КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Кафедра фізики високих технологій та електроніки
Кафедра інформаційних систем та технологій

І.Р. ПАРХОМЕЙ
В.А. ДРУЖИНІН
М.П. ТРЕМБОВЕЦЬКИЙ
О.С. БОНДАРЕНКО

ТЕХНОЛОГІЇ ІНТЕРНЕТ РЕЧЕЙ

методичні рекомендації до практичних робіт
для здобувачів освітнього ступеня «магістр» спеціальності
E6 «Прикладна фізика та наноматеріали» /
F6 «Інформаційні системи і технології»
освітня програма «Фізика інформаційних технологій»

За загальною редакцією
в.о. завідувача кафедри фізики високих технологій та електроніки
ННІ високих технологій КНУ імені Тараса Шевченка
Прокопенка Олександра Володимировича

Київ – 2025

Рецензенти:

д.т.н., професор, завідувач кафедри телекомунікаційних систем та мереж Державного університету інформаційно-комунікаційних технологій Заїка Віктор Федорович;

д.т.н., професор, професор кафедри кібербезпеки та захисту інформації Київського національного університету імені Тараса Шевченка Толюпа Сергій Васильович.

Рекомендовано до публікації кафедрою фізики високих технологій та електроніки, протокол № 8 від «26» вересня 2025 р.

Рекомендовано до публікації кафедрою інформаційних систем та технологій, протокол № 2_25/26 від «11» вересня 2025 р.

Рекомендовано до публікації Вченою радою Навчально-наукового інституту високих технологій, протокол № 4 від «8» жовтня 2025 р.

І.Р. Пархомей, В.А. Дружинін, М.П. Трембовецький, О.С. Бондаренко

Технології інтернет речей [Електронний ресурс]: методичні рекомендації до практичних робіт для здобувачів освітнього ступеня «магістр» спеціальності Е6 «Прикладна фізика та наноматеріали» / F6 «Інформаційні системи і технології» освітня програма «Фізика інформаційних технологій» / Укл. І.Р. Пархомей, В.А. Дружинін, М.П. Трембовецький, О.С. Бондаренко; за заг. ред. О.В. Прокопенка. – К.: КНУ імені Тараса Шевченка, 2025.– 69 с.

Містить п'ять практичних робіт, метою яких є формування у здобувачів знань і практичних навичок, пов'язаних з основними технологіями Інтернету речей. Кожна робота включає стислий теоретичний матеріал, покроковий опис виконання та контрольні питання для самоперевірки.

Відповідальний за оформлення та дизайн: Бойко Д.М., аспірант кафедри інформаційних систем та технологій

Публікується в авторській редакції.

Електронна версія цього видання опублікована на сайті кафедри інформаційних систем та технологій факультету інформаційних технологій Київського національного університету імені Тараса Шевченка за адресою: https://www.ist.fit.knu.ua/

_____

(дата публікації «___» _____ 20___ р.)

ЗМІСТ

Вступ 4

Практична робота № 1. Node-RED: основи, обробка даних та інтеграція у мікросистемних технологіях 6

Практична робота № 2. Інтеграція Node-RED із зовнішніми сервісами та інтерфейсами в мікросистемних технологіях 19

Практична робота № 3. Аналіз промислових IoT-даних із застосуванням кластеризації та технології MapReduce 41

Практична робота № 4. Інтелектуальний пошук і керування даними у Smart Factory за допомогою Apache Spark 50

Практична робота № 5. Задачі класифікації та регресії з використанням бібліотеки машинного навчання MLlib у контексті розумного виробництва 59

Перелік посилань 68

ВСТУП

Підготовка фахівців освітнього ступеня «магістр» за спеціальностями Е6 «Прикладна фізика та наноматеріали» та F6 «Інформаційні системи та технології» освітньої програми «Фізика інформаційних технологій» передбачає опанування актуальних цифрових технологій, зокрема технологій Інтернету речей (IoT) та засобів аналізу великих масивів даних (Big Data).

Під Інтернетом речей (IoT) розуміють концепцію побудови розподіленої мережі, у якій фізичні об'єкти, оснащені вбудованими датчиками та програмними модулями, взаємодіють із цифровими системами через стандартизовані протоколи передачі даних. Технології IoT охоплюють широке коло застосувань: автоматизоване управління будівлями та міською інфраструктурою (Smart City), промислову автоматизацію (IIoT), інтелектуальні енергетичні мережі (Smart Grid), цифрові виробничі системи (Smart Factory), а також алгоритми машинного навчання та методи обробки великих даних.

Великі дані розглядаються як сукупність технологій і методів, призначених для обробки значних за обсягом та різнорідних потоків інформації, що швидко оновлюються. Їхні ключові характеристики позначаються поняттям «трьох V» (Volume, Velocity, Variety): обсяг даних, швидкість їх надходження та різноманітність форматів. Використання цих технологій відкриває можливості для виявлення закономірностей у даних і створює підґрунтя для розвитку інтелектуальних сервісів у різних сферах діяльності.

Методичні рекомендації орієнтовані на практичне ознайомлення з інструментами та платформами, що забезпечують функціонування IoT-систем та аналітику зібраних даних. Спершу розглядається середовище Node-RED, яке використовується для побудови потоків даних, інтеграції пристроїв і сервісів, реалізації протоколів обміну (зокрема Modbus) та створення інтерфейсів моніторингу. Далі подається матеріал, присвячений принципам обробки великих даних за допомогою парадигми MapReduce та платформи Hadoop, а також використанню сучасних інструментів аналітики, зокрема Apache Spark SQL.

Завершальний блок робіт присвячено методам машинного навчання з використанням бібліотеки MLlib, що дозволяє опанувати алгоритми класифікації, регресії та ансамблеві методи аналізу даних.

У комплексі ці практичні завдання формують логічно завершену навчальну траєкторію, яка відображає основні етапи розвитку IoT-систем: від збору та інтеграції даних – до їх масштабної обробки і глибинного аналітичного аналізу із застосуванням методів машинного навчання.

Загальний порядок виконання комп’ютерних практикумів включає:

- опрацювання теоретичних відомостей до відповідної теми;
- виконання індивідуального завдання;
- складання протоколу практичної роботи;
- подання та захист результатів.

ПРАКТИЧНА РОБОТА № 1

Node-RED: основи, обробка даних та інтеграція у мікросистемних технологіях

Мета роботи. Ознайомитися з принципами візуального програмування в середовищі Node-RED, навчитися створювати базові потоки для обробки повідомлень, використовувати JavaScript-об’єкти та здійснювати аналіз системної інформації. Закріпити навички, необхідні для побудови мікросистем з підтримкою збору, перетворення і діагностики даних.

Теоретичні відомості

Node-RED – це програмний інструмент, що забезпечує інтеграцію апаратних пристроїв, прикладних програмних інтерфейсів (API) та онлайн-сервісів у єдиний потік обробки даних. Ключовим елементом середовища є браузерний редактор, у якому потоки (flows) конструюються шляхом з'єднання функціональних вузлів (nodes) із доступної палітри компонентів. Весь процес редагування відбувається безпосередньо у браузері без необхідності встановлення додаткового програмного забезпечення. Після завершення побудови логіки потік розгортається у середовище виконання одним натисканням кнопки Deploy, що суттєво скорочує час між розробкою та введенням рішення в дію.

У середовищі Node-RED реалізовано текстовий редактор, що дозволяє створювати функціональні вузли на основі мови JavaScript. Вбудована бібліотека забезпечує збереження та повторне використання корисних функцій, шаблонів і потоків.

Node-RED розроблено на базі платформи Node.js, яка використовує подієво-орієнтовану неблокуючу модель обробки даних. Така архітектура сприяє ефективній роботі на периферійних пристроях (Edge), таких як Raspberry Pi, а також у хмарних середовищах.

Функціональність Node-RED легко розширюється завдяки інтеграції більш ніж 225 000 модулів, доступних у сховищі Node Package Manager (npm), що дозволяє додавати нові можливості.

Потоки зберігаються у форматі JSON, що спрощує їх перенесення між середовищами та обмін між розробниками. Для публікації та пошуку готових рішень доступна офіційна онлайн-бібліотека потоків.

Завдяки відкритій архітектурі, модульності та зручності використання, Node-RED виступає потужним інструментом для розробки IoT-рішень, автоматизації, збору та обробки даних у режимі реального часу (Рис. 1.1).

Рисунок 1.1 – Мультиплатформна інтеграція Node-RED у мікросистемних технологіях: локальні, периферійні та хмарні середовища

Завдання до практичної роботи

1. Виконати інсталяцію Node-RED.

2. Ознайомитись з Node-RED.

3. Виконати роботу з JS-об’єктами та здійснити обробку системної інформації.

Хід роботи

Завдання 1. Інсталяція Node-RED під Windows.

Node-RED підтримується на сучасних версіях Windows (7, 8, 10). Для його встановлення спочатку необхідно підготувати середовище – встановити Node.js, що забезпечує виконання JavaScript-проєктів, і разом із ним менеджер пакунків npm.

Перейдіть на офіційний сайт Node.js (https://nodejs.org/uk/) та завантажте інсталяційний файл версії LTS. Встановлення слід виконати з правами адміністратора, залишаючи налаштування за замовчуванням. Після завершення інсталяції відкрийте командний рядок і перевірте коректність встановлення, ввівши:

node --version && npm –version

Команда має повернути версії Node.js і npm, що підтверджує готовність системи до подальшої роботи.

npm (Node Package Manager) – штатний менеджер залежностей для JavaScript-проєктів, що функціонує у середовищі Node.js. Він складається з двох компонентів: CLI-клієнта, що виконується у командному рядку, та централізованого онлайн-реєстру пакетів, який містить як відкриті публічні бібліотеки, так і приватні корпоративні модулі. Завдяки npm зовнішні залежності встановлюються, оновлюються, видаляються та версіонуються автоматично – без необхідності ручного відстеження сумісності між пакетами.

