кількості інтерфейсів та низькій вартості.

2.2 Процес встановлення та конфігурування

Підготовка та процес встановлення карти пам'яті. Оскільки Raspberry Pi не має вбудованої постійної пам'яті, для його роботи необхідно попередньо підготувати карту пам'яті, на яку потрібно розпакувати образ обраної операційної системи [12-13]. Процес підготовки картки виглядає наступним чином:

— SD карта пам'яті (формат MMC/SDIO) об'ємом від 8 до 32 ГБ, з високим класом швидкості читання (не нижче класу 8);

— картридер для підключення картки до комп'ютера;

— програма Win32DiskImager для запису образу операційної системи на картку пам'яті;

— образ операційної системи, наприклад, Raspbian, який буде використовуватися на Raspberry Pi.

Після цього карту пам'яті необхідно підключити до комп'ютера через картридер і за допомогою програми Win32DiskImager записати образ операційної системи на карту.

Перше підключення Raspberry Pi та первинне налаштування. Для першого запуску та налаштування Raspberry Pi, крім самої плати та карти пам'яті з операційною системою, вам знадобляться:

– USB-клавіатура;

– монітор з HDMI-виходом або телевізор з RSA роз'ємом та відповідний кабель для підключення;

– блок живлення 5В, потужністю не менше 0.5А [13].

Після підключення всіх необхідних елементів та ввімкнення живлення, Raspberry Pi автоматично завантажить операційну систему (рис. 2.3).

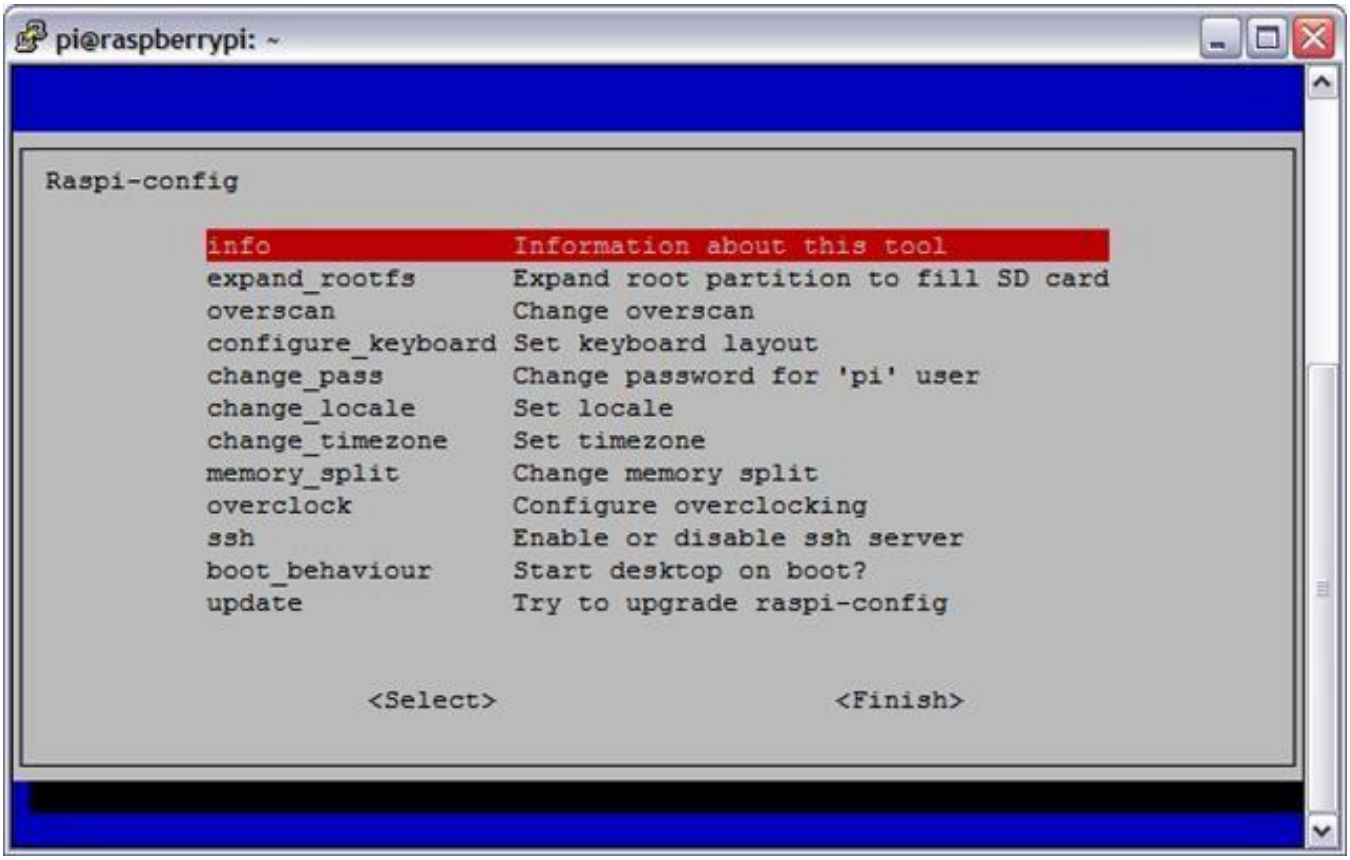


Рисунок 2.3 — Конфігурування платформи Raspberry Pi

Після першого запуску Raspberry Pi користувач потрапляє до вбудованого меню налаштування системи. Це меню дозволяє швидко адаптувати пристрій під власні потреби ще до початку роботи з графічним інтерфейсом або

встановлення додаткового програмного забезпечення. Нижче описані основні пункти цього меню:

— expand_rootfs — дозволяє автоматично розширити системний розділ на весь доступний об’єм карти пам’яті, що дає змогу використовувати її повністю;

— configure_keyboard — дає можливість вибрати відповідну розкладку клавіатури або драйвер, у разі якщо варіант за замовчуванням не підходить;

— change_pass — тут можна задати новий пароль для стандартного користувача з ім’ям "pi". Пароль вводиться двічі, при цьому він не відображається на екрані;

— change_locale — дозволяє змінити мову інтерфейсу, формат дати, часу та чисел відповідно до обраної локалі;

— change_timezone — встановлює часовий пояс. Raspberry Pi не має вбудованого годинника реального часу, тому для визначення часу використовує мережу Інтернет;

— memory_split — дає змогу розподілити оперативну пам’ять між центральним і графічним процесорами. Наприклад, для завдань без використання графіки можна зменшити обсяг пам’яті, виділений GPU;

— overclock — активує розгін процесора, підвищуючи його продуктивність. Ця опція вимагає обережності, оскільки може впливати на стабільність системи та нагрівання плати;

— ssh — вмикає або вимикає SSH-сервер, що дозволяє здійснювати віддалене підключення до пристрою через мережу;

— boot_behaviour — визначає, чи буде система запускатись з графічним інтерфейсом одразу після завантаження, чи лише з командним рядком.

Після внесення змін необхідно перезавантажити систему. Це можна зробити, натиснувши комбінацію клавіш Ctrl + F, після чого слід підтвердити вибір.

Після перезапуску рекомендується встановити пароль для адміністративного користувача "root", це виконується за допомогою команди в терміналі: sudo passwd root.

Після введення команди потрібно двічі ввести новий пароль,це дозволяє убезпечити доступ до системи.

Для перегляду поточних процесів, що виконуються в системі, можна скористатися утилітою top, ввівши відповідну команду в терміналі: top (див. рис. 2.4).

Цей інструмент надає інформацію про використання ресурсів, активні процеси, навантаження на процесор та пам’ять, що особливо корисно для моніторингу стану пристрою в реальному часі.

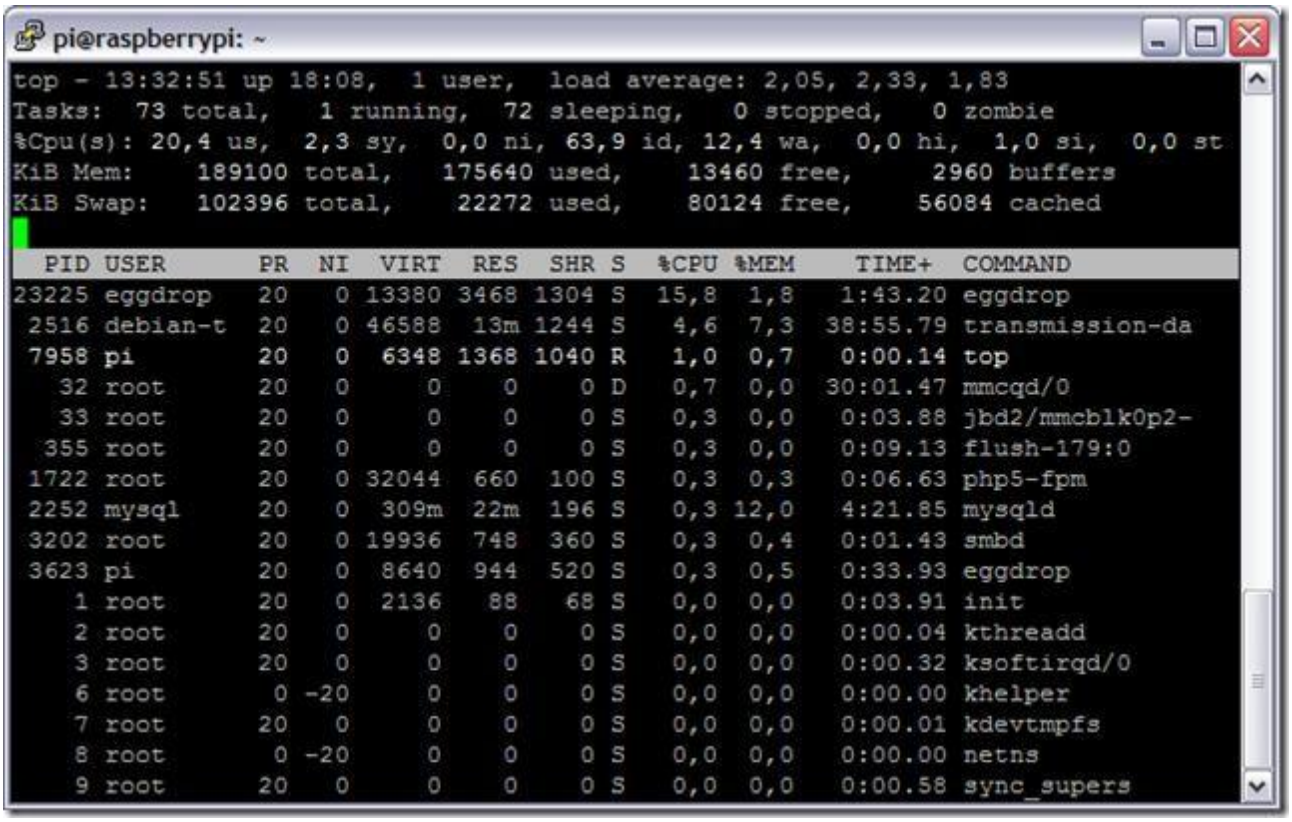


Рисунок 2.4 — Вікно після запуску команди top

2.3 Зовнішня пам'ять та модуль Wi-Fi

Для організації бездротового з’єднання з мережею на Raspberry Pi буде використано Wi-Fi адаптер Auspi Wireless Adapter, який підтримує стандарти бездротового зв’язку 802.11 b/g/n. Основними перевагами цього модуля є сумісність з USB-інтерфейсом, робота в поширених діапазонах частот, а також те, що він не потребує додаткового джерела живлення — живлення здійснюється безпосередньо через порт USB, що спрощує підключення та зменшує кількість елементів живлення в системі.

У якості зовнішнього накопичувача можна використовувати будь-який USB-диск або флеш-накопичувач. Головна умова — підтримка підключення через USB-інтерфейс, оскільки саме він використовується Raspberry Pi для взаємодії з периферією.

Після фізичного підключення адаптера та зовнішнього накопичувача необхідно переконатися, що операційна система їх розпізнала. Для цього в терміналі виконується команда:lsusb.

Ця команда виведе список усіх USB-пристроїв, що були успішно виявлені системою. Якщо адаптер Wi-Fi та зовнішній диск присутні у списку, можна переходити до налаштування мережі.

Наступним кроком є внесення конфігурації бездротового інтерфейсу wlan0 у системний файл налаштувань мережі. Для цього потрібно відкрити файл /etc/network/interfaces за допомогою текстового редактора nano з правами адміністратора: sudo nano /etc/network/interfaces.

У відкритий файл вносяться відповідні параметри з’єднання — зокрема, ім’я мережі (SSID), тип автентифікації та пароль. Це дозволить Raspberry Pi автоматично підключатися до вказаної Wi-Fi мережі при кожному запуску системи.

Після завершення налаштування параметрів мережі, необхідно перезапустити мережеві служби, щоб зміни набули чинності. Це виконується за допомогою команди: sudo service networking restart.

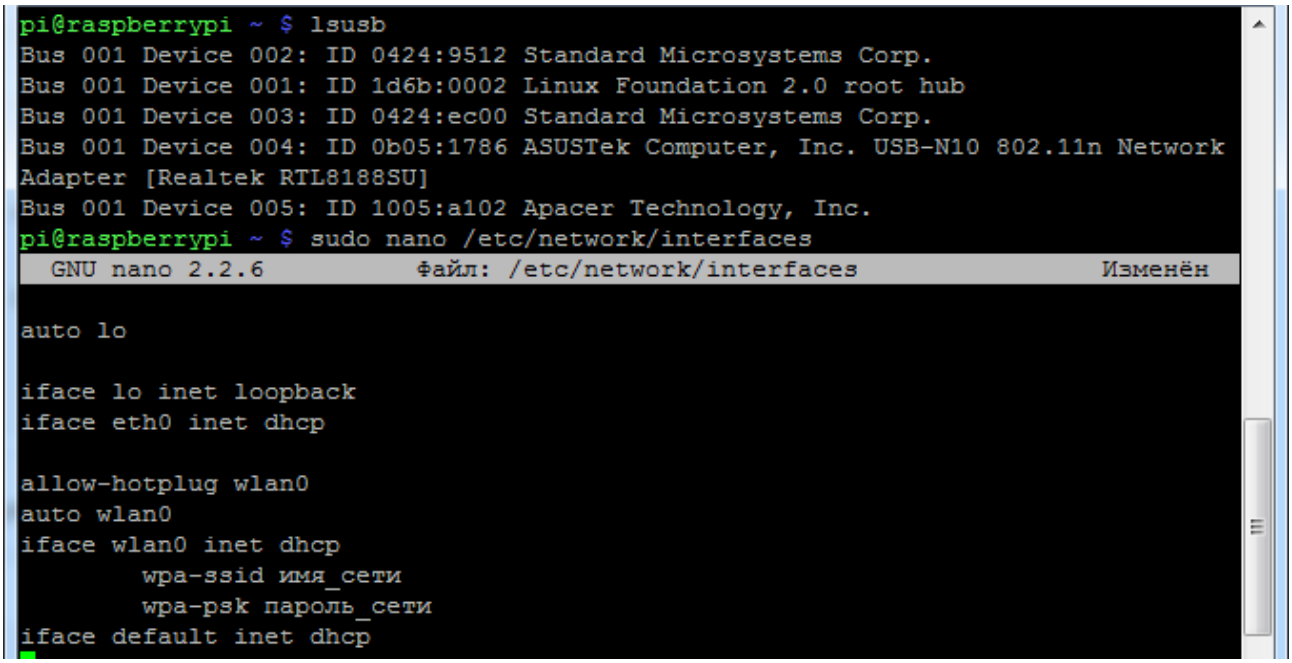


Рисунок 2.5 — Вікно після запуску команди lsusb

Далі рекомендується оновити системні пакети до актуального стану. Для цього в терміналі по черзі вводяться команди: sudo apt-get update та sudo apt-get upgrade

Це дозволяє оновити інформацію про доступні пакети та встановити всі останні оновлення, що підвищує стабільність і безпеку системи.