Node-RED встановлюється через npm. У тому ж вікні командного рядка виконайте:

npm install -g --unsafe-perm node-red

Це запустить автоматичне завантаження та встановлення необхідних компонентів. Після завершення можна запускати середовище Node-RED командою:

node-red

При першому запуску Windows може запитати дозвіл на доступ до мережі – його потрібно надати. Після запуску Node-RED відкривається в браузері за адресою http://127.0.0.1:1880/.

Важливо не закривати вікно командного рядка, оскільки воно забезпечує роботу сервера.

Завдання 2. Знайомство з редактором Node-RED

Програма в середовищі Node-RED складається з потоків (flows) – логічних структур, що діють як незалежні фрагменти коду. Потік являє собою сукупність вузлів (nodes), з'єднаних між собою інформаційними дротами (wires). Кожен вузол виконує окрему функцію – від обробки даних до інтеграції з зовнішніми сервісами.

1. Відкрийте в браузері редактор Node-RED. Ознайомтеся з основними складовими редактора, зокрема панеллю вузлів, робочим простором та інформаційною панеллю (Рис. 1.2). Детальніше з функціоналом інтерфейсу можна ознайомитися через інструкцію користувача, що доступна на офіційному сайті проєкту.

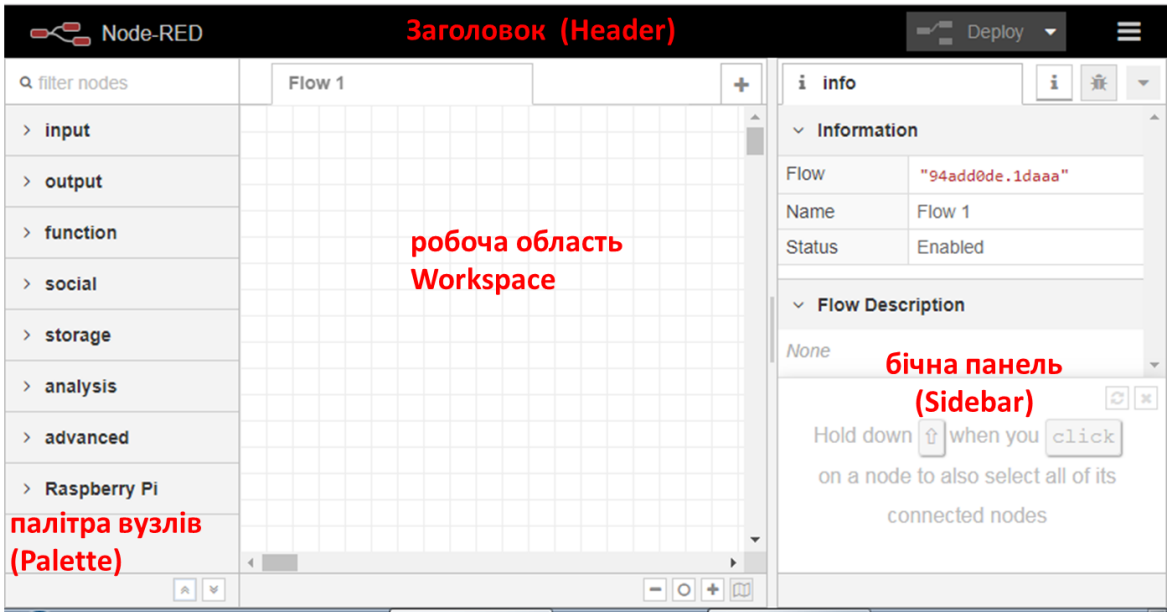

Рисунок 1.2 – Інтерфейс редактора Node-RED

2. У палітрі ліворуч оберіть вузол типу **Inject** з групи Input і вузол **Debug** з

групи Output, після чого перетягніть їх на робочу область. З'єднайте обидва вузли між собою – на виході Inject і на вході Debug з’явиться з’єднання, яке і передаватиме повідомлення. На цьому етапі вузли будуть позначені блакитними кружечками, що вказує на незбережені зміни в потоці (Рис. 1.3).

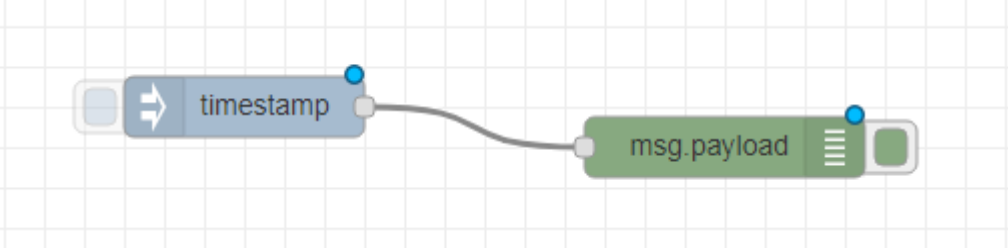

Рисунок 1.3 – Розміщення та з’єднання вузлів Inject і Debug у потоці

3. Для збереження та запуску потоку натисніть у верхній частині екрана Deploy, обравши при цьому пункт Modified Nodes, і підтвердьте розгортання (Рис. 1.4).

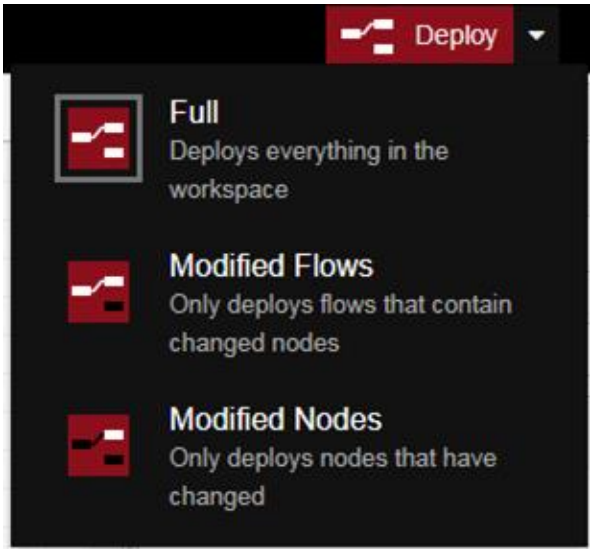

Рисунок 1.4 – Вибір опції Modified Nodes для розгортання зміненого потоку

Після успішного збереження з’явиться відповідне повідомлення, а кольорові індикатори зникнуть (Рис. 1.5).

Рисунок 1.5 – Повідомлення про успішне розгортання потоку

4. Для перевірки роботи програми, відкрийте бічну панель **Debug messages,** натиснувши на іконку з «жуком» (Рис. 1.6).

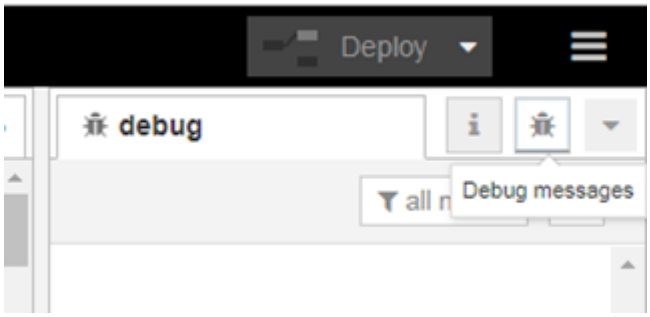

Рисунок 1.6 – Кнопка відкриття панелі налагодження Debug (іконка «жука»)

Після цього натисніть кнопку запуску на вузлі Inject (ліворуч від його назви *timestamp*). В результаті на панелі Debug з’явиться повідомлення про передачу даних (Рис. 1.7).

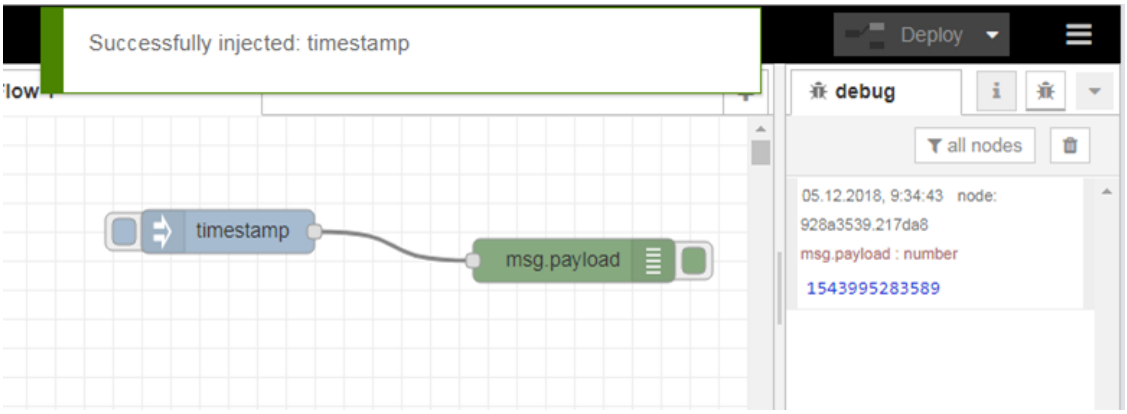

Рисунок 1.7 – Результат виконання Inject у панелі налагодження Debug

У даному прикладі розглядається послідовність виконання програми в середовищі Node-RED. Як правило, обчислення (перерахунок) вузлів ініціюється в момент надходження на їхній вхід повідомлення (*message*). Повідомлення в

середовищі Node-RED являє собою простий об’єкт JavaScript (структуроподібна змінна), який може містити довільний набір властивостей.

У розглядуваній програмі, після спрацювання вузла з назвою **«timestamp»**, формується об’єкт-повідомлення msg, що передається по з’єднувальному дроту до наступного вузла. Основна властивість повідомлення – msg.payload, тобто «корисне навантаження».

Варто зазначити, що вузол «timestamp» не приймає вхідних повідомлень, оскільки виступає ініціатором процесу обчислення. Усі вузли палітри **Input** є ініціаторами запуску потоку. Зокрема, вузол типу **Inject** може бути ініційований вручну (натисканням кнопки) або автоматично з певною періодичністю, заданою у налаштуваннях. Ініціація повідомлення передбачає формування полів об’єкта msg та його передавання іншим вузлам за допомогою з’єднань. Повідомлення, створене вузлом Inject, за замовчуванням має дві властивості – payload (корисне навантаження) та topic (тема). Типово, у поле topic записується мітка часу (timestamp), яка визначає кількість мілісекунд, що минули з 1980 року.

Для відображення повідомлень, які проходять потоком, використовується вузол типу **Debug** з параметром msg.payload. Після отримання повідомлення цей вузол виводить його зміст у вікні налагодження (Debug pane).

5. Відповідно до схеми на рисунку 1.8 відредагуйте властивості вузлів: задайте їм нові імена, визначте значення поля topic, а також встановіть інтервал автоматичного оновлення для вузла Inject. Для відкриття налаштувань будь-якого вузла достатньо двічі клацнути на ньому лівою кнопкою миші.

Після внесення всіх необхідних змін виконайте розгортання програми за допомогою кнопки Deploy та уважно перегляньте повідомлення, що з'являтимуться у вікні налагодження Debug.

Зверніть увагу на структуру та вміст кожного повідомлення — це дозволить краще зрозуміти, як змінилася поведінка потоку після редагування параметрів вузлів.

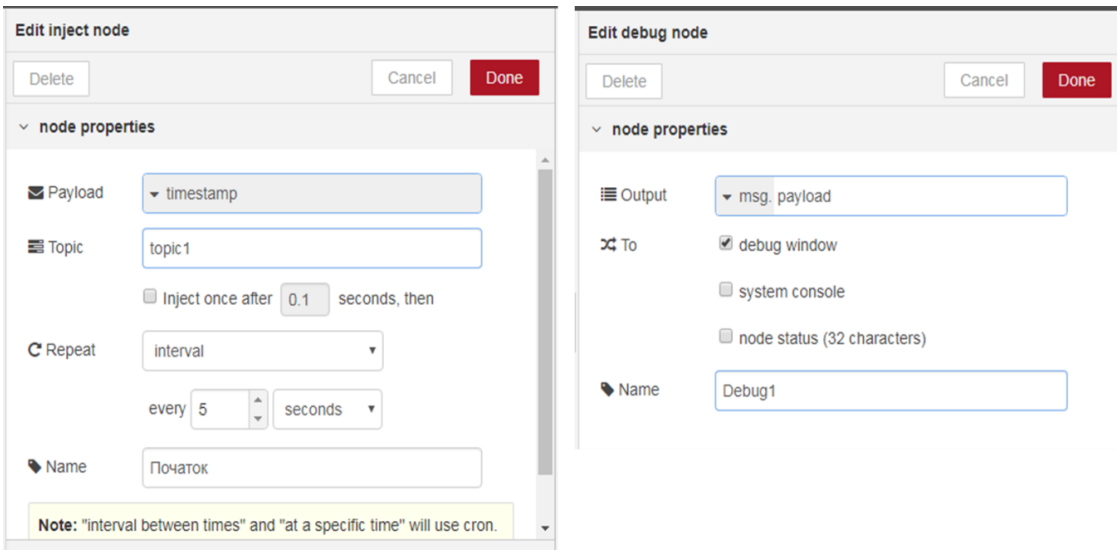

Рисунок 1.8 – Зміна налаштування властивостей вузлів

6. Змініть вузол з іменем «Початок» так, щоб він формував корисне навантаження у вигляді текстового повідомлення «Це текстове повідомлення» (Рисунок 1.9). Розгорніть програму та проаналізуйте, як саме повідомлення відображається у вікні Debug.

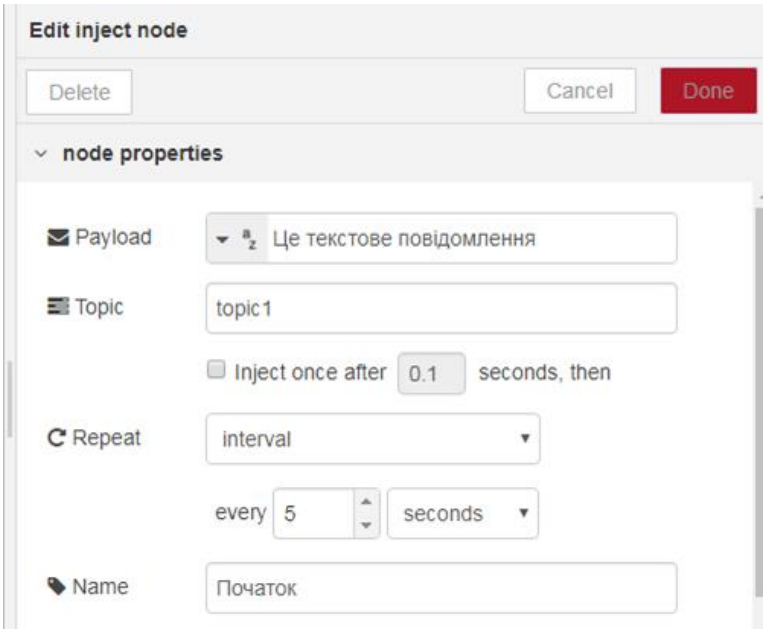

Рисунок 1.9 – Формування корисного навантаження текстом

7. Налаштуйте вузол «Початок» таким чином, щоб він знову формував корисне навантаження у вигляді мітки часу (timestamp). Після розгортання

переконайтеся, що відмітка часу з періодичністю кожні 5 секунд відображається у вікні повідомлень.

Вивчіть призначення вузлів change та delay за офіційною документацією Node-RED. Після ознайомлення доповніть наявну програму відповідними вузлами, затримки та зміни корисного навантаження, згідно зі схемою, наведеною на Рис. 1.10.

Для реалізації затримки використайте вузли:

- delay 1s – пауза тривалістю 1 секунда,
- delay 2s – пауза тривалістю 2 секунди,
- delay 3s – пауза тривалістю 3 секунди,
- delay 4s – пауза тривалістю 4 секунди.

Для послідовного присвоєння текстових значень властивості payload використайте вузли change із правилом set та значеннями:

- set1 – рядок «один»,
- set2 – рядок «два»,
- set3 – рядок «три»,
- set4 – рядок «чотири»,
- set5 – рядок «п'ять».

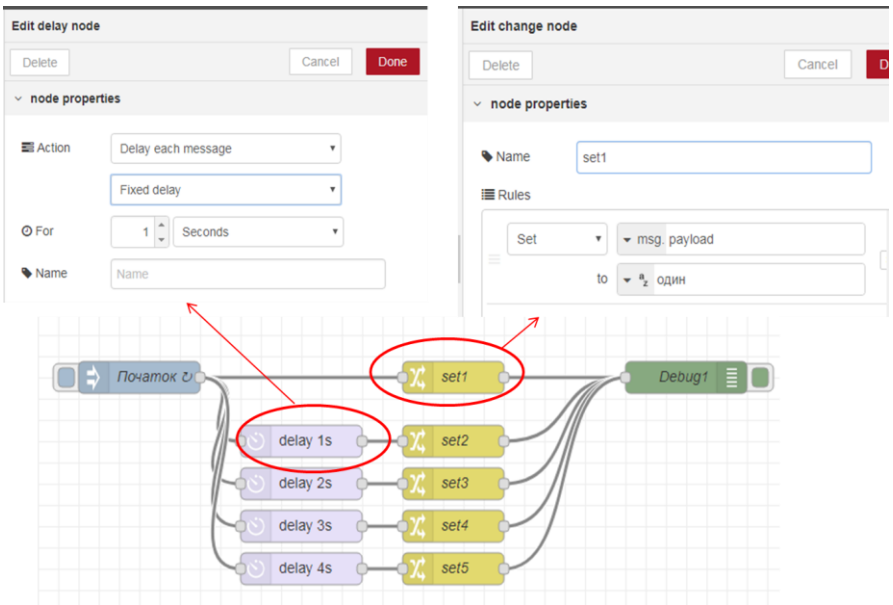

Рисунок 1.10 – Структура затримки та зміни повідомлень від «один» до «п’ять» у Node-RED з використанням вузлів delay та change

Після розгортання переконайтеся, що у вікні повідомлень з інтервалом одна секунда виводяться відповідні значення від «один» до «п’ять».

8. Скористайтеся офіційною документацією Node-RED, щоб детальніше вивчити можливості вузла function. Цей вузол призначений для довільної обробки повідомлень засобами мови JavaScript, що відкриває широкі можливості для перетворення даних у потоці.