Підключення зовнішнього накопичувача.

На наступному етапі підключається зовнішній жорсткий диск. Для цього спочатку необхідно створити каталог, в який буде монтуватись диск. Найчастіше використовується папка /media, де створюється підкаталог:sudo mkdir /media/data.

Далі слід додати запис до конфігураційного файлу /etc/fstab, щоб диск автоматично монтувався при кожному запуску системи. Для цього потрібно дізнатися UUID (універсальний ідентифікатор) підключеного пристрою командою: sudo blkid.

Після отримання UUID відкривається файл fstab для редагування: sudo nano /etc/fstab.

У файл додається рядок приблизно такого вигляду: UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx /media/data auto defaults,nofail 0 0

Це забезпечує автоматичне підключення жорсткого диска до каталогу /media/data після кожного перезавантаження.

Щоб змонтувати диск вручну (не чекаючи перезавантаження), використовується команда: sudo mount /media/data.

Для кожного нового розділу або накопичувача ця процедура повторюється окремо.

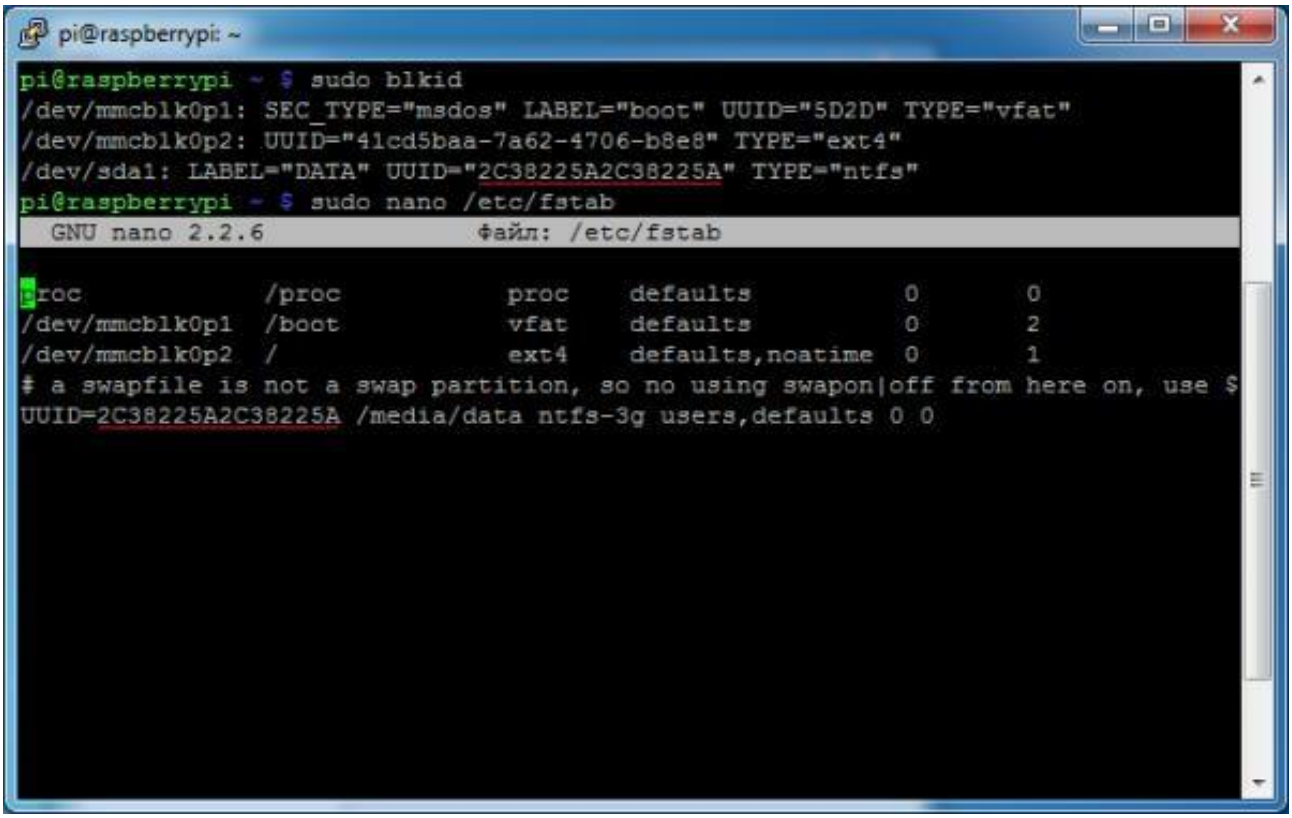


Рисунок 2.6 – Виявлення зовнішнього диска та редагування файлу /etc/fstab для автоматичного монтування

Після успішного підключення диска Raspberry Pi можна перезавантажити: sudo reboot.

Після перезавантаження клавіатура та монітор більше не потрібні, оскільки подальше керування пристроєм здійснюватиметься через SSH-з'єднання.

Для забезпечення стабільного доступу до пристрою через мережу бажано закріпити за ним постійну IP-адресу. Це робиться в налаштуваннях маршрутизатора, де необхідно прив’язати MAC-адресу Raspberry Pi до конкретної IP-адреси.

Також, у випадку використання динамічної IP-адреси від провайдера, доцільно налаштувати службу DDNS (Dynamic DNS), що дозволить звертатися до пристрою за фіксованим доменним ім’ям, навіть якщо IP-адреса змінюється.

На цьому базове налаштування плати завершене. Raspberry Pi готова до роботи в якості автономного сервера, керування яким буде здійснюватися через віддалене підключення по SSH.

2.4 Віддалене підключення через SSH та налаштування з’єднання

Для організації віддаленого підключення до Raspberry Pi з комп’ютера на операційній системі Windows можна скористатись протоколом SSH. Одним із найзручніших рішень є утиліта PuTTY, яка дозволяє встановити захищене з’єднання з пристроєм.

Для повноцінної роботи потрібно завантажити наступні компоненти:

— putty.exe — основна програма для створення SSH-з'єднання;

— puttygen.exe — генератор пари SSH-ключів (публічного та приватного);

— pageant.exe — агент, який зберігає ключі та виконує автентифікацію;

— pscp.exe — консольна утиліта для захищеного копіювання файлів;

— psftp.exe — інструмент для передавання файлів через FTP із шифруванням.

Для забезпечення коректної роботи в середовищі Windows необхідно внести відповідні зміни до системних змінних середовища. Це дозволяє запускати програми з будь-якого місця в командному рядку.

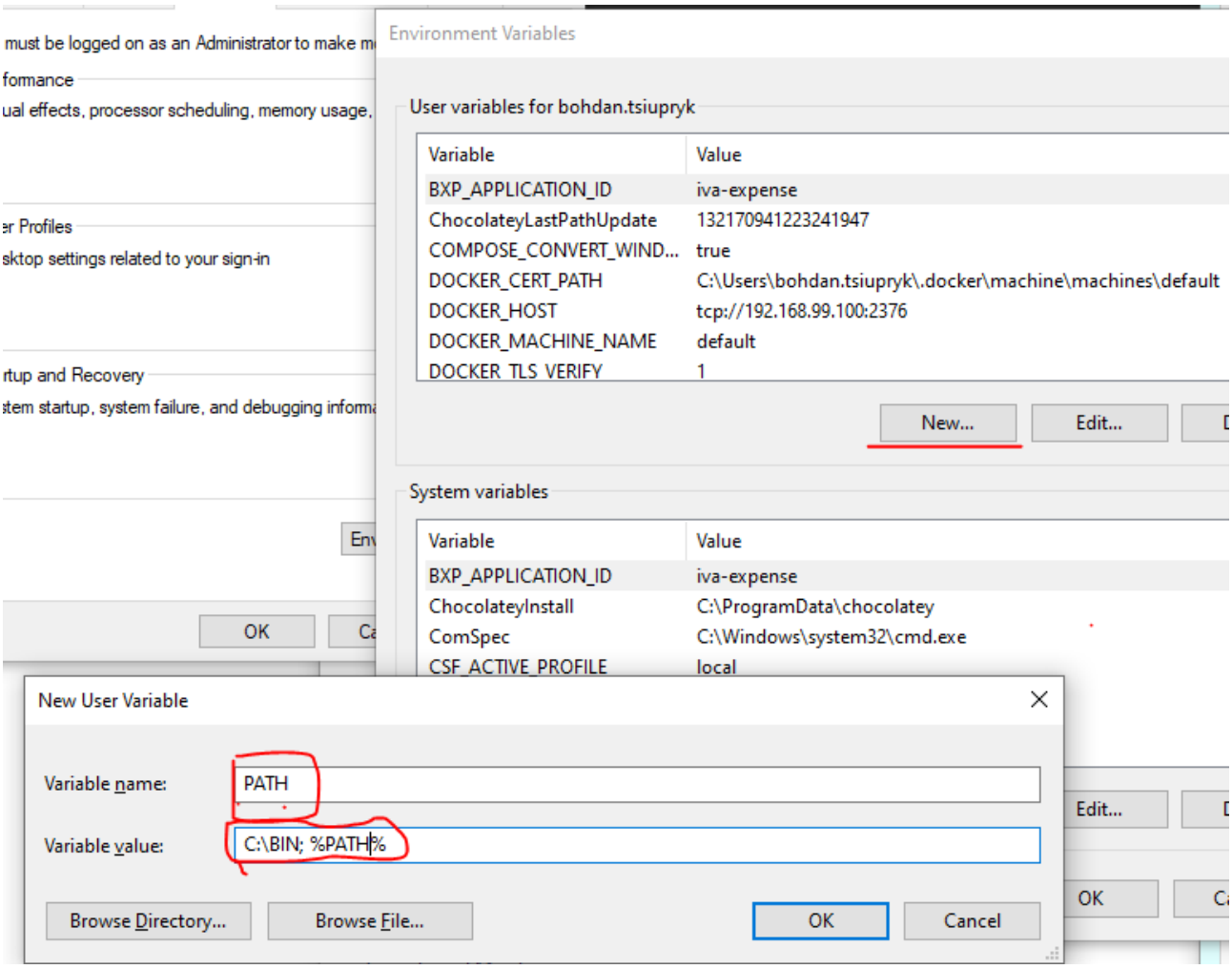


Рисунок 2.7 – Приклад додавання шляху до утиліт PuTTY у змінні середовища Windows

Щоб налаштувати змінну середовища:

— відкрити вікно властивостей системи;

— перейти до розділу додаткових параметрів та натиснути "Змінні середовища";

— у полі "Ім’я змінної" вказати: PATH

— у полі "Значення змінної" ввести шлях до каталогу, де знаходяться файли PuTTY, наприклад (див. рис. 2.7): C:\BIN;%PATH%, (де C:\BIN — це директорія, у якій збережені завантажені файли PuTTY).

Після завершення налаштувань середовища можна запускати PuTTY і підключатись до Raspberry Pi, вказуючи IP-адресу пристрою, порт (зазвичай 22) та тип з’єднання – SSH.

Далі розглянемо саму утиліту PuTTY, її конфігурацію та процес встановлення з’єднання з Raspberry Pi. На рисунку 2.8 показано вікно налаштувань сесії для підключення до пристрою.

Основні параметри, які потрібно вказати перед підключенням:

— Host Name (or IP address) — сюди вводиться IP-адреса або доменне ім’я пристрою, до якого планується з’єднання;

— Port — стандартний порт для SSH-з’єднань — 22, його потрібно вказати для успішного підключення;

— Connection type — для використання безпечного каналу з’єднання потрібно обрати пункт SSH;

— Saved Sessions — ця опція дозволяє зберегти налаштування поточної сесії, це зручно, якщо планується часте підключення до одного й того ж пристрою, для цього достатньо ввести назву сесії у відповідне поле та натиснути кнопку Save.

Після налаштування всіх параметрів можна натиснути Open, щоб встановити підключення до Raspberry Pi через SSH. У разі першого підключення програма запросить підтвердження довіри до ключа хоста, після чого відкриється консоль для введення логіна й пароля.

Після того, як усі необхідні дані будуть введені, потрібно натиснути кнопку "Open". Це відкриє термінал, в якому слід ввести логін і пароль, що були встановлені під час початкової конфігурації Raspberry Pi. Після введення цих даних встановиться з’єднання з пристроєм.

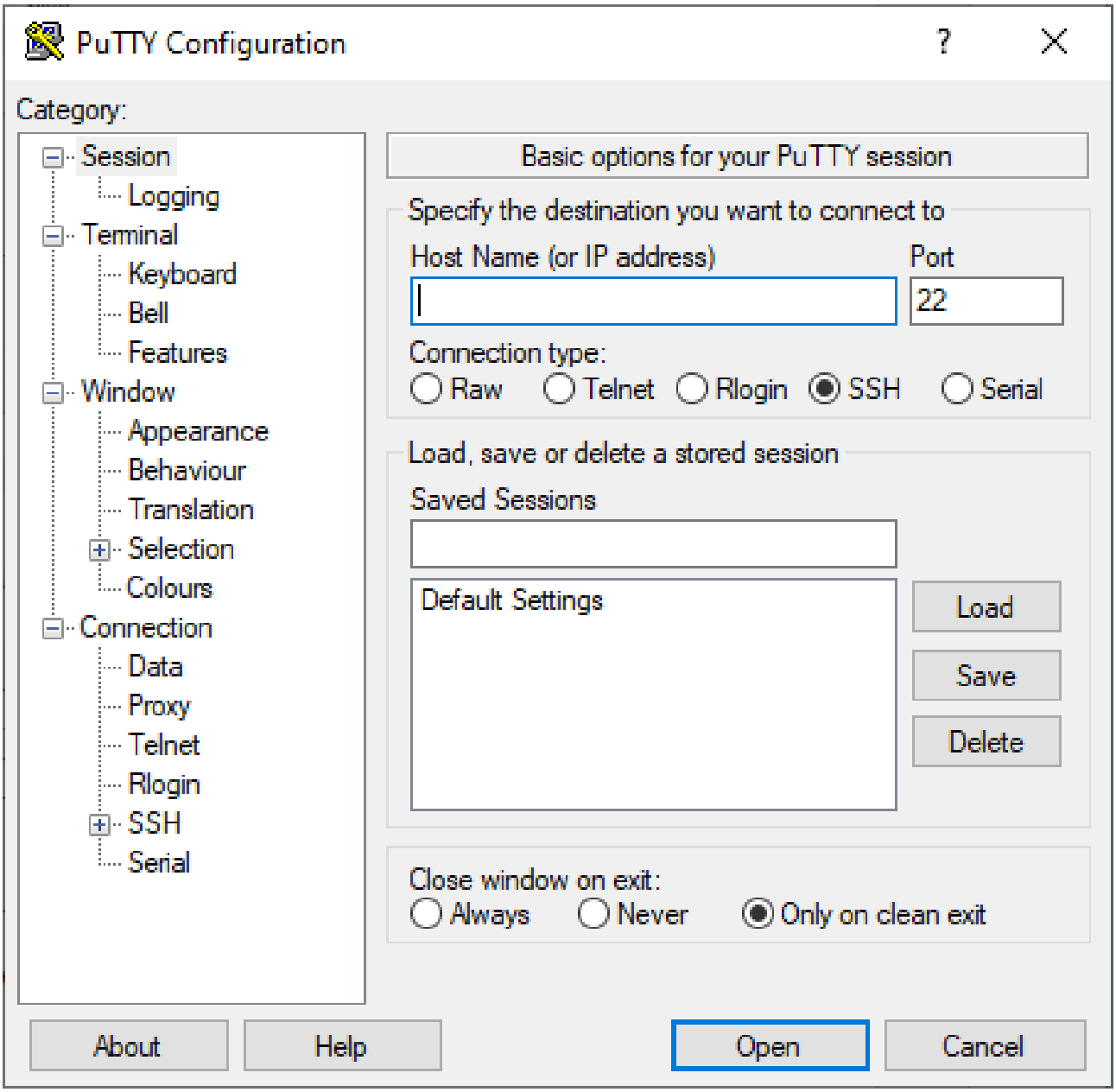


Рисунок 2.8 — Вікно програми PuTTY

2.5 Domoticz API як інструменту для керування пристроями

Щоб розпочати встановлення, потрібно перейти в домашній каталог за допомогою команди cd /home/, а потім завантажити та запустити установчий скрипт за допомогою команди: curl -L https://install.domoticz.com | bash.

Після завершення установки, відкрийте браузер і введіть адресу: IP-адреса-Raspberry-Pi:8080. Це приведе вас на стартову сторінку Domoticz (рис. 2.9). Оскільки ще немає підключених пристроїв, сторінка буде порожньою.

Для налаштування потрібно перейти до розділу Setup → Settings (рис. 2.10), де можна налаштувати такі параметри, як мова, тема оформлення, локація, захист сайту тощо. Щоб зберегти зміни, натисніть "Apply Settings".

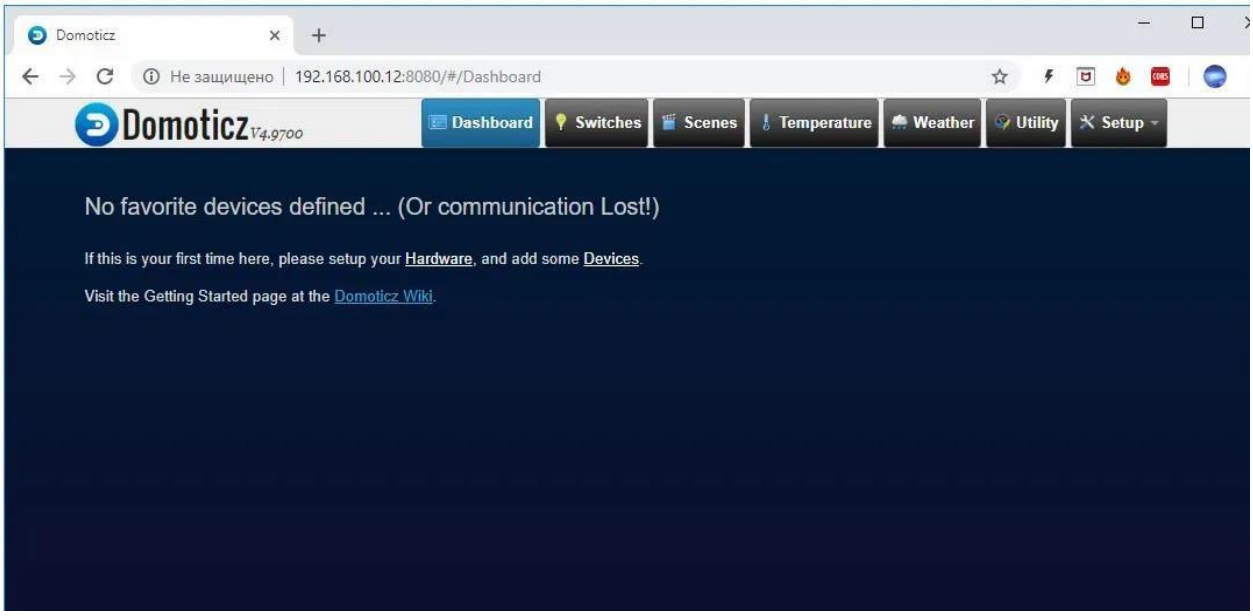


Рисунок 2.9 — Вікно домашньої сторінки Domoticz

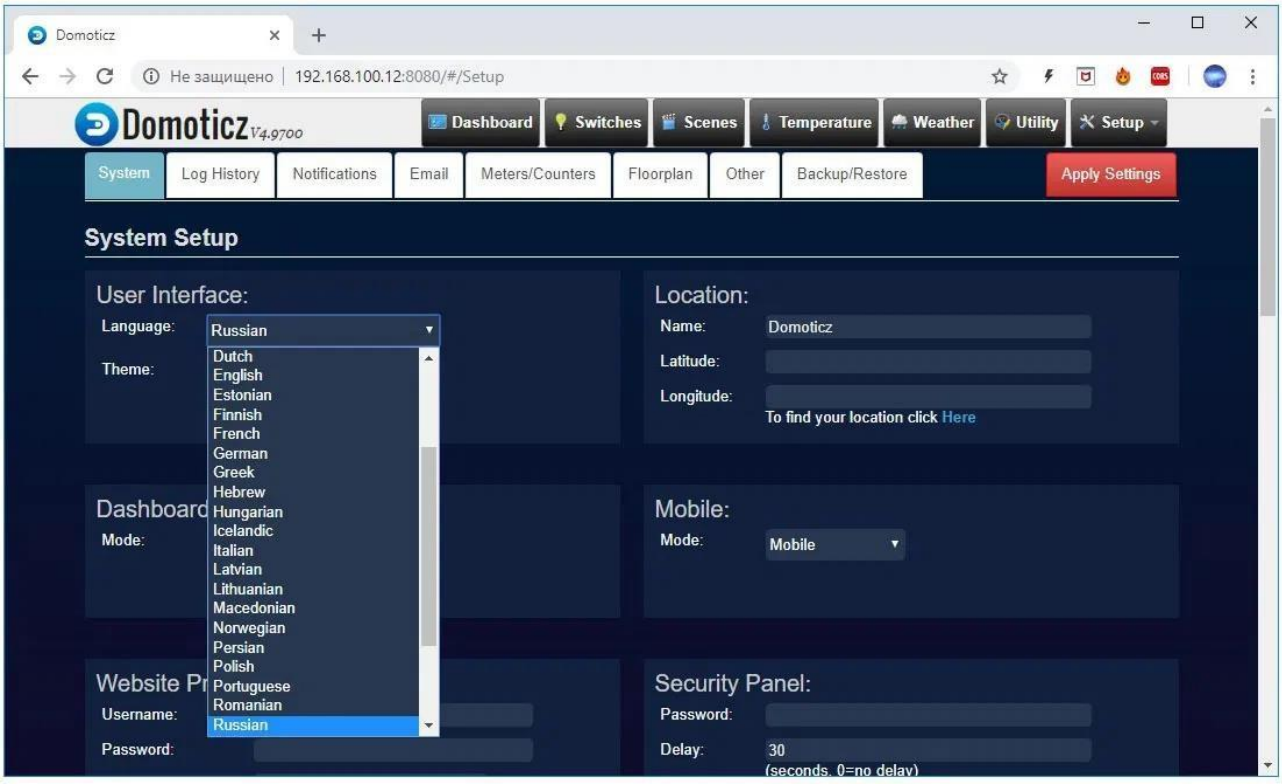


Рисунок 2.10 — Вікноналаштування Domoticz

Після цього можна завершити початкову настройку та додати кілька датчиків для тестування. Далі підключимо два датчики температури DS18B20 через інтерфейс 1-Wire, датчик температури і атмосферного тиску BMP180, а також датчик освітленості TSL2561 через інтерфейс I2C. Схему підключення шини GPIO для Raspberry Pi можна знайти в додатку Б.

Підключення датчиків до програми здійснюється через веб-інтерфейс. Для цього потрібно перейти в меню Setup, вибрати опцію Hardware, і в новому вікні буде представлений список підключених пристроїв, а також можливість додавати нові (рис. 2.11). Для додавання нового пристрою слід ввести його ім'я в полі Name, наприклад, "DS1820 #1", а в полі Type обрати інтерфейс "1-Wire (System)". Поле для шляху до патчу OWFS Path залишити порожнім. Інші налаштування можна залишити за замовчуванням. Після внесення всіх змін натискаємо кнопку Add, щоб зберегти пристрій. Аналогічно налаштовується другий датчик DS18B20[18].

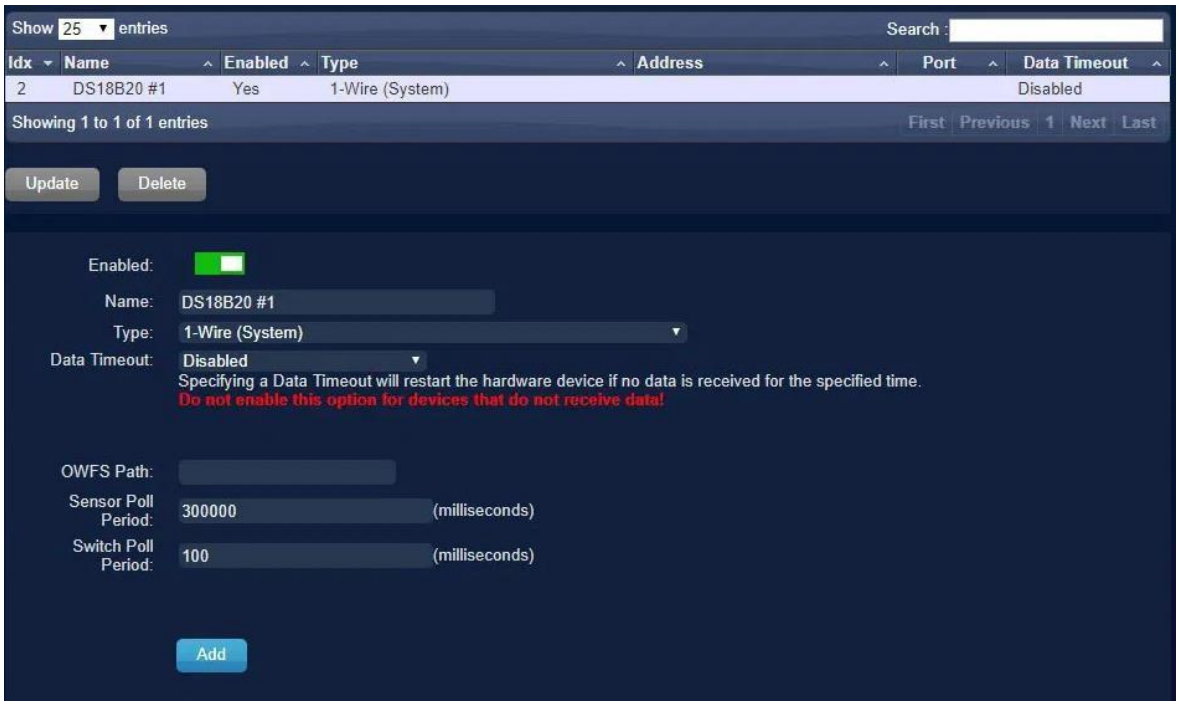


Рисунок 2.10 — Вікно добавлення нового пристрою та відображення підключених

Наступним етапом є налаштування датчика температури та атмосферного тиску BMP180. Для цього в полі Type вибираємо I2C sensors, а в полі SubType — I2C sensor BMP085/180 Temp+Baro (рис. 2.12). Те ж саме робиться і для налаштування датчика освітленості TSL2561, де в пункті SubType потрібно вибрати I2C sensor TSL2561 Illuminance[19].

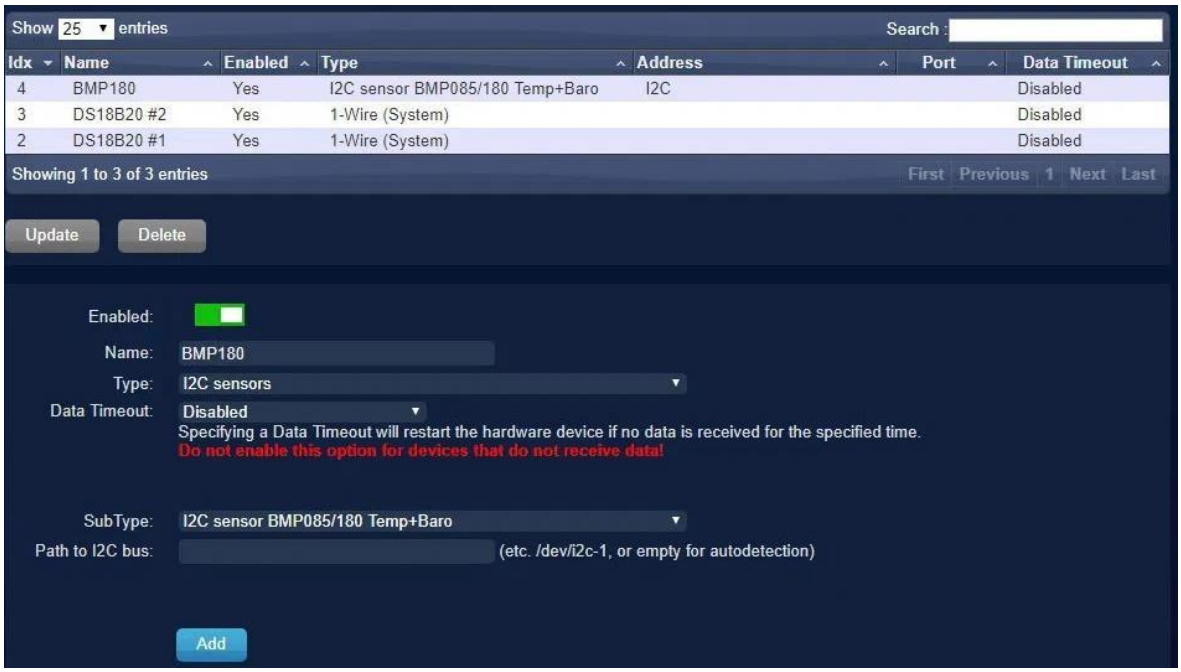


Рисунок 2.11 — Вікно налаштування давача BMP 180

Для активації підключених датчиків необхідно перейти в розділ Setup, обрати Setting та увімкнути потрібні датчики. Це робиться шляхом натискання на зелене коло поруч з кожним датчиком, що потребує активації, і присвоєння йому назви. Після активації його назва відобразиться в системі.