Доповніть програму таким чином, щоб часова мітка відображалася у зручному форматі дати та часу. Для реалізації цього скористайтеся вбудованим об'єктом Date і його методом toLocaleString(), який автоматично форматує дату відповідно до локальних налаштувань системи. Після успішного розгортання деактивуйте виведення повідомлень у вузлі «Debug1», переключивши його у режим приховання.

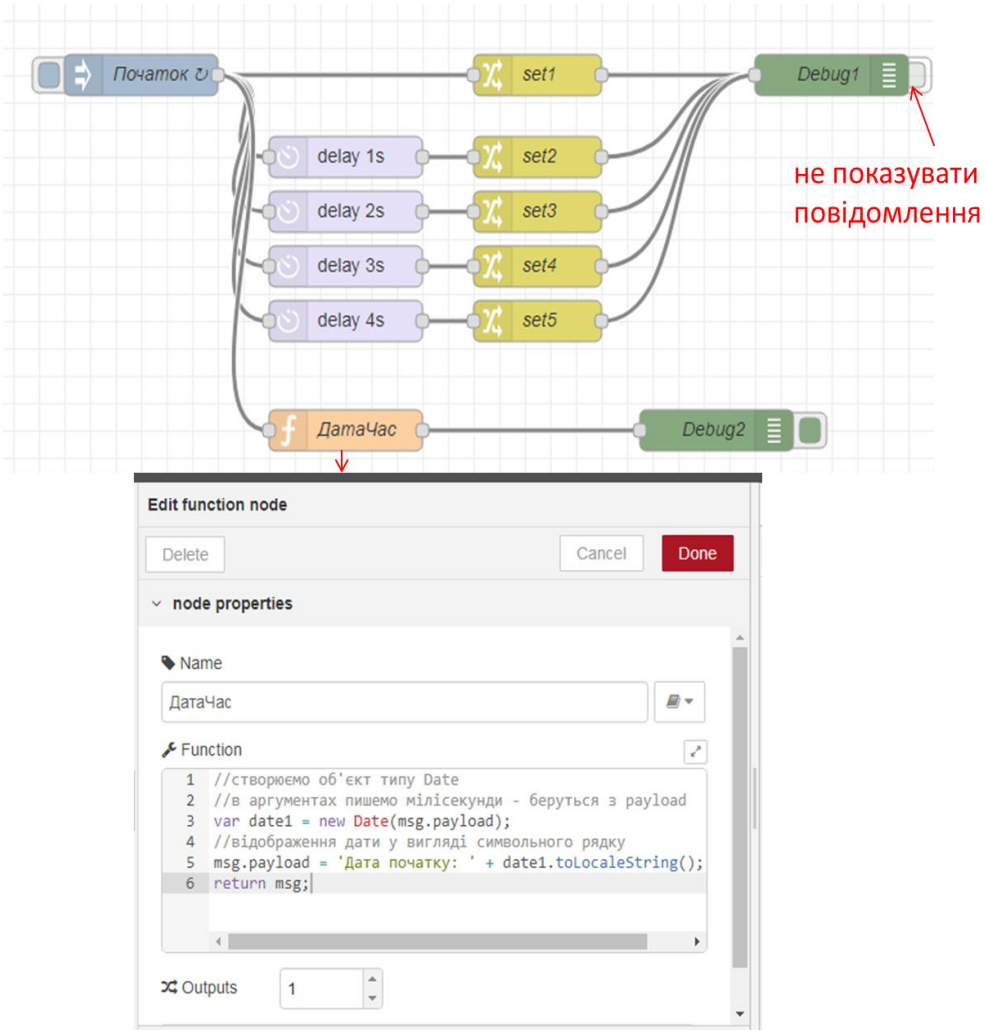

Рисунок 1.11 – Розгортання та переведення вузла “Debug1” у режим приховання повідомлень

Завдання 3. Робота з JS об’єктами та обробка системної інформації

1. Встановлення модуля node-red-contrib-os.

Для забезпечення можливості доступу до системної інформації про операційну систему у середовищі Node-RED необхідно встановити додатковий модуль node-red-contrib-os через інтерфейс Manage Palette. У вкладці Install у полі пошуку потрібно ввести назву модуля node-red-contrib-os, після чого натиснути кнопку Install та підтвердити встановлення у вікні повідомлення (Рис. 1.12).

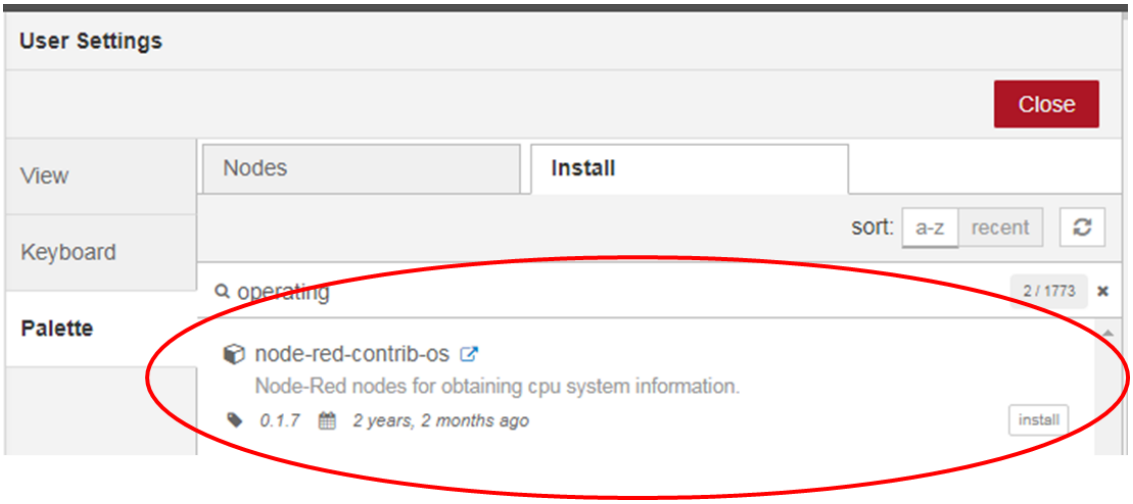

Рисунок 1.12 – Встановлення модуля node-red-contrib-os через Palette Manager

2. Використання вузла Networkintf

Після встановлення модуля node-red-contrib-os у палітрі з’являються нові вузли. Для подальшої роботи потрібно скористатися вузлом **Networkintf**, який виводить інформацію про мережеві інтерфейси комп’ютера. Створіть простий потік, у якому вузол Networkintf передає дані до вузла debug для відображення результатів у відповідному вікні.

Рисунок 1.13 – Потік із використанням вузла Networkintf

Після розгортання потоку та ініціалізації надсилається повідомлення, яке містить JavaScript-об’єкт. Цей об’єкт містить поле NetworkInterfaces, яке є вкладеним об’єктом, що включає перелік усіх доступних мережевих інтерфейсів. Кожен мережевий інтерфейс, у свою чергу, представлений як масив об’єктів, що містять описові параметри відповідних протоколів (IPv4, IPv6 тощо).

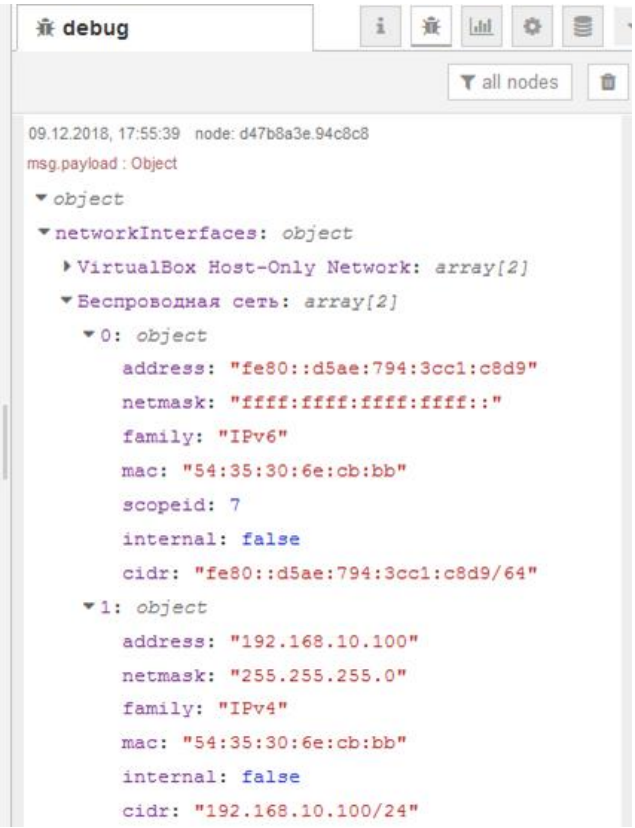

Рисунок 1.14 - Вивід системної інформації про мережеві інтерфейси у форматі JS об’єкта

3. Отримання MAC-адрес

Розробіть програму у середовищі Node-RED, яка зчитує та виводить перелік MAC-адрес мережевих адаптерів, встановлених на вашому комп'ютері. Результат виконання програми має відповідати зразку, наведеному на Рис. 1.15.

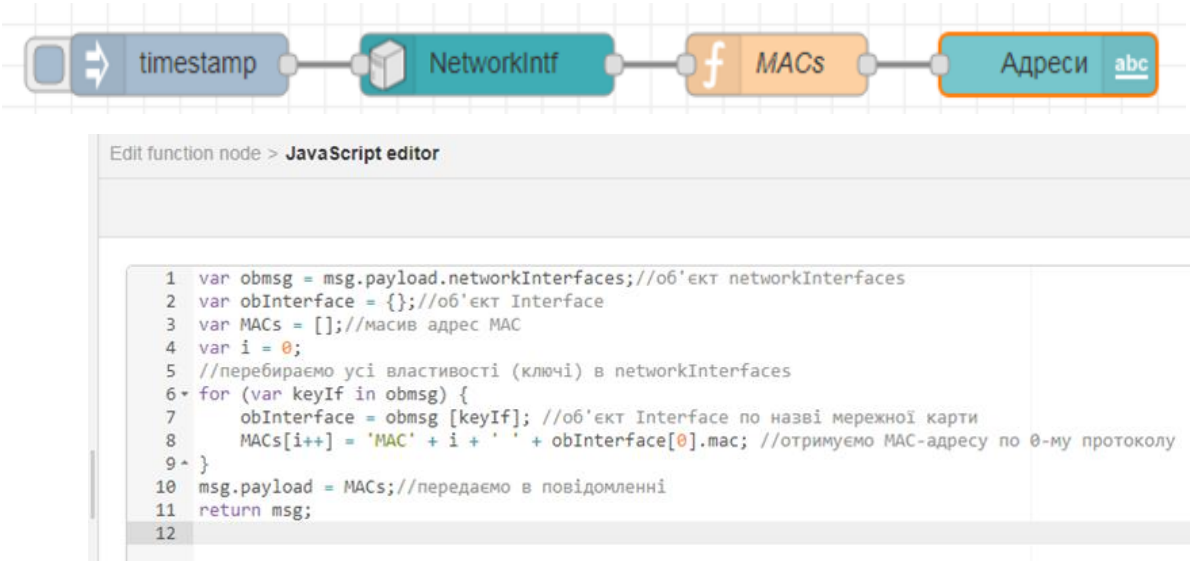

Рисунок 1.15 – Створення програми для виводу MAC-адрес мережевих карт

Контрольні питання

1. Які основні принципи роботи середовища Node-RED?
2. Які типи вузлів (nodes) доступні в Node-RED за замовчуванням?
3. Що таке msg в Node-RED і як із ним працювати?
4. Як у Node-RED організовано обробку подій?
5. Які можливості надає функціональний вузол Function?
6. Як відрізняються Inject та Debug вузли?
7. Що таке потік (flow) у Node-RED і які в нього властивості?
8. Як Node-RED реалізує роботу з JavaScript-об’єктами?
9. Які є способи налагодження програм у Node-RED?
10. Які приклади застосувань Node-RED у мікросистемних технологіях ви можете навести?

ПРАКТИЧНА РОБОТА № 2

Інтеграція Node-RED із зовнішніми сервісами та інтерфейсами в мікросистемних технологіях

Мета роботи. Навчитися інтегрувати Node-RED з промисловими протоколами (Modbus), електронною поштою та засобами візуалізації даних. Розвинути практичні вміння побудови елементів користувацького інтерфейсу (Dashboard) для виводу інформації в реальному часі в межах мікросистемних технологій.

Теоретичні відомості

Node-RED є середовищем візуального програмування, яке широко застосовується у сфері мікросистемних технологій та Інтернету речей (IoT). Його ключова роль полягає у забезпеченні інтеграції між різноманітними сервісами, пристроями та користувацькими інтерфейсами. Завдяки блочному підходу, Node-RED дає змогу створювати складні логічні схеми обробки даних без необхідності глибокого занурення у програмування, що суттєво скорочує час розробки IoT-рішень.

Важливою перевагою Node-RED є підтримка широкого спектра протоколів обміну даними, що охоплюють як загальні мережеві сервіси (HTTP, MQTT, WebSocket, електронна пошта), так і спеціалізовані промислові стандарти (Modbus TCP/RTU, OPC UA тощо). Це робить платформу універсальним інструментом для побудови мікросистем моніторингу, керування та автоматизації.

Особливу увагу в інтеграційних завданнях приділяють таким сервісам:

- Електронна пошта (SMTP/IMAP/POP3), яка використовується для нотифікації користувачів, передачі системних звітів або сигналів тривоги. Включення поштових вузлів у потоки Node-RED дозволяє автоматизувати сповіщення про стан системи, що підвищує її надійність і зручність використання.

• Промислові протоколи, зокрема Modbus TCP, які залишаються стандартом де-факто у системах автоматизації. Взаємодія з датчиками, контролерами, приводами чи шлюзами на основі Modbus забезпечує двосторонню комунікацію між апаратними пристроями та програмними сервісами Node-RED. Це створює умови для реалізації реальних задач керування виробничим обладнанням та моніторингу технологічних процесів.

• Не менш важливим компонентом інтеграційних рішень є побудова інтерфейсів візуалізації. Для цього Node-RED пропонує модуль Dashboard, який дозволяє створювати графічні інтерфейси користувача безпосередньо у веб-браузері. Dashboard-інструменти забезпечують відображення параметрів системи у вигляді діаграм, графіків і покажчиків, а також дають змогу реалізувати елементи керування (кнопки, слайдери, перемикачі). Завдяки цьому формується основа для організації дистанційного моніторингу, оперативної діагностики та інтерактивного керування мікросистемами.

Таким чином, практичні навички роботи з поштовими сервісами, промисловими протоколами (Modbus TCP) та інструментами візуалізації (Node-RED Dashboard) є важливою складовою підготовки фахівців у галузі Інтернету речей. Вони дозволяють реалізовувати комплексні мікросистемні рішення, що поєднують апаратні та програмні компоненти, забезпечуючи гнучкість, масштабованість і надійність сучасних IoT-платформ.

Завдання до практичної роботи

1. Виконати роботу з поштою.
2. Виконати роботу з Modbus.
3. Підключити та ознайомитися з модулем node-red-dashboard

Хід роботи

Завдання 1. Робота з поштою

1. Перед початком роботи з електронною поштою в середовищі Node-RED потрібно з'ясувати параметри поштового сервера, що використовуватиметься

для надсилання та отримання листів. З метою безпеки та зручності тестування рекомендується зареєструвати окремий обліковий запис на одному з доступних поштових сервісів, не використовуючи основну робочу адресу.

Налаштування для відправлення пошти (вихідна пошта):

- SMTP Server: вказати відповідно до обраного сервісу.
- Port: 465 / 587 (залежно від сервісу та протоколу захисту).

Налаштування для отримання пошти (вхідна пошта):

- POP3 Server або IMAP Server: вказати відповідно до обраного сервісу.
- Port: 993 (IMAP) / 995 (POP3).

Інформація про конфігурацію серверів наводиться в офіційних довідниках поштових сервісів.

Поштовий сервіс	Для відправки Сервер вихідних повідомлень		Для отримання Сервер вхідних повідомлень		Примітка
www.ukr.net	SMTP Server	smtp.ukr.net	IMAP Server	imap.ukr.net	В налаштуваннях треба увімкнути IMAP/SMTP (рис.27)
	port	465 або 2525	port	993	
www.gmail.com	SMTP Server	smtp.gmail.com	IMAP Server	imap.gmail.com Вимагає SSL: так	В налаштуваннях треба увімкнути IMAP (рис.28) Активувати доступ до додатків https://myaccount.google.com/lesssecureapps (рис.29)
	port	465(SSL) або 587(TLS)	port	993	

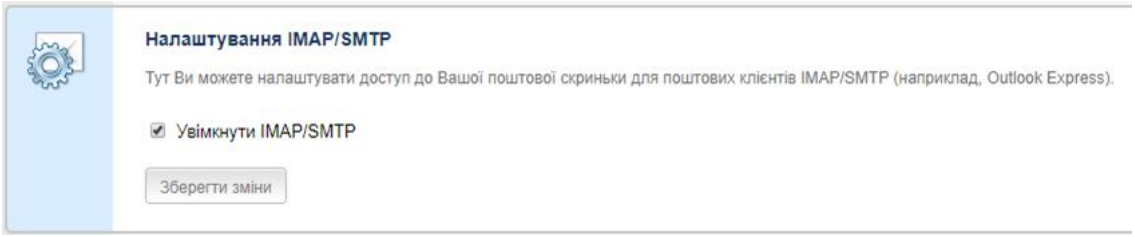

Рисунок 2.1 – Приклад інформації про налаштування поштових сервісів

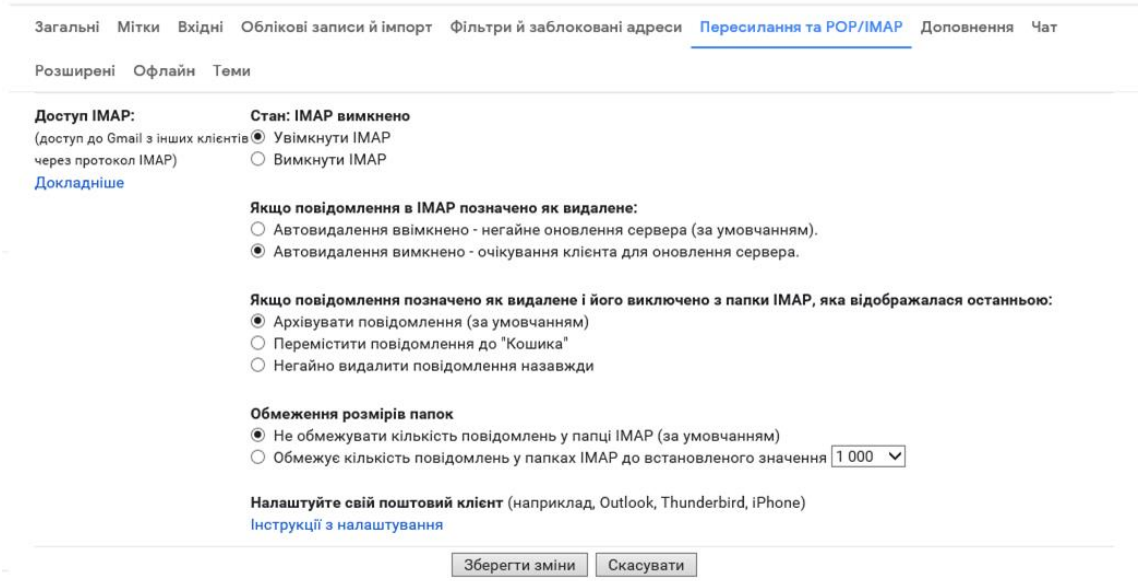

Рисунок 2.2 – Приклад взаємодії з поштовими клієнтами через IMAP/SMTP

У разі використання сервісу Gmail разом із Node-RED може виникнути ситуація, коли навіть після успішного увімкнення протоколів IMAP та SMTP система відображатиме повідомлення про блокування з'єднання. Щоб усунути цю проблему, тимчасово надайте дозвіл на доступ до облікового запису з боку менш захищених застосунків у налаштуваннях безпеки Google.