Всі датчики в системі згруповані за основною функціональністю. Наприклад, температурні датчики, як DS18B20 та BMP180, будуть відображатися в розділі Temperature, а датчик атмосферного тиску BMP180 — у вкладці Weather. Датчик освітленості TSL2561 потрапить у категорію Utility.

Групування датчиків за функціональністю є дуже корисним, особливо при великій кількості різних пристроїв. Такі датчики, які використовуються частіше за інші, можна додавати на головну панель — Dashboard (рис. 2.13).

Для цього достатньо натискати на іконку у вигляді зірки поруч з пристроєм на панелі. Коли зірка стане жовтою, цей пристрій з’явиться на головній панелі Dashboard.

Рисунок 2.13 — Вікно з давачами на головній панелі Dashboard

У системі Domoticz також є функція логування інформації, яка дозволяє відстежувати зміни показань датчиків протягом певного часу. Для того, щоб переглянути графік показань конкретного датчика, потрібно вибрати цей датчик і натискати на його іконку. Наприклад, на рис. 2.14 продемонстровано графік добової температури датчика DS18B20.

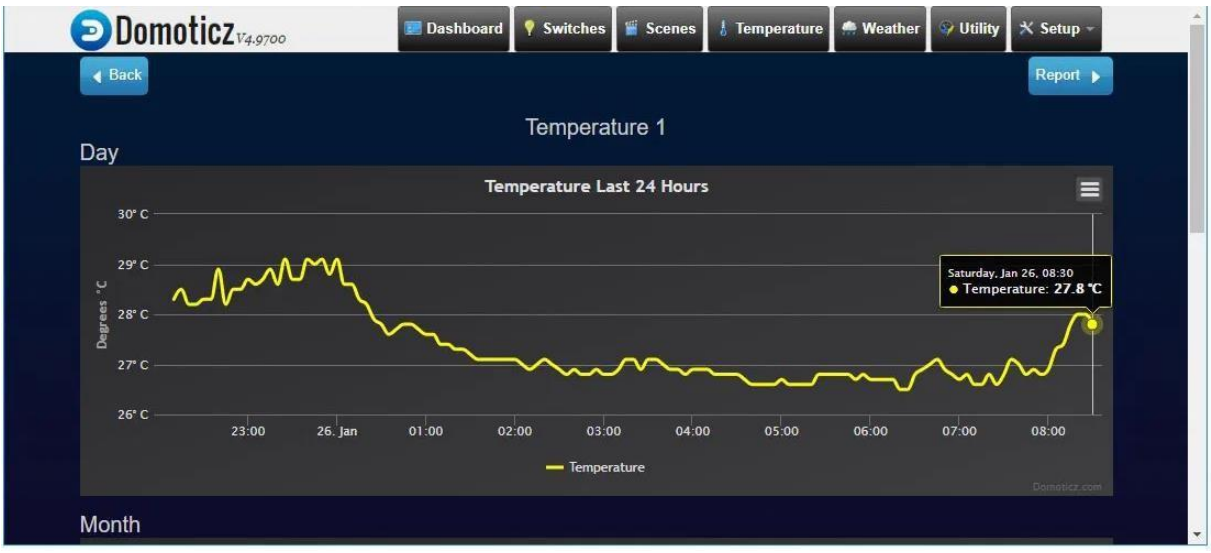


Рисунок 2.14— Вікно з графіком змін на здавачу температури DS18B20

2.5 Domoticz API як інструменту для керування пристроями

Domoticz надає можливість взаємодії з усіма підключеними комутаторами та датчиками через JSON. Це можна здійснювати як в інтерактивному режимі через браузер, так і програмно за допомогою скриптів. Система використовує наступний формат доступу:

— username:password — логін та пароль для доступу;

— domoticz-ip — IP-адреса системи Domoticz;

— port — порт для доступу (за замовчуванням 8080);

— api-call — запит для виклику API.

Цей інтерфейс дозволяє виконувати всі функції системи, від простих, як показ статусу пристрою або вмикання лампи, до складніших сценаріїв автоматизації. Наприклад, запит для вимкнення лампи виглядатиме так:

/json.htm?type=command&param=switchlight&idx=<device_id>&switchcmd=off

де idx — це ідентифікатор пристрою, а switchcmd — команда для увімкнення або вимкнення пристрою (параметри: "on" або "off").

Якщо запит виконано успішно, система поверне JSON-об’єкт з відповідним статусом та інформацією про пристрій, як показано на рис. 2.15.

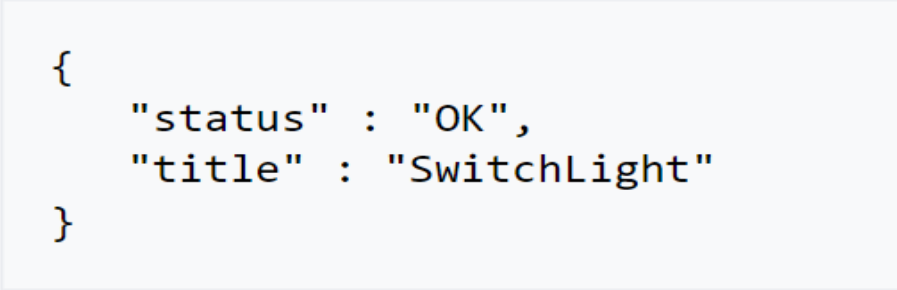


Рисунок 2.15— Вікно з підтвердженням успішного виконання

Отже, у підсумок модна сказати, що проведено аналіз одноплатного комп’ютера Raspberry Pi, який є оптимальним вибором для створення засобу моніторингу та управління елементами інтернету речей (IoT). Описано процес первинної конфігурації Raspberry Pi, включаючи налаштування операційної

системи та підключення необхідних пристроїв. Також розглянуто процес встановлення та налаштування системи Domoticz на Raspberry Pi. Окрім цього, було досліджено можливості користувацького програмованого інтерфейсу (API), що дозволяє інтегрувати та автоматизувати взаємодію з підключеними пристроями в системі.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СИСТЕМИ ВІДДАЛЕНОГО КОНТРОЛЮ ТА УПРАВЛІННЯ ІОТ

3.1 Архітектура програмного забезпечення

Для реалізації проекту була вибрана клієнт-серверна архітектура. Клієнт, у даному випадку, являє собою програму, яка встановлюється на одному пристрої або в локальній мережі, а сервер може бути розміщений на віддаленому комп’ютері або хостингу. Така організація дозволяє серверу працювати автономно, не залежачи від підключених клієнтів, а клієнтам – зручно підключатися до сервера без залежності від інших пристроїв.

Для розробки використано середовище Java версії 8 та фреймворк Spring Boot. Зберігання даних забезпечується за допомогою системи керування базами даних PostgreSQL. Для взаємодії з базою даних застосовується ORM-фреймворк Hibernate, що використовує об’єктно-орієнтований підхід для створення запитів. Hibernate дозволяє уникнути написання складних SQL-запитів, оскільки більшість необхідних запитів уже реалізовано фреймворком, і розробнику залишається лише викликати готові методи.

Веб-інтерфейс був розроблений із використанням шаблонізатора FreeMarker. Для розробки використовувалось середовище Intellij IDEA від JetBrains, яке надає безкоштовну версію з потужними плагінами та інструментами для розробки програм на мові Java.

Архітектура серверного додаткупобудована за шаблоном MVC (model-view-controller), який забезпечує слабку зв'язність між компонентами системи, що сприяє простоті підтримки та можливості заміни окремих модулів[23-25].

Розглянемо основні модулі:

— Model — відповідає за взаємодію з базою даних, обробку та валідацію даних;

— View — частина, що приймає вхідні дані та відправляє оброблені користувачеві;

— Controller — реалізує основну бізнес-логіку, обробляє операції та виконання дій з даними.

Модуль Model інкапсулює основну частину даних і функціональність їх обробки, при цьому він не залежить від процесів введення або виведення інформації.

View може містити кілька взаємопов'язаних областей, таких як таблиці та форми, що показують дані користувачу.

Controller відповідає за відстеження подій, що виникають через дії користувача. Контролер дозволяє структурувати код, об'єднуючи подібні функції в один клас. Наприклад, в типовому проекті MVC може бути контролер користувача, що містить методи для реєстрації, авторизації, редагування профілю, зміни пароля тощо.

Події, що реєструються в контролері, призводять до запитів, які передаються компонентам моделі або об'єктам, що відповідають за відображення даних. Відокремлення моделі від вигляду дозволяє використовувати різні компоненти для представлення інформації. Таким чином, зміни, внесені користувачем через контролер до моделі, автоматично відображаються в усіх відповідних візуальних компонентах.

Насамперед, слід описати архітектуриу клієнтського додатку. Клієнтський додаток побудований за монолітною архітектурою. Вибір такої архітектури обумовлений відсутністю необхідності підключення до бази даних або веб-інтерфейсу[25].

Основним завданням цього додатку є підтримка з'єднання з віддаленим сервером та забезпечення трансферу повідомлень. У зв'язку з цим, використання мікросервісної архітектури або шаблону MVC є зайвим, оскільки додаток виконує досить обмежену кількість функцій, які можна ефективно реалізувати в рамках єдиної монолітної структури.

Далі потрібно звернути увагу на мовупрограмування Java та фреймворк Spring Boot.

Java — це об’єктно-орієнтована мова програмування, випущена в 1995 році, з синтаксисом, подібним до C. Однією з основних характеристик Java є Java Virtual Machine (JVM), яка дозволяє розробляти програмне забезпечення, здатне працювати на різних платформах, що значно прискорює розробку і знижує її витрати. JVM використовує байт-код, який зазвичай генерується з вихідних кодів Java, але не завжди.

Основні причини вибору Java для розробки цього додатку:

— простий об’єктно-орієнтований синтаксис;

— безпечна та надійна робота з пам'яттю;

— незалежність від платформи та архітектури;

— висока продуктивність.

Spring Boot — це фреймворк, який спрощує розробку додатків на Java, надаючи реалізацію багатьох функцій, що в стандартному Java потребують окремої реалізації. Він дозволяє швидко створювати повноцінні, ефективні Spring-додатки, які можна запустити практично без додаткових налаштувань[22]. До Spring-платформи входять сторонні бібліотеки, що дають змогу швидко налаштувати додаток із мінімальними зусиллями. Більшість Spring Boot додатків потребують лише мінімальної конфігурації.

Основні можливості Spring Boot:

— створення повноцінних додатків, що не потребують додаткових сторонніх програм;

— вбудовані серлети, такі як Tomcat або Jetty;

— автоматична конфігурація, де це можливо;

— наявність можливостей моніторингу стану, розширеної конфігурації та метрик;

— конфігурація без необхідності генерувати код або писати XML-файли, на відміну від стандартної версії Spring.

Для розробки веб-додатків з використанням Spring Boot необхідно підключити залежність для роботи з веб-інтерфейсами. Завдяки інтеграції з

Maven, що автоматизує управління проектами, підключення необхідних бібліотек здійснюється просто, через кілька рядків коду в файлі pom.xml (лістинг 3.1).

Лістинг 3.1 — Управління проектами

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals'; import {UserProvider} from './context/userContext';
import '../node_modules/react-draft-wysiwyg/dist/react-draft-wysiwyg.css';
ReactDOM.render(
<React.StrictMode>
<UserProvider>
<App />
</UserProvider>
</React.StrictMode>,
document.getElementById('root'),
);
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals reportWebVitals();
```

PostgreSQL — це об’єктно-реляційна система управління базами даних, яка є відкритим програмним забезпеченням і має широку підтримку спільноти, оскільки не залежить від конкретних компаній. Ця база даних підтримує різноманітні типи даних і дозволяє користувачам створювати свої власні типи. Основні функціональні можливості PostgreSQL включають:

— забезпечення відповідності принципам ACID;

— підтримка запитів і підзапитів з операціями JOIN, UNION тощо;

— механізми послідовностей;

— реплікація даних;

— контроль за цілісністю даних;

— рекурсивні запити та загальні табличні вирази.

Hibernate, в свою чергу, є інструментом для перетворення Java класів у таблиці бази даних, а також забезпечує зручні інтерфейси для роботи з даними. Головною метою Hibernate є скорочення часу розробки завдяки вже реалізованим механізмам для роботи з базою даних. Hibernate автоматично управляє зв'язками між класами та відповідними таблицями і надає прості методи для їх взаємодії. Крім того, Hibernate використовує власну мову запитів HQL, яка є спрощеним аналогом SQL, орієнтованим на об'єкти, а не на таблиці.

Шаблонізатор Freemarkerє простим інструментом, який дозволяє розробникам створювати гнучкі та зручні веб-інтерфейси без потреби глибоких знань мов програмування, таких як JavaScript або TypeScript. Його основною перевагою є використання макросів, що дає змогу зменшити обсяг коду, оскільки можна повторно використовувати вже написані компоненти. Це сприяє ефективнішому та швидшому розробленню інтерфейсів.

3.2 Реалізація системи

У системі реалізовані такі моделі даних:

— Device — модель, що представляє пристрій, підключений до системи;

— DeviceDTO — модель, призначена для серіалізації та передачі даних пристрою без надмірної інформації;

— DeviceState — модель, що відображає стан пристрою;

— User — модель, яка містить дані користувача системи;

— UserRole — модель, що визначає ролі користувачів;

— Diagram — модель для графічного відображення даних

— Client — модель, що відповідає за клієнтську частину системи;

— Command — модель команди, яку можна передавати і зберігати в історії пристрою.

Керування цими моделями здійснюється через репозиторії Repository — інтерфейси фреймворку Hibernate. Імплементація цих інтерфейсів продемонстрована на лістинг 3.2. Такий підхід дозволяє уникнути дублювання

коду, адже в іншому випадку для кожної моделі довелося б створювати окремі методи для доступу до бази даних [24].

Лістинг 3.2 — Керування інтерфейсами

```
<Route path="/" element={<ProtectedLayout />}> <Route
path={HOME}
element={<ProtectedRoute><Courses/></ProtectedRoute>} />
<Route path="course">
<Route path=":courseId" element={
<ProtectedRoute><Course /></ProtectedRoute>
} />
</Route>
```

Інтерфейси JpaRepository та CrudRepository надають лише базові методи для взаємодії з базою даних, такі як пошук усіх об'єктів, пошук об'єктів за ідентифікатором, видалення об'єктів за ідентифікатором і оновлення інформації про об'єкт. Однак, якщо необхідна розширена функціональність, Java Persistence API (JPA) надає механізм для додавання спеціальних методів для управління базою даних, зокрема, через створення методів з певною назвою.

Наприклад, для пошуку пристроїв, що знаходяться в статусі OFF, базовий інтерфейс дозволяє здійснити пошук лише за ідентифікатором. Для більш складного запиту потрібно правильно сформулювати назву методу. Спочатку вказується тип повернення, оскільки кількість вимкнених пристроїв може бути більше одного, ми очікуємо масив або список пристроїв, отже, вказуємо тип List<Device>. Далі, вказуємо, що це пошуковий метод, використовуючи ключове слово find, потім додаємо All, щоб отримати всі записи, і додаємо ByStatus, де в параметри передаємо статус (лістинг 3.3).

Лістинг 3.3 – Програма пошукового методу

```
try {
const {id} = await db.collection('courses').add({
creatorName: user.name,
name: courseName,
description: courseDescription,
users: [],
});
await user.ref.collection('createdCourses').add({ id,
name: courseName,
description: courseDescription,
});
```

Реалізація бізнес-логіки додатку передбачає, що уся бізнес-логіка зосереджена в пакеті service. Основною метою додатку є збір даних з кількох серверів, на яких встановлено Domoticz, їх обробка та зберігання в єдиному місці, а також можливість збору статистики та керування пристроями.

Система Domoticz має API, яке дозволяє здійснювати запити в форматі JSON для отримання даних або відправки команд. Для надсилання запитів використовується вбудований інструмент Spring Boot — RestTemplate, що забезпечує зручне надсилання даних без необхідності явних перетворень. Клас, що реалізує відправку запитів, називається HttpUtil і є сервісом для здійснення запитів. Цей клас наслідує абстрактний клас ClosableHttpUtil (лістинг 3.4).

Лістинг 3.4 — Частина роботи Spring Boot

```
<Container>
<H2 sx={{margin: '12px 0'}}>My enrolled courses</H2> <Grid container spacing={2}>
{enrolledCourses && enrolledCourses.map( ({name, description, id, creatorName}) => (
<Grid item xs={6} md={4} lg={3} key={id}> <CourseCard
title={name}
description={description}
author={creatorName}
id={id}/>
</Grid>
))}
<Grid item xs={6} md={4} lg={3} key="join"> <AddCourseCard
```

Основні сервіси:

— HttpUtil— сервіс для здійснення запитів;

— DeviceService — обслуговування та управління пристроями;

— CommandService — створення та обробка команд;

— UserService — управління обліковими записами користувачів, реєстрація, авторизація тощо;

— DiagramService — обробка та конвертація діаграм у дані, придатні для відображення;

— MailService — надсилання електронних листів з інформацією про статистику та роботу системи;

— StatisticService — збір і аналіз статистичних даних про пристрої та функціонування програми;

— InitializationService — налаштування з'єднань із серверами та первісна конфігурація системи.

Користувацький інтерфейс створений за допомогою шаблонізатора Freemarker, основною перевагою якого є використання макросів. Макроси дозволяють повторно використовувати вже написаний код, змінюючи лише невеликі його частини.

Наприклад, для відображення діаграм код, що відповідає за відображення, був написаний один раз, а дані можуть динамічно підставлятися.

Для відображення діаграм застосовувався інструмент Google Chart — легкий скрипт, який є зручним у використанні. Для його інтеграції необхідно лише підключити бібліотеку, вказати тип діаграми та дані для її побудови (рис. 3.5).

Оформлення інтерфейсу здійснено за допомогою бібліотеки Bootstrap 4, яка має переваги у вигляді простоти освоєння та легкості.

Обробка даних, що надходять від користувачів через інтерфейс, виконується контролерами. Вони визначають шлях, за яким будуть отримані

дані, після чого ці дані обробляються і передаються сервісу для подальших операцій.

Контролер оголошується за допомогою анотацій @Controller або @RestController (рис. 3.6). Шлях, який обробляє контролер, задається за допомогою анотації @RequestMapping(path). Крім того, методи контролера можуть мати власні шляхи, які будуть доповнювати шлях контролера. Наприклад, якщо в контролері вказаний шлях "/device", а методу додана анотація @GetMapping("all"), то цей метод оброблятиме запит за шляхом "/device/all" з HTTP методом GET.

```
<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]
    ]);
```

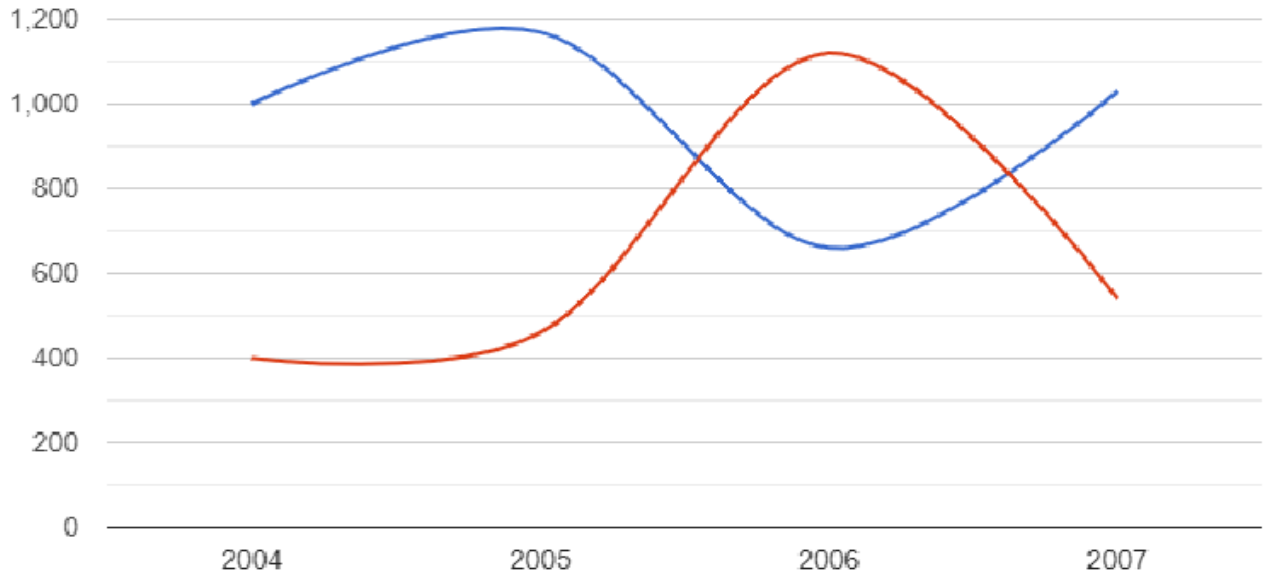


Рисунок 3.5— Приклад коду для створення діаграм та вигляд при відображенні