2. Доповніть наявну програму фрагментом, зображеним на Рис. 2.3.

Конфігурацію вузлів виконайте відповідно до Рис. 2.4: у поле user-id введіть адресу електронної пошти, у поле password — відповідний пароль облікового запису, а в поле адресата вкажіть власну поштову скриньку. Усі інші параметри налаштуйте згідно з даними, визначеними в першому пункті цього завдання.

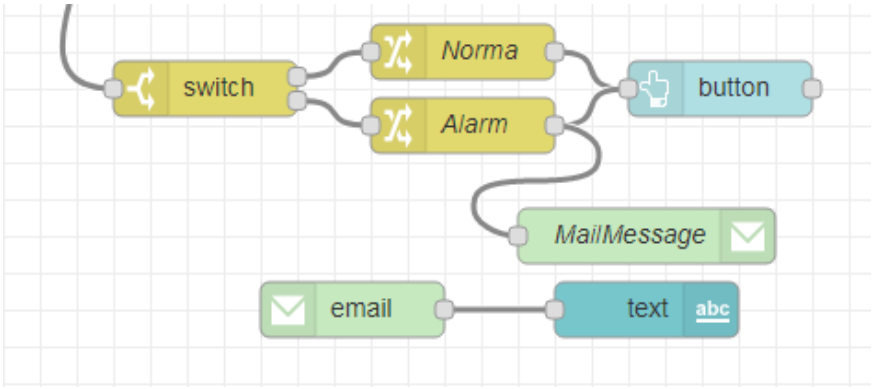

Рисунок 2.3 – Модифікація програми для надсилання електронних повідомлень

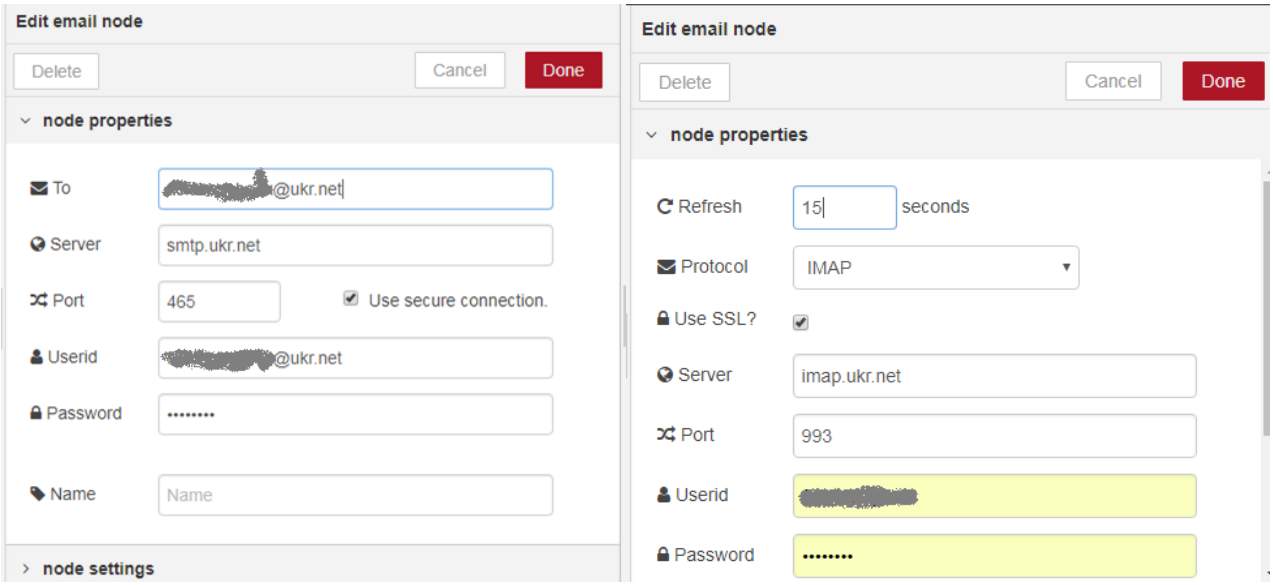

Рисунок 2.4 – Налаштування та приклад введення параметрів вузлів у Node-RED

Після розгортання проєкту необхідно встановити значення температури, що викликає спрацювання події Alarm. Повідомлення має надійти на пошту, а також протягом 15 секунд відобразитися на веб-сторінці у відповідному полі (Рис. 2.5).

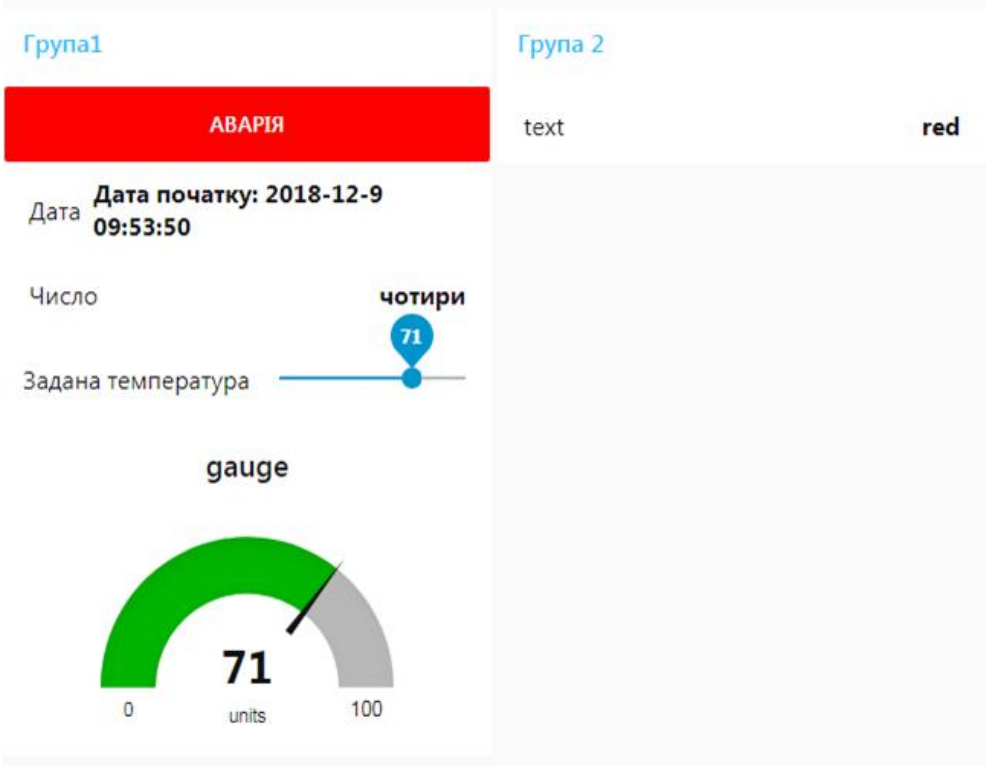

Рисунок 2.5 – Відображення отриманого повідомлення на веб-сторінці

Завдання 2. Робота з Modbus

Node-RED, як правило, використовується або на рівні Edge-пристроїв, або як хмарний додаток. У разі використання на Edge-платформі (наприклад, Raspberry Pi), Node-RED може виступати у ролі концентратора, шлюзу або маршрутизатора, що забезпечує збір та обробку даних від пристроїв через промислові протоколи.

Одним з найпоширеніших протоколів у цій сфері є **Modbus TCP/IP**, завдяки простоті реалізації та підтримці великою кількістю пристроїв. У спільноті Node-RED доступні бібліотеки, що забезпечують інтеграцію з пристроями за протоколом Modbus. Raspberry Pi у такій конфігурації може виступати посередником між промисловими пристроями та хмарними сервісами (Рис. 2.6).