```java
@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);
    }
```

Рисунок 3.6— Приклад визначення шляху для обробки HTTP методу GET

3.3 Налаштування та оптимізація системи

Конфігурація доступу до бази даних здійснюється через конфігураційний файл db.properties (рис. 3.7). Після внесення змін в конфігурацію, для застосування змін необхідно перезапустити додаток, оскільки конфігурація завантажується при запуску програми.

Основні параметри конфігурації:

— datasource.url — URL-адреса бази даних;

— datasource.username — ім’я користувача для підключення до бази даних;

— datasource.password — пароль для підключення до бази даних.

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

Рисунок 3.7—Приклад налаштування доступу до бази даних

Налаштування клієнтської частини системи.

Для конфігурації необхідно внести відповідні параметри до конфігураційного файлу application.properties:

— remote-server.ip— IP-адреса віддаленого сервера, куди будуть надсилатися дані;

— remote-server.port— необов’язковий параметр, вказується тільки якщо віддалений сервер використовує порт, відмінний від 8080;

— remote-server.key— ключ безпеки, без якого запити будуть відхилені;

— remote-server.local-id— ідентифікатор клієнта;

— domoticz.login— логін для підключення до Domoticz;

— domoticz.password— пароль для підключення до Domoticz;

— domoticz.port— необов’язковий параметр, вказується лише якщо додаток використовує порт, відмінний від 8080.

Після налаштування, потрібно запустити клієнтську частину. Вона автоматично підключиться до Domoticz, знайде всі підключені пристрої, а після успішного підключення з’єднається з віддаленим сервером. Якщо сервер недоступний, програма увійде в режим очікування та періодично намагатиметься відновити підключення.

3.4 Тестування та демонстрація роботи

Інструкція з встановлення та запуску.

Вимоги до системи: для правильного функціонування системи необхідно, щоб на машині була встановлена JVM версії не нижче 8.

Клієнтський додаток: для запуску клієнтської частини системи потрібно скопіювати файл IoT-control-system-client.jar на машину, де працює додаток Domoticz. Після цього запустіть додаток за допомогою наступної команди: java -jar IoT-control-system-client.jar

Серверний додаток: для запуску серверної частини системи необхідно встановити базу даних з попередньо створеним користувачем. Після цього

скопіюйте файл IoT-control-system-server.jar та запустіть його командою: java -jar IoT-control-system-server.jar

Після цього відкриється вікно входу в систему (рис. 3.8).

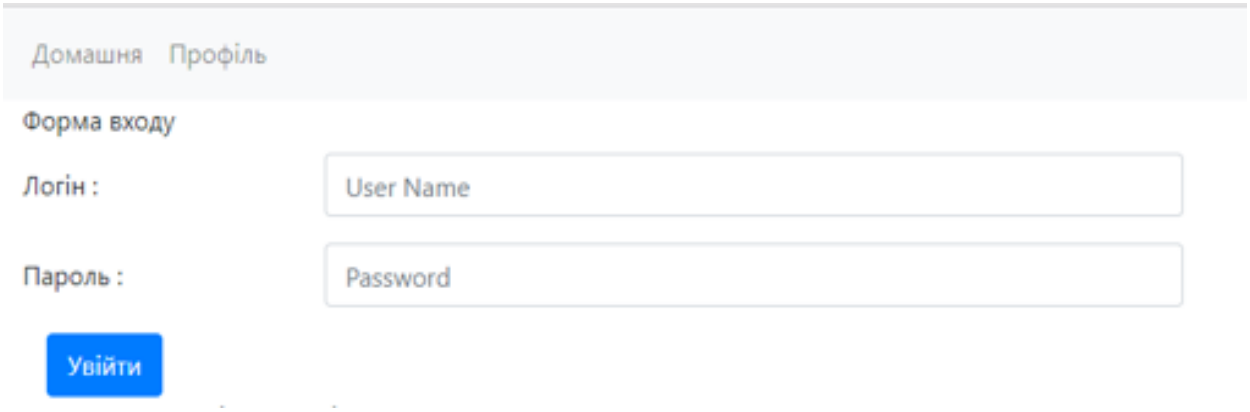


Рисунок 3.8— Вікно входження в систему

Демонстрація роботи: системою можна користуватися як на персональних комп'ютерах та ноутбуках, так і на мобільних пристроях, за умови підтримки браузера. Для того, щоб почати використання, необхідно перейти за адресою: https://<ip-адреса-сервера>:8080.

Перший запуск та налаштування користувачів.

Вхід у систему: під час першого запуску увійдіть у систему, використовуючи стандартні креденціали.

Логін: admin

Пароль: admin

Зміна пароля: після входу вам буде запропоновано змінити пароль на більш безпечний.

Додавання нових користувачів: у разі необхідності, для додавання нових користувачів або зміни прав існуючих, використовуйте панель управління користувачами (рис. 3.9).

На цій панелі можна редагувати інформацію про користувачів, налаштовувати їхні права доступу та додавати нових користувачів.

Рисунок 3.9— Вікно панелі адміністраторів

Підключення клієнтських додатків до системи

Створення клієнта: для підключення додатків-клієнтів до системи, спочатку необхідно створити новий клієнт у панелі створенняклієнтів (рис. 3.10).

Налаштування клієнта.

У процесі створення клієнта потрібно вказати:

— назву клієнта, яка буде використовуватись в системі як унікальний ідентифікатор;

— згенерувати ключ безпеки, який забезпечить захищене підключення клієнта до системи.

Після цього згенерований ключ безпеки можна використовувати для встановлення з'єднання з системою.

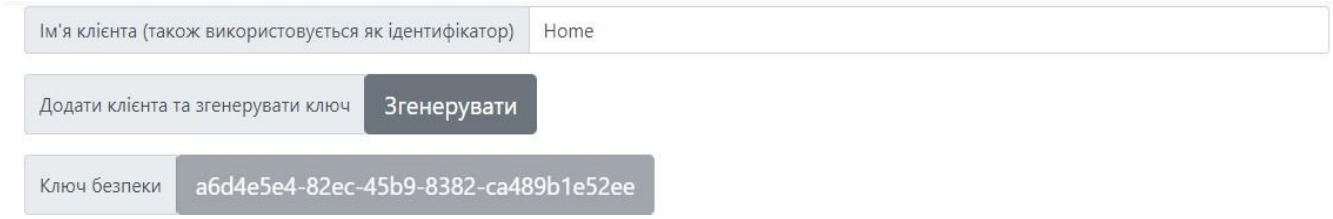


Рисунок 3.10 — Вікно панелі додавання клієнтів

Після додавання клієнта, необхідно виконати його конфігурацію та запустити. Під час налаштування потрібно вказати ідентифікатор та ключ, які були отримані під час створення клієнта.

Після успішного підключення клієнт відображатиметься в списку з статусом "активний" (рис. 3.11). Якщо підключення не відбулося, клієнт буде позначений як "новий". Якщо підключення було, але через певні обставини перервалося або пристрій вимкнений, клієнт змінить статус на "не активний".

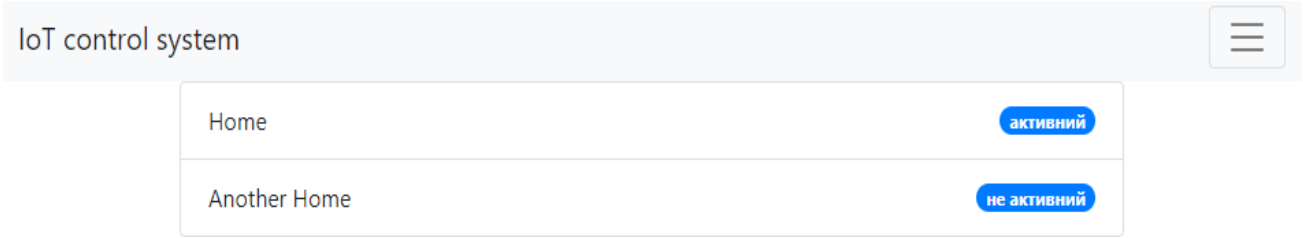


Рисунок 3.11— Вікно списку клієнтів

Щоб переглянути деталі про клієнта, слід натиснути на його запис. Це відкриє вікно з інформацією, такою як ім’я пристрою, IP-адреса, час роботи, кількість підключених пристроїв та можливість перегляду інформації про кожен з них окремо, а також інші важливі дані.

Для перегляду пристроїв, потрібно перейти на вкладку "Домашня", де будуть відображені всі пристрої, підключені на даний момент (рис. 3.12). Їх можна відсортувати за клієнтом або за типом пристрою, таким як датчик температури, лампа тощо.

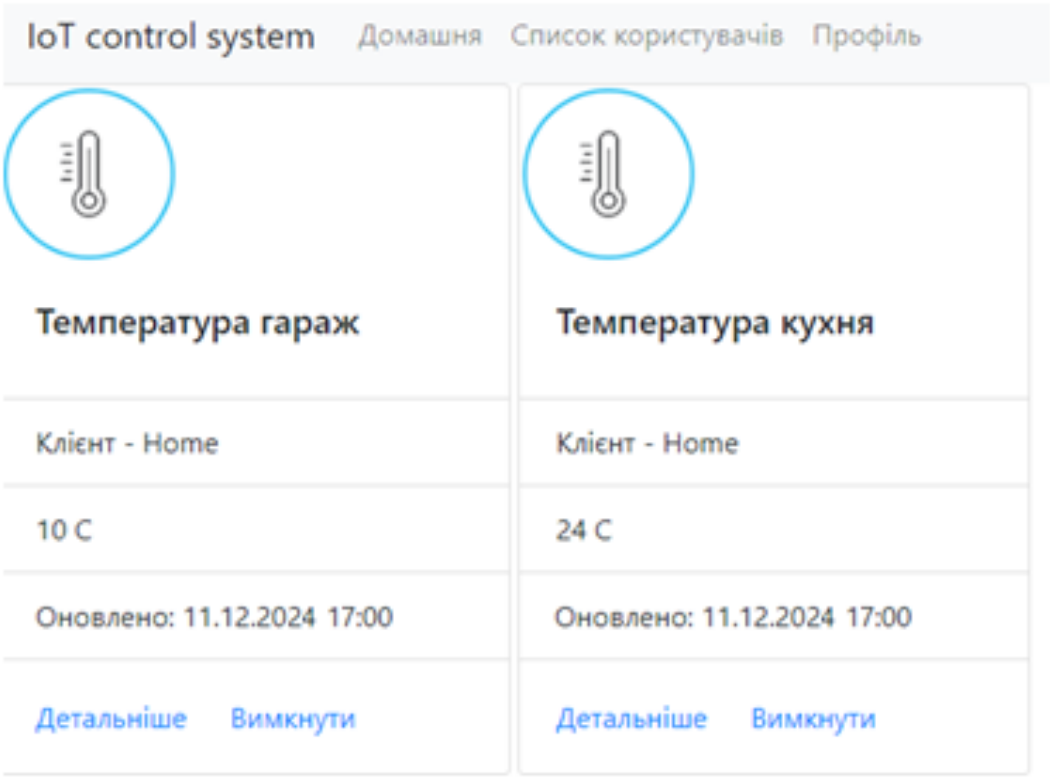
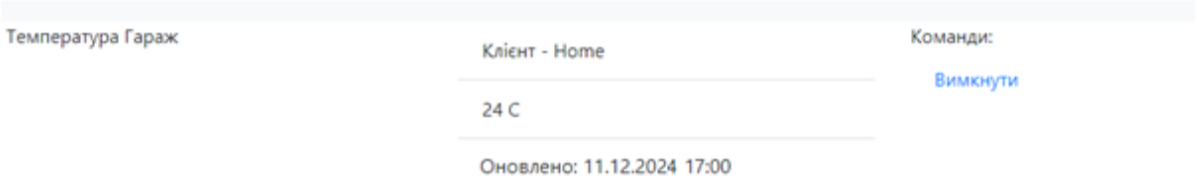


Рисунок 3.12— Вікно показників давача температури

З домашньої сторінки також можна перейти до детального перегляду пристроїв. Якщо пристрій вимкнено, він не відображатиметься на домашній сторінці, і для його увімкнення необхідно перейти до налаштувань клієнта.

Коли ви натискаєте на посилання "Детальніше" біля пристрою, відкривається вікно з детальною інформацією про нього, такою як його назва, показники, доступні команди для відправлення, а також, якщо це, наприклад, датчик температури, графік змін температури за останню добу (рис. 3.13).



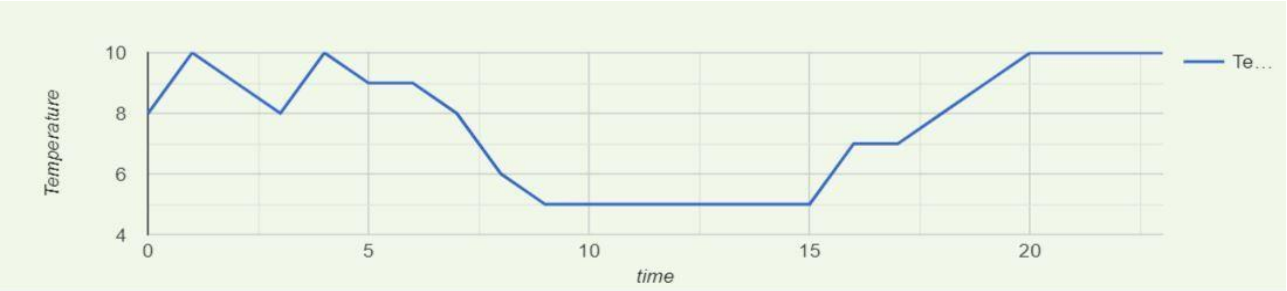


Рисунок 3.13— Вікно деталізації показників давача температури

Отже, можемо підбити підсумки, що зроблено: опис інструментів та мови програмування Java з використанням фреймворку SpringBoot, які були застосовані для розробки системи. Вона функціонує поверх платформи Domoticz, розширюючи її функціональні можливості. Використана модульна архітектура, що забезпечує простоту підключення додаткових серверів-клієнтів до системи без необхідності змінювати основний код. Описано процес налаштування клієнтів, додавання нових пристроїв, а також їх конфігурацію та параметри.

ВИСНОВКИ

У рамках виконаної роботи було розроблено систему віддаленого моніторингу та управління пристроями Інтернету речей (IoT), яка забезпечує гнучкість, масштабованість та надійність у процесах збору, обробки та передачі даних. Розглянуті в роботі підходи та технології забезпечують ефективний обмін даними між пристроями, а також можливість інтеграції з різними інтерфейсами для управління та моніторингу.

Дослідження сучасних підходів до побудови IoT-систем і застосовані методи дозволили реалізувати систему, що відповідає вимогам безпеки, ефективності та зручності у використанні. Зокрема, використання одноплатних комп’ютерів як основи апаратної частини системи продемонструвало свою ефективність для досягнення необхідної продуктивності та енергоефективності.

Система була протестована в умовах, наближених до реального використання, що дозволило оцінити її ефективність і потенціал для подальшої адаптації у різних сферах, таких як агропромисловість, енергетика, медицина та розумний будинок.

На основі проведеного дослідження можна зробити висновок, що запропонована система має великий потенціал для подальшого розвитку та впровадження у різноманітні галузі, де необхідний ефективний віддалений контроль та управління IoT-пристроями.

Зроблені висновки та отримані результати можуть стати основою для майбутніх наукових досліджень і розробок у сфері IoT, а також для впровадження нових технологій в сфері автоматизації та управління розподіленими системами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інтернет речей у наукових дослідженнях. 2017. URL: https://ieeexplore.ieee.org/document/7777630 (дата звернення: 01.09.2024)

2. Огляд та порівняльний аналіз технологій LPWAN мереж. 2016. URL: https://www.researchgate.net/publication/311797128 (дата звернення: 01.10.2024)

3. Z-Wave: Технічні основи. 2017. URL: https://www.z-wave.com/technology/ (дата звернення: 01.10.2024)

4. Технологія NFC: принципи роботи та переваги. 2016. URL: https://www.nfc-forum.org/what-is-nfc/ (дата звернення: 01.10.2024)

5. RFID технології та магнітні мітки. 2015. URL: https://www.rfidjournal.com/what-is-rfid (дата звернення: 01.10.2024)

6. The Basics of Bluetooth Low Energy (BLE). 2010. URL: https://www.novelbits.io/basics-bluetooth-low-energy/ (дата звернення: 01.10.2024)

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: 01.10.2024)

8. LoRa Mesh Networking with Simple Arduino-Based Modules. 2018. URL: https://github.com/nootropicdesign/lora-mesh (дата звернення: 15.10.2024).

9. BryanBoydetal.BuildingReal-timeMobileSolutionswithMQTT and IBM MessageSight. IBM Redbooks, 2014

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

11. RaspberryPiDocumentation.Officialresource.URL: https://www.raspberrypi.org/documentation/(Last accessed: 14.10.2024).

12. THE OFFICIAL Raspberry Pi Beginner’s Guide. URL: https://www.raspberrypi.org/magpi-issues/Beginners_Guide_v1.pdf(Last accessed: 14.10.2024).

13. Questions regarding armhf port for Raspberry Pi. URL: https://lists.debian.org/debian-arm/2012/03/msg00002.html (Last accessed: 14.10.2024).

14. PuTTYDocumentationPage.URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/docs.html (Last accessed: 14.10.2024).

15. Domoticz. Частина Друга. webhomepi. URL: https://whp.home.blog/2024/02/02/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.OpenSourceHomeAutomationSystem.URL: https://www.domoticz.com/DomoticzManual.pdf.

17. Domoticz.OfficialGitHubPage.URL: https://github.com/domoticz/domoticz

18. DS18B20.ProgrammableResolution 1-WireDigital Thermometer. URL: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

19. BMP180Datasheet(HTML)-TexasInstruments.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. TheJavaLanguageEnvironment.1997SunMicrosystems,Inc. URL: https://www.oracle.com/technetwork/java/intro-141325.html.

22. RobHarrop,ClarenceHo.ProSpring3.UnitedKingdom.2012.

23. JoshuaBloch.EffectiveJava.Addison-WesleyProfessional.2018.

24. AlexBerson.Client/ServerArchitecture(McGraw-HillComputer.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф.
_____О. Д. Азаров
"_21_" березня_____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврськоїкваліфікаційної роботи
«Система віддаленого контролю та управління IoT-пристроями»
08-54.БКР.009.00.000 ТЗ

Науковий керівник к.т.н., доц. каф. ОТ
_____ Колесник І.С.

Студент групи 1КІ-23мс
_____ Зять М.В.

Вінниця 2025

1 Підстава виконання магістерської кваліфікаційної роботи

Наказ про затвердження теми бакалаврськоїкваліфікаційної роботи від 20.03.2025 р. №97.

2 Мета та призначення БКР

2.1 Мета роботи — розробка програмно-апаратного комплексу призначена для віддаленого моніторингу та управління пристроями Інтернету речей.

2.2 Призначення розробки—виконання бакалаврської кваліфікаційної роботи.

3 Вихідні дані для виконання БКР

Вихідні дані: технічний опис архітектури та технологій IoT: платформа Raspberry Pi, Wi-Fi адаптер Auspi Wireless Adapter і, SSH-з’єднання, мова програмування Java та фреймворк Spring Boot.

4. Вимоги до виконання БКР

БКР повинна задовольняти такі вимоги:

— аналіз проблем та вимог до вебзастосунків та telegram-ботів;

— аналіз сучасних підходів до побудови IoT-систем та їх архітектурні особливості;

— дослідження існуючих технологій, протоколів та засобів для реалізації зв’язку між IoT-пристроями;

— підбір оптимальних апаратних компонентів для реалізації IoT-рішення;

— створення програмного забезпечення для збору, передачі та візуалізації даних;

— тестування системи в умовах, наближених до реального використання, та оцінити її ефективність.

5 Етапи виконання БКР та очікувані результати наведені в табл.. А1.

Таблиця А.1 — Етапи виконання роботи

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Постановка задачі роботи	21.03.25	
2	Пошук матеріалів по технологіях IoT	21.03.25— 29.03.25	
3	Огляд існуючих рішень для моніторингу та керування IOT-пристроями	21.03.25— 29.03.25	
4	Встановлення та налаштування системи Domoticz на Raspberry Pi	30.03.25— 11.04.25	
5	Створення програмного забезпечення для збору, передачі та візуалізації даних	12.04.25— 26.04.25	
6	Тестування та демонстрація роботи системи	27.04.25— 07.05.25	
7	Оформлення пояснювальної записки та ілюстративного матеріалу	08.05.25	
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	14.05.25	

6 Матеріали, що подаються до захисту БКР

Пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відзив наукового керівника, рецензія опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР.

8.1 При оформлювання БДР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи Система віддаленого контролю та управління IoT-пристроями

Тип роботи: Бакалаврська кваліфікаційна робота
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет (інститут), навчальна група)

Керівник Колесник І.С. доц. каф. ОТ
(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	79%
Загальнасхожість	21%

Аналіз звіту подібності (відмітитипотрібне)

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор Зять М.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____
_____
_____
_____

Особа, відповідальна за перевірку _____ Захарченко С.М
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

ДОДАТОК В

Графічна частина

СХЕМА ВЗАЄМОДІЇ ЗАСОБІВ ІОТ

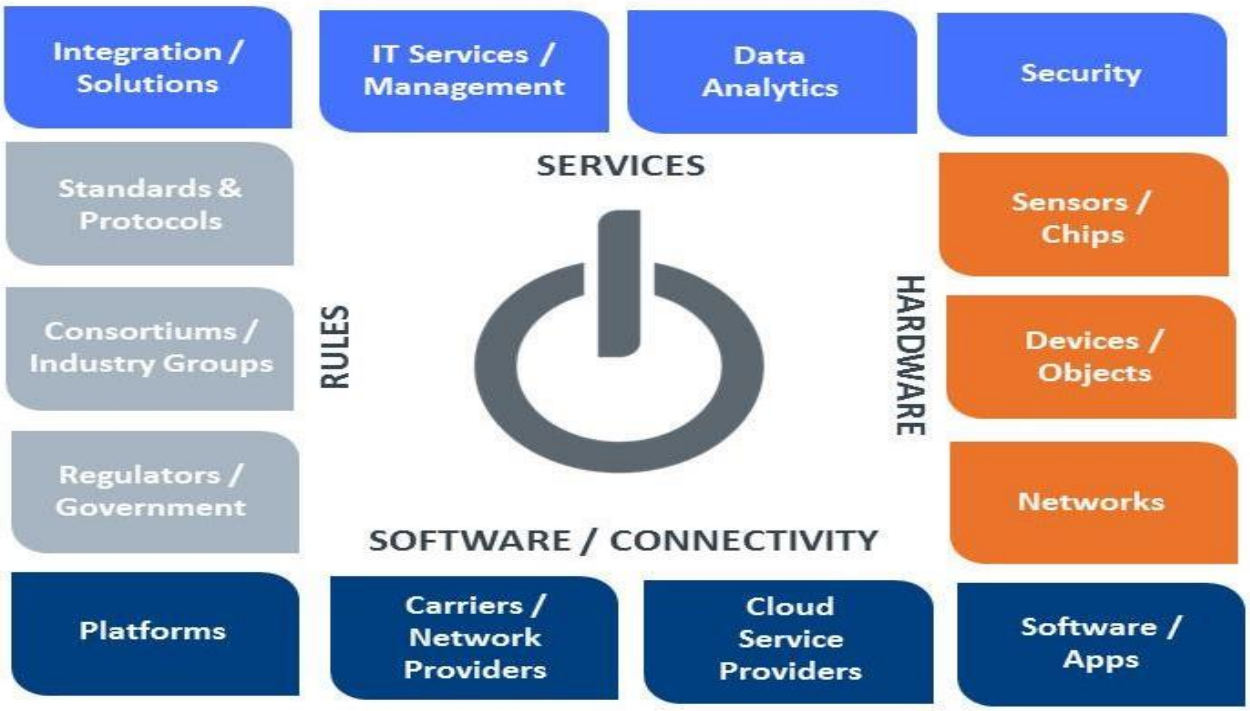


Рисунок В.1 —Схема взаємодії засобів

ДОДАТОК Г

ілюстративна частина

СХЕМА ШИНИ GPIO ПЛАТФОРМИ RASPBERRYPI

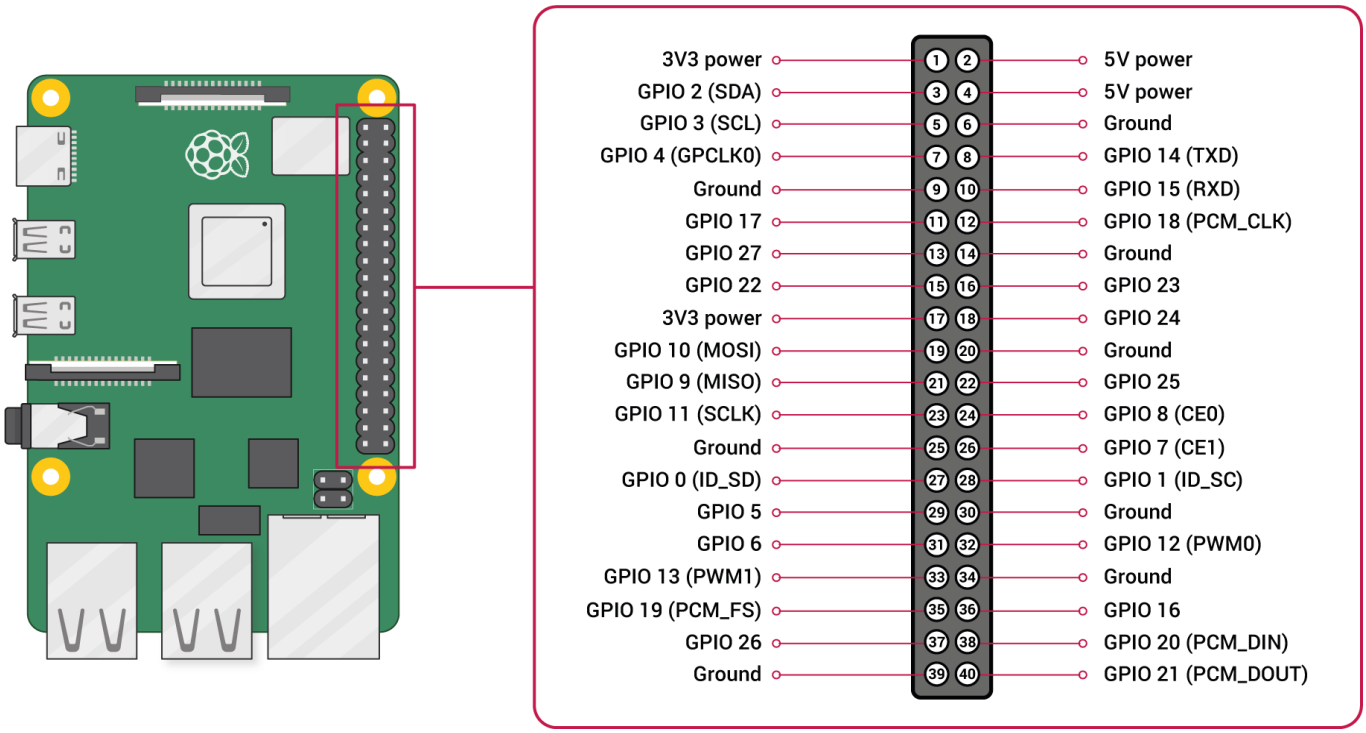


Рисунок Г.1 — Схема шини GPIO платформи RaspberryPi

ДОДАТОК Д
Лістинг коду програми

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals'; import {UserProvider} from './context/userContext';
import '../node_modules/react-draft-wysiwyg/dist/react-draft-wysiwyg.css';
ReactDOM.render(
<React.StrictMode>
<UserProvider>
<App />
</UserProvider>
</React.StrictMode>,
document.getElementById('root'),
);
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals reportWebVitals();
import React from 'react';
import {BrowserRouter, Link, Navigate, Route, Routes} from 'react-router-dom'; import PublicLayout from
'./components/templates/PublicLayout';
import {HOME, LOGIN} from './constants/routes'; import {publicRoutes} from './routes/routes'; import
PublicRoute from './routes/PublicRoute';
import ProtectedLayout from './components/templates/ProtectedLayout'; import ProtectedRoute from
'./routes/ProtectedRoute'; import Courses from './pages/Courses';
import Course from './pages/Course';
function App() {
return (
<BrowserRouter>
<Routes>
<Route path="/" element={<PublicLayout />}>
<Route index element={<Navigate to={LOGIN} replace />} /> {publicRoutes.map(({path, element}) => (
<Route
key={path}
path={`/${path}`}
element={<PublicRoute>{element}</PublicRoute>} />
))}
</Route>
<Route path="/" element={<ProtectedLayout />}> <Route
path={HOME}
element={<ProtectedRoute><Courses/></ProtectedRoute>} />
<Route path="course">
<Route path=":courseId" element={
<ProtectedRoute><Course /></ProtectedRoute>
} />
</Route>
</Route>
<Route path="*" element={<Link to="login">Go to log in</Link>} /> </Routes>
</BrowserRouter>
);
}
export default App;
import firebase from 'firebase';
const firebaseConfig = {
apiKey: 'AIzaSyCkcUIPkZWnk5ur4-lcHen2WJ18Lha8mmY',
authDomain: 'eduspace-15506.firebaseapp.com',
projectId: 'eduspace-15506',
```

```
storageBucket: 'eduspace-15506.appspot.com',
messagingSenderId: '895198289171',
appId: '1:895198289171:web:4e0156630f43f1bb890ee6', };
const app = firebase.initializeApp(firebaseConfig); const auth = app.auth();
const db = app.firestore();
const checkUser = async (user) => {
const querySnapshot = await db
.collection('users')
.where('uid', '==', user.uid)
.get();
if (querySnapshot.docs.length === 0) {
// create a new user
await db.collection('users').add({
uid: user.uid,
name: user.displayName,
photo: user.photoURL,
});
}
};
const googleProvider = new firebase.auth.GoogleAuthProvider();
// Sign in and check or create account in firestore const signInWithGoogle = async (onError) => {
try {
const response = await auth.signInWithPopup(googleProvider); await checkUser(response.user);
} catch (err) { onError(err.message);
}
};
const signUpWithEmailAndPassword = async ( email,
password,
firstName,
lastName,
onError,
)=>{
try {
const {user} = await auth.createUserWithEmailAndPassword(email, password); const tempUser = {...user,
displayName: `${firstName} ${lastName}`}; await checkUser(tempUser);
} catch (err) {
onError(err.message);
}
};
const logout = () => {
auth.signOut();
};
export {app, auth, db, signInWithGoogle, signUpWithEmailAndPassword, logout};
import React from 'react';
import LoginForm from '../components/organisms/LoginForm';
function Login() {
return (
<LoginForm />
);
}
export default Login;
import React from 'react';
import SignupForm from '../components/organisms/SignupForm';
function Login() {
return (
<SignupForm />
);
}
export default Login;
```

```
import React, {useEffect, useState} from 'react'; import {
Container, Divider, Grid,
} from '@mui/material';
import H1 from '../components/atoms/H1';
import H2 from '../components/atoms/H2';
import CourseCard from '../components/molecules/CourseCard'; import AddCourseCard from '../components/molecules/AddCourseCard'; import {db} from '../firebase';
import DialogModal from '../components/organisms/DialogModal'; import useUser from '../hooks/useUser';
import JoinCourseModal from '../components/organisms/JoinCourseModal'; import moment from 'moment';
function Courses() {
const [isCreateModalOpen, setIsCreateModalOpen] = useState(false); const [isJoinModalOpen, setIsJoinModalOpen] = useState(false); const {user} = useUser();
const [enrolledCourses, setEnrolledCourses] = useState([]);
const [createdCourses, setCreatedCourses] = useState([]); const [loading, setLoading] = useState(false);
const fetchCourses = async () => {
try {
setLoading(true);
await user.ref.collection('enrolledCourses')
.onSnapshot((snapshot) => {
const courses = snapshot?.docs.map((doc)=>{ return doc.data();
});
setEnrolledCourses(courses);
});
await user.ref.collection('createdCourses')
.onSnapshot((snapshot) => {
const courses = snapshot?.docs.map((doc)=>{ return doc.data();
});
setCreatedCourses(courses);
});
} catch (error) { console.error(error.message);
} finally {
setLoading(false);
}
};
useEffect(() => {
fetchCourses();
}, [user]);
const joinClass = async (courseId) => { try {
setIsJoinModalOpen(false);
const classRef = await db.collection('courses').doc(courseId).get(); if (!classRef.exists) {
return alert(`Class doesn't exist, please provide correct ID`);
}
await classRef.ref.collection('users').add({ name: user.name,
joiningDate: moment().format('MMM Do YY'), courseId,
studentUid: user.uid,
});
const classData = await classRef.data();
await user.ref.collection('enrolledCourses').add({ creatorName: classData.creatorName,
id: courseId,
name: classData.name,
description: classData.description,
});
alert(`Enrolled in ${classData.name} successfully!`);
} catch (err) { console.error(err);
alert(err.message);
}
};
const createClass = async (courseName, courseDescription) => { setIsCreateModalOpen(false);
try {
```

```
const {id} = await db.collection('courses').add({
creatorName: user.name,
name: courseName,
description: courseDescription,
users: [],
});
await user.ref.collection('createdCourses').add({ id,
name: courseName,
description: courseDescription,
});
} catch (err) {
alert(`Cannot create class – ${err.message}`);
}
};
if (loading) return 'Loading';
if (!user) return 'User not found';
return (
<div>
<H1 sx={{textAlign: 'center', margin: '16px 0'}}> Welcome,
{' '}
{user.name}
</H1>
<Divider />
<Container>
<H2 sx={{margin: '12px 0'}}>My created courses</H2> <Grid container spacing={2}>
{createdCourses && createdCourses.map(({name, description, id}) => ( <Grid item xs={6} md={4} lg={3}
key={id}>
<CourseCard title={name} description={description} id={id} /> </Grid>
))}
<Grid item xs={6} md={4} lg={3} key="create"> <AddCourseCard
title="Create your course"
description="Create your own course and invite your students"
onClick={() => setIsCreateModalOpen(true)}
/>
</Grid>
</Grid>
</Container>
<Divider sx={{marginTop: 4}} />
<Container>
<H2 sx={{margin: '12px 0'}}>My enrolled courses</H2> <Grid container spacing={2}>
{enrolledCourses && enrolledCourses.map( ({name, description, id, creatorName}) => (
<Grid item xs={6} md={4} lg={3} key={id}> <CourseCard
title={name}
description={description}
author={creatorName}
id={id}/>
</Grid>
))}
<Grid item xs={6} md={4} lg={3} key="join"> <AddCourseCard
title="Join course"
description="Join course"
onClick={()=>setIsJoinModalOpen(true)}/> </Grid>
</Grid>
</Container>
<JoinCourseModal
title="Join course"
label="ID"
isOpen={isJoinModalOpen}
handleClose={()=>setIsJoinModalOpen(false)} handleSubmit={joinClass}
```

```
/>
<DialogModal
title="Create class"
label="Name"
isOpen={isCreateModalOpen}
handleClose={()=>setIsCreateModalOpen(false)} handleSubmit={createClass}
/>
</div>
);
}
export default Courses;
import React, {useEffect, useState} from 'react';
import {Box, Tab, Tabs, Typography} from '@mui/material'; import {db} from '../firebase';
import {useParams} from 'react-router-dom'; import useUser from '../hooks/useUser';
import Users from '../components/organisms/Users'; import Stream from '../components/organisms/Stream'; import Tasks from '../components/organisms/Tasks'; import {TeacherContext} from '../context/TeacherContext'; import {node, number} from 'prop-types';
function TabPanel(props) {
const {children, value, index, ...other} = props;
return (
<div
role="tabpanel"
hidden={value !== index}
id={`simple-tabpanel-${index}`} aria-labelledby={`simple-tab-${index}`} {...other}
>
{value === index && (
<Box sx={{p: 3}}>
{children}
</Box>
)}
</div>
);
}
TabPanel.propTypes = {
children: node.isRequired,
value: number.isRequired,
index: number.isRequired,
};
function a11yProps(index) {
return {
'id': `simple-tab-${index}`,
'aria-controls': `simple-tabpanel-${index}`, };
}
const Course = () => {
const [isTeacher, setIsTeacher] = useState(false); const [value, setValue] = React.useState(0); const handleChange = (event, newValue) => {
setValue(newValue);
};
const [classData, setClassData] = useState({});
const {user} = useUser();
const {courseId} = useParams();
useEffect(() => {
db.collection('courses')
.doc(courseId)
.onSnapshot((snapshot) => {
const data = snapshot.data();
console.log(user);
setClassData(data);
});
```

```
user.ref.collection('createdCourses').where('id', '==', courseId)
.onSnapshot((snapshot)=>{
if (snapshot.docs.length > 0) setIsTeacher(true); else setIsTeacher(false);
});
}, [courseId]);
return (
<TeacherContext.Provider value={isTeacher}> <Box sx={{width: '100%'}}>
<Box sx={
{display: 'flex',
justifyContent: 'space-around',
borderBottom: 1,
borderColor: 'divider',
padding: '16px 0',
}
}>
<Typography variant="h6" color="primary"> Course Name: {classData?.name}
</Typography>
<Typography variant="h6" color="primary"> Course ID: {courseId}
</Typography>
</Box>
<Box sx={{borderBottom: 1, borderColor: 'divider'}}> <Tabs
value={value}
onChange={handleChange}
aria-label="basic tabs example"
>
<Tab label="Stream" {...a11yProps(0)} />
<Tab label="Tasks" {...a11yProps(1)} />
<Tab label="People" {...a11yProps(2)} />
<Tab label="Description" {...a11yProps(3)} />
</Tabs>
</Box>
<TabPanel value={value} index={0}>
<Stream />
</TabPanel>
<TabPanel value={value} index={1}>
<Tasks/>
</TabPanel>
<TabPanel value={value} index={2}>
<Users />
</TabPanel>
<TabPanel value={value} index={3}>
{classData.description}
</TabPanel>
</Box>
</TeacherContext.Provider>