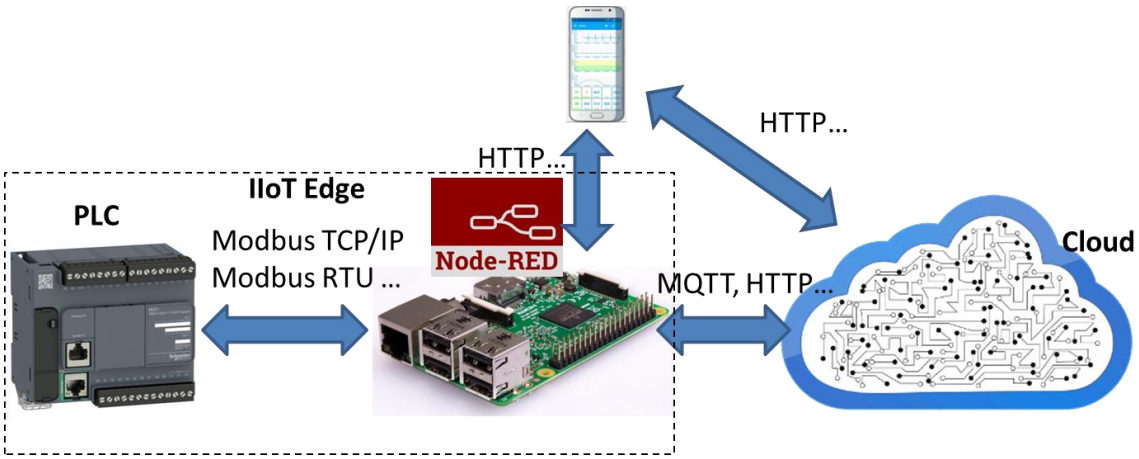

Рисунок 2.6 – Схема взаємодії Node-RED з хмарними сервісами та пристроями Modbus

Для тестування системи на початковому етапі доцільно використовувати лише програмні компоненти. Замість фізичного програмованого логічного контролера (ПЛК) можна застосувати його імітатор, що підтримує протокол Modbus TCP/IP, а апаратну платформу Raspberry Pi – замінити на віртуальну машину з операційною системою Raspbian та попередньо встановленим програмним забезпеченням.

Ще простіше – обмежитися використанням лише Node-RED, який буде

безпосередньо підключатися до імітатора ПЛК або до Modbus TCP/IP Server (Рис. 2.7).

У даній практичній роботі передбачено використання програмного пакета Mod_RSsim як імітатора ПЛК, а також бібліотеки node-red-contrib-modbustcp для взаємодії з протоколом Modbus у середовищі Node-RED.

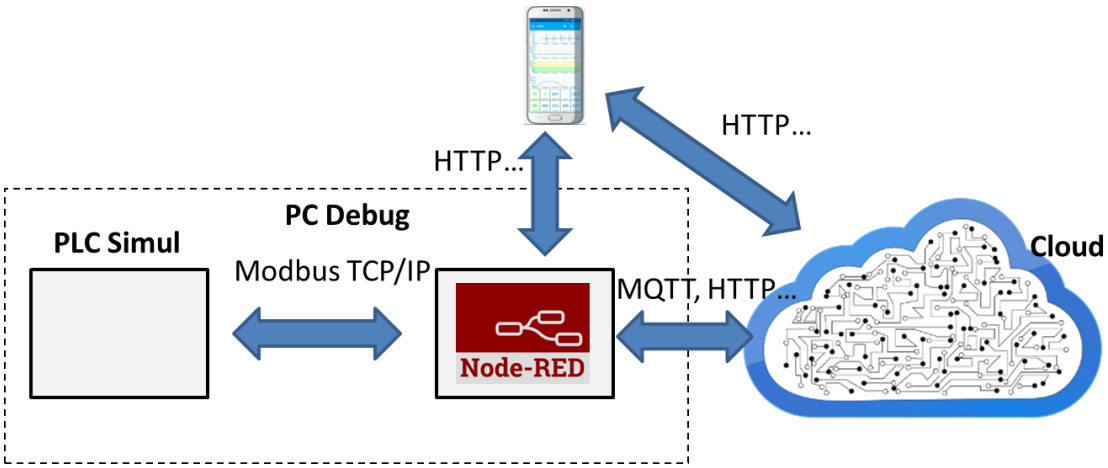

Рисунок 2.7 – Застосування Node-RED для встановлення з'єднання з програмним імітатором програмованого логічного контролера або імітатором сервера Modbus TCP/IP

1. Для початку необхідно встановити пакет node-red-contrib-modbustcp, який забезпечує підтримку протоколу Modbus у середовищі Node-RED. Це робиться за допомогою меню Manage Palette, де у вкладці Install в поле пошуку вводиться назва пакету, після чого виконується інсталяція (Рис. 2.8).

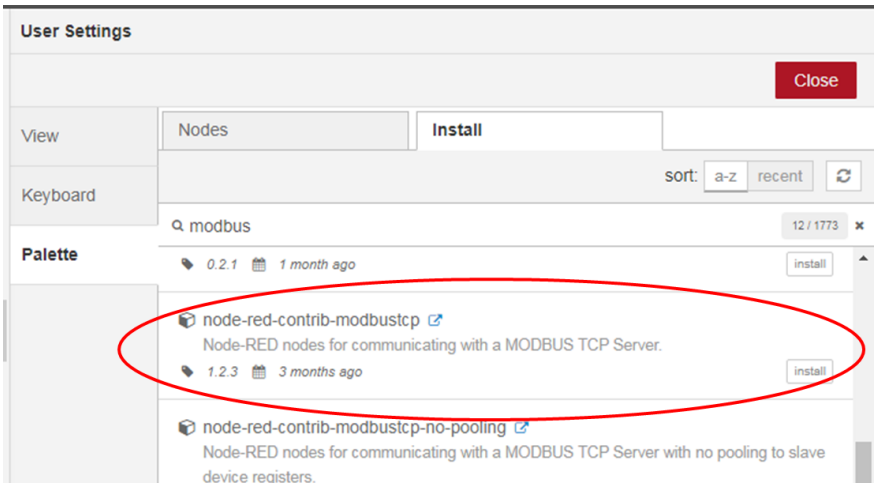

Рисунок 2.8 – Встановлення пакету node-red-contrib-modbustcp через інтерфейс Manage Palette

2. У якості програмного імітатора програмованого логічного контролера (ПЛК) у цій роботі використовується програмний пакет Mod_RSsim, що підтримує протокол Modbus TCP. Для його використання необхідно:

– завантажити інсталяційний файл з офіційного сайту: http://www.plcsimulator.org/downloads/SimSetup.msi?attredirects=0;
– завантажити файл ключа реєстру для налаштування параметрів доступу: http://www.plcsimulator.org/downloads/Vista_key.reg?attredirects=0;
– запустити програму Mod_RSsim з каталогу C:\Program Files (x86)\EmbeddedIntelligence\Mod_RSsim.

У вікні налаштувань необхідно встановити значення Prot: Modbus TCP, що показано на Рис. 2.9.

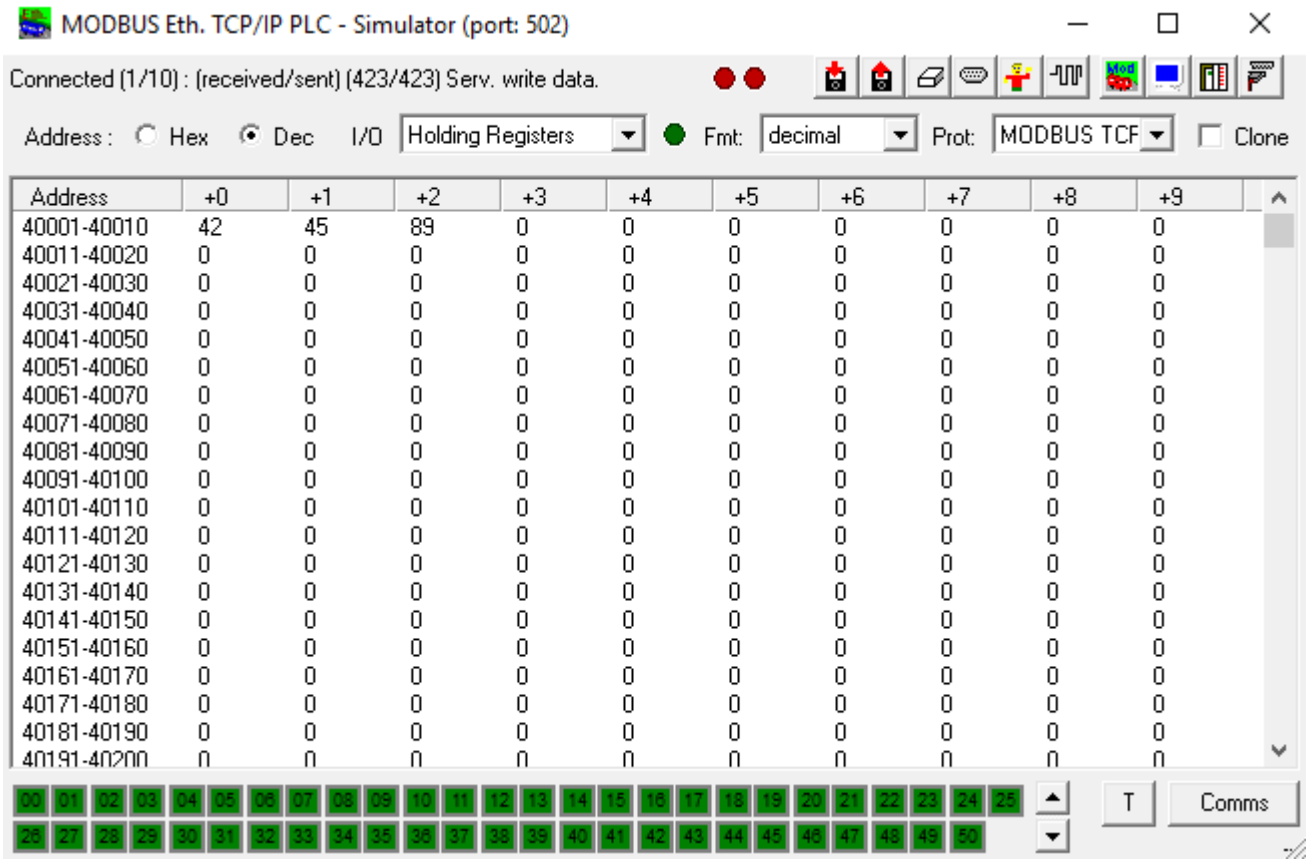

Рисунок 2.9 – Встановлення протоколу обміну в Mod_RSsim на Modbus TCP

3. Для ефективного використання вузлів бібліотеки node-red-contrib-modbustcp рекомендується ознайомитися з офіційною документацією за адресою: https://flows.nodered.org/node/node-red-contrib-modbustcp

4. Створення з’єднання та зчитування даних з регістрів

У палітрі вузлів у розділі Inputs виберіть елемент modbustcp-read та перейдіть до його налаштувань. У правій частині поля Server клацніть на кнопку з іконкою олівця, що відкриє форму для налаштування нового серверного з'єднання (Рис. 2.10).