);
};
export default Course;
import {useContext} from 'react';
import {UserContext} from '../context/userContext';
const useUser = () => {
const {user, setUser} = useContext(UserContext); return {user, setUser};
};
export default useUser;
import React, {useEffect, useState, createContext} from 'react'; import {useAuthState} from 'react-firebase-hooks/auth';
import {auth, db} from '../firebase';
import {node} from 'prop-types';
export const UserContext = createContext({ user: null,
```

```
setUser: ()=>{},
});
export const UserProvider = ({children}) => { const [user, setUser] = useState(null); const [authUserAuth] = useAuthState(auth);
useEffect(() => {
if (authUserAuth) {
const unsubscribe = db.collection('users')
.where('uid', '==', authUserAuth.uid)
.onSnapshot((snapshot) => {
const docRef = snapshot?.docs[0];
setUser({...docRef.data(), ref: docRef.ref}); });
return () => {
unsubscribe();
};
} else {
setUser(null);
}
}, [authUserAuth]);
return (
<UserContext.Provider
value={{user, setUser}}
>
{children}
</UserContext.Provider>
);
};
UserProvider.propTypes = {
children: node.isRequired,
};
import {createContext} from 'react';
export const TeacherContext = createContext(false);
export const LOGIN = '/login';
export const SIGNUP = '/signup';
export const HOME = '/home';
export const COURSE = '/course/:courseId';
import React from 'react';
import {
Box,
} from '@mui/material';
import {Outlet} from 'react-router-dom';
import PublicHeader from '../organisms/PublicHeader'; import Container from '../atoms/Container';
function PublicLayout() {
return (
<div>
<PublicHeader />
<Container>
<Box sx={{
mx: 'auto',
mt: 4,
maxWidth: 500,
}}
>
<Outlet />
</Box>
</Container>
</div>
);
}
export default PublicLayout;
```

```
import React from 'react';
import {
Outlet,
} from 'react-router-dom';
import ProtectedHeader from '../organisms/ProtectedHeader';
function ProtectedLayout() {
return (
<div>
<ProtectedHeader />
<Outlet />
</div>
);
}
export default ProtectedLayout;
/* eslint-disable max-len */
import * as React from 'react';
import Table from '@mui/material/Table';
import TableBody from '@mui/material/TableBody'; import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer'; import TableHead from '@mui/material/TableHead';
import TableRow from '@mui/material/TableRow'; import Paper from '@mui/material/Paper'; import {db} from '../../firebase';
import {useEffect, useState} from 'react'; import {useParams} from 'react-router-dom';
export default function Users() {
const [users, setUsers] = useState([]);
const [completedTasks, setCompletedTasks] = useState([]);
const {courseId} = useParams();
useEffect(async ()=>{
const tempUsers = await db.collection(`courses/${courseId}/users`).get(); setUsers(tempUsers.docs.map((doc)=>doc.data()));
const tempCompletedTasks = await
db.collection('completedTasks').where('courseId', '==', courseId).get();
setCompletedTasks(tempCompletedTasks);
}, [courseId]);
const getCurrentRating = (studentUid) => { let rating = 0;
completedTasks.docs?.filter((doc)=>doc.data().studentUid ===
studentUid).forEach((task)=>{
rating+=parseInt(task.data()?.rating);
});
return rating;
};
return (<>
{users.length === 0 ? <p style={{textAlign: 'center', color: 'gray', marginBottom: '12px'}}>No students.<br/> Share the course ID with your students so they can join</p> :
<TableContainer component={Paper}>
<Table sx={{minWidth: 650}} aria-label="simple table"> <TableHead>
<TableRow>
<TableCell>№</TableCell>
<TableCell>Name</TableCell>
<TableCell align="right">Joining date</TableCell> <TableCell align="right">Current rate</TableCell>
</TableRow>
</TableHead>
<TableBody>
{users.map((user, index) => (
<TableRow
key={user.name}
sx={{'&:last-child td, &:last-child th': {border: 0}}}
>
<TableCell>
```

```
{index+1}
</TableCell>
<TableCell component="th" scope="row">
{user.name}
</TableCell>
<TableCell
align="right">{user.joiningDate}</TableCell> <TableCell
align="right">{getCurrentRating(user.studentUid)}</TableCell>
</TableRow>
))}
</TableBody>
</Table>
</TableContainer>}
</>
);
}
/* eslint-disable react/prop-types,max-len */ import React, {useEffect, useState} from 'react'; import {db} from '../../firebase';
import TableRow from '@mui/material/TableRow'; import TableCell from '@mui/material/TableCell'; import IconButton from '@mui/material/IconButton';
import KeyboardArrowUpIcon from '@mui/icons-material/KeyboardArrowUp';
import KeyboardArrowDownIcon from '@mui/icons-material/KeyboardArrowDown';
import moment from 'moment';
import {
Button,
Dialog,
DialogActions,
DialogContent,
DialogContentText,
DialogTitle,
} from '@mui/material';
import Collapse from '@mui/material/Collapse'; import Box from '@mui/material/Box';
import CompletedTasks from '../molecules/CompletedTasks'; import {useParams} from 'react-router-dom';
import TableContainer from '@mui/material/TableContainer'; import Paper from '@mui/material/Paper'; import Table from '@mui/material/Table';
import TableHead from '@mui/material/TableHead'; import TableBody from '@mui/material/TableBody';
import CreateTaskModal from '../molecules/CreateTaskModal';
const TeacherTask = ({task}) => {
const [open, setOpen] = React.useState(false);
const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);
const [isDescriptionModalOpen, setIsDescriptionModalOpen] = useState(false); const deleteTask = async () => {
await db.collection('tasks').doc(task.id).delete(); const tempCompletedTasks = await
db.collection('completedTasks').where('taskId', '==', task.id).get(); tempCompletedTasks.docs.forEach((doc)=>{
doc.ref.delete();
});
};
return (
<React.Fragment>
<TableRow sx={{'& > *': {borderBottom: 'unset'}}}>
<TableCell>
<IconButton
aria-label="expand row"
size="small"
onClick={() => setOpen(!open)}
>
{open ? <KeyboardArrowUpIcon /> : <KeyboardArrowDownIcon />} </IconButton>
</TableCell>
```

```
<TableCell component="th" scope="row">
{task.name}
</TableCell>
<TableCell align="right">{task.maxRating}</TableCell>
<TableCell align="right"> {moment(task.deadline?.toDate() || 0).format('L')}</TableCell>
<TableCell align="right">
<Button variant="text"
onClick={()=>setIsDescriptionModalOpen(true)}>Open</Button> <Button variant="text" color="error"
onClick={()=>setIsDeleteModalOpen(true)}>Delete</Button> </TableCell>
</TableRow>
<TableRow>
<TableCell style={{paddingBottom: 0, paddingTop: 0}} colSpan={6}> <Collapse in={open} timeout="auto" unmountOnExit>
<Box sx={{margin: 1}}>
<CompletedTasks taskId={task.id} maxRating={task.maxRating} deadline={task.deadline}/>
</Box>
</Collapse>
</TableCell>
</TableRow>
<Dialog
open={isDeleteModalOpen}
onClose={()=>setIsDeleteModalOpen(false)} aria-labelledby="alert-dialog-title" aria-describedby="alert-dialog-description"
>
<DialogTitle id="alert-dialog-title">
{`Are you sure you want to delete ${task.name}?`}
</DialogTitle>
<DialogContent>
<DialogContentText id="alert-dialog-description">
Once deleted, no one will be able to see this task.
</DialogContentText>
</DialogContent>
<DialogActions>
<Button onClick={()=>setIsDeleteModalOpen(false)}>Cancel</Button> <Button onClick={deleteTask} color="error" autoFocus>
Confirm
</Button>
</DialogActions>
</Dialog>
<Dialog
open={isDescriptionModalOpen}
onClose={()=>setIsDescriptionModalOpen(false)} aria-labelledby="alert-dialog-title" aria-describedby="alert-dialog-description"
>
<DialogTitle id="alert-dialog-title">
Description
</DialogTitle>
<DialogContent>
<div dangerouslySetInnerHTML={{__html: task.description}}></div>
</DialogContent>
<DialogActions>
<Button onClick={()=>setIsDescriptionModalOpen(false)}>Cancel</Button> </DialogActions>
</Dialog>
</React.Fragment>
);
};
function TeacherTasks() {
const [isModalOpen, setIsModalOpen] = useState(false); const {courseId} = useParams();
const [tasks, setTasks] = useState([]);
```

```
useEffect(()=>{
db.collection('tasks').where('courseId', '==', courseId)
.orderBy('deadline', 'asc').onSnapshot((snapshot)=>{
const tempTasks = [];
snapshot.docs.forEach((doc)=>tempTasks.push({...doc.data(), id:
doc.id}));
setTasks(tempTasks);
});
}, [courseId]);
return (<>
{tasks.length === 0 ? <p style={{textAlign: 'center', color: 'gray',
marginBottom: '12px'}}>No tasks</p> : <TableContainer component={Paper}>
<Table aria-label="collapsible table">
<TableHead>
<TableRow>
<TableCell/>
<TableCell>Name</TableCell>
<TableCell align="right">Max rating</TableCell> <TableCell align="right">Deadline</TableCell> <TableCell
align="right">Actions</TableCell>
</TableRow>
</TableHead>
<TableBody>
{tasks.map((task) => (
<TeacherTask key={task.id} task={task} /> ))}
</TableBody>
</Table>
</TableContainer>}
<Box sx={{display: 'flex', justifyContent: 'end', p: 2}}>
<Button variant="contained" onClick={()=>setIsModalOpen(true)}>
Create new task
</Button>
</Box>
<CreateTaskModal isOpen={isModalOpen} handleClose={()=>setIsModalOpen(false)}/>
</>
);
};
export default TeacherTasks;
import * as React from 'react';
import {useContext} from 'react';
import {TeacherContext} from '../../context/TeacherContext'; import StudentTasks from './StudentTasks';
import TeacherTasks from './TeacherTasks';
export default function Tasks() {
const isTeacher = useContext(TeacherContext); if (isTeacher) return <TeacherTasks />; else return
<StudentTasks />;
}
/* eslint-disable react/prop-types,max-len */ import React, {useEffect, useState} from 'react'; import
{useParams} from 'react-router-dom'; import {db} from '../../firebase';
import TableContainer from '@mui/material/TableContainer'; import Paper from '@mui/material/Paper';
import Table from '@mui/material/Table';
import TableHead from '@mui/material/TableHead'; import TableRow from '@mui/material/TableRow';
import TableCell from '@mui/material/TableCell'; import TableBody from '@mui/material/TableBody'; import {
Button,
Dialog,
DialogActions,
DialogContent,
DialogContentText,
DialogTitle,
} from '@mui/material'; import moment from 'moment';
import useUser from '../../hooks/useUser'; import firebase from 'firebase';
```

```
import {EditorState} from 'draft-js'; import {Editor} from 'react-draft-wysiwyg';
import {convertContentToHTML} from '../../utils/convertContentToHTML';
const Task = ({number, task, isSent, currentRating}) => { const [isModalOpen, setIsModalOpen] = useState(false);
const {user} = useUser();
const [content, setContent] = useState(EditorState.createEmpty()); const {courseId} = useParams();
const sendWork = (taskId) => {
try {
db.collection('completedTasks').add({
taskId,
student: user.name,
studentUid: user.uid,
date: firebase.firestore.FieldValue.serverTimestamp(),
content: convertContentToHTML(content), courseId,
});
setContent(EditorState.createEmpty());
setIsModalOpen(false);
} catch (e) {
alert(e.message);
}
};
return <React.Fragment key={task.id}>
<TableRow
sx={{'&:last-child td, &:last-child th': {border: 0}}}
>
<TableCell>
{number}
</TableCell>
<TableCell component="th" scope="row">
{task.name}
</TableCell>
<TableCell align="right">{task.maxRating}</TableCell>
<TableCell align="right">{moment(task.deadline?.toDate() || 0).format('L')}</TableCell>
<TableCell align="right">{isSent ? (currentRating ? currentRating :
'Evaluating') : 'Not sent'}</TableCell>
<TableCell align="right"> <Button disabled={isSent} variant="text"
onClick={()=>setIsModalOpen(true)}>{isSent ? 'Sent' : 'Send work'}</Button></TableCell>
</TableRow>
<Dialog open={isModalOpen} onClose={()=>setIsModalOpen(false)}> <DialogTitle>Send your
work</DialogTitle> <DialogContent>
<DialogContentText>
Task
</DialogContentText>
<div dangerouslySetInnerHTML={{__html: task.description}}></div> <DialogContentText>
Please enter answer to your teacher evaluates it.
</DialogContentText>
<Editor
editorState={content}
wrapperClassName='demo-wrapper'
editorClassName="send-work-editor"
onEditorStateChange={setContent}
placeholder="Add your answer"
/>
</DialogContent>
<DialogActions>
<Button onClick={()=>setIsModalOpen(false)}>Cancel</Button> <Button
onClick={()=>sendWork(task.id)}>Submit</Button>
</DialogActions>
</Dialog>
```