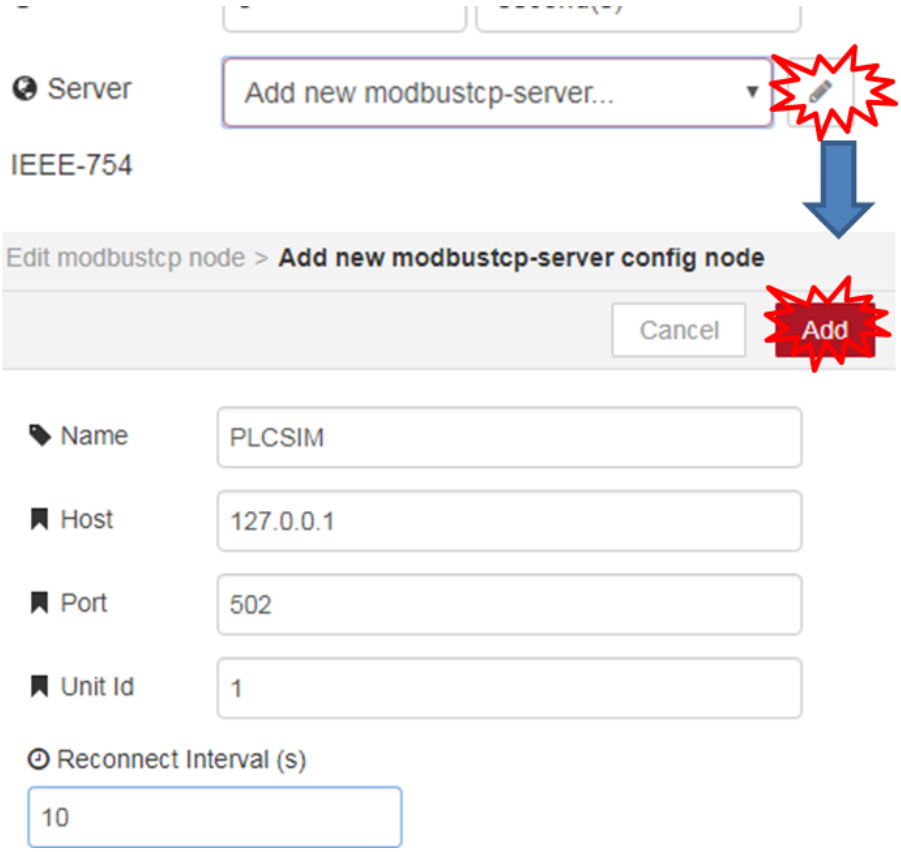

Рисунок 2.10 – Діалогове вікно створення нового серверного підключення у вузлі modbustcp-read

Після створення з’єднання налаштуйте параметри зчитування – прочитати 10 регістрів типу Holding Register, починаючи з адреси 0. Приклад конфігурації наведено на Рис. 2.11.

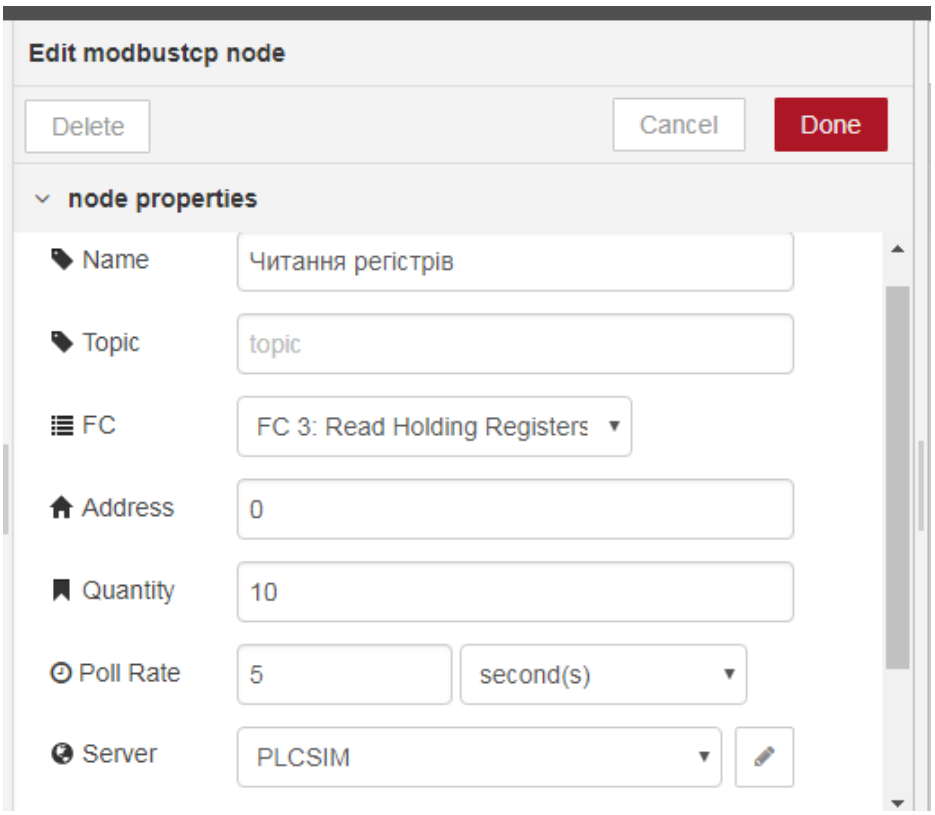

Рисунок 2.11 – Налаштування вузла modbustcp-read для зчитування десяти Holding-регістрів з адреси 0

Додайте вузол debug для виводу отриманих даних та з'єднайте його з вузлом modbustcp-read. Здійсніть розгортання (Deploy) потоку. Для зручності аналізу залиште активним лише останній вузол debug (Рис. 2.12).

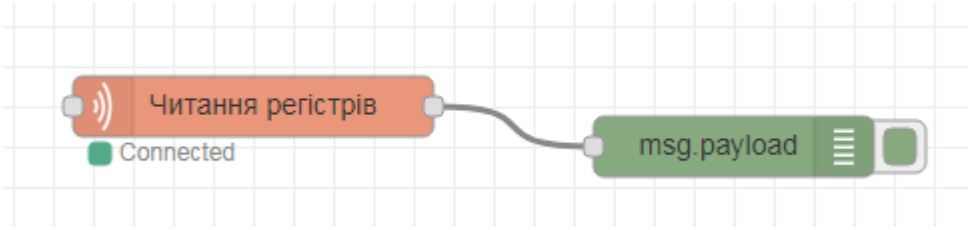

Рисунок 2.12 – Фрагмент потоку Node-RED для зчитування регістрів з Mod_RSsim

Відредагуйте значення перших десяти регістрів безпосередньо у вікні програми Mod_RSsim. Після цього у панелі Debug середовища Node-RED має з'явитися масив із відповідними числовими значеннями.

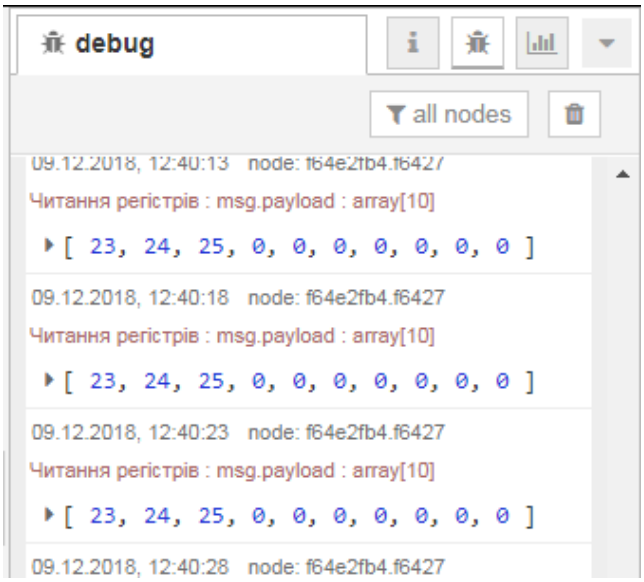

Рисунок 2.13 – Відображення результату зчитування у вікні Debug: масив значень регістрів

Зверніть увагу на те, що властивість msg.payload у даному випадку являє собою масив типу Array, що містить десять елементів. Перш ніж використовувати ці дані для подальших цілей, наприклад, для відображення у веб-інтерфейсі, необхідно виконати їх попередню обробку за допомогою відповідних функцій перетворення.

5. Внесіть зміни до програми відповідно до структурних схем, наведених на рисунках 2.14 – 2.16. Кожна ітерація змін демонструє послідовне ускладнення логіки або доповнення інтерфейсу.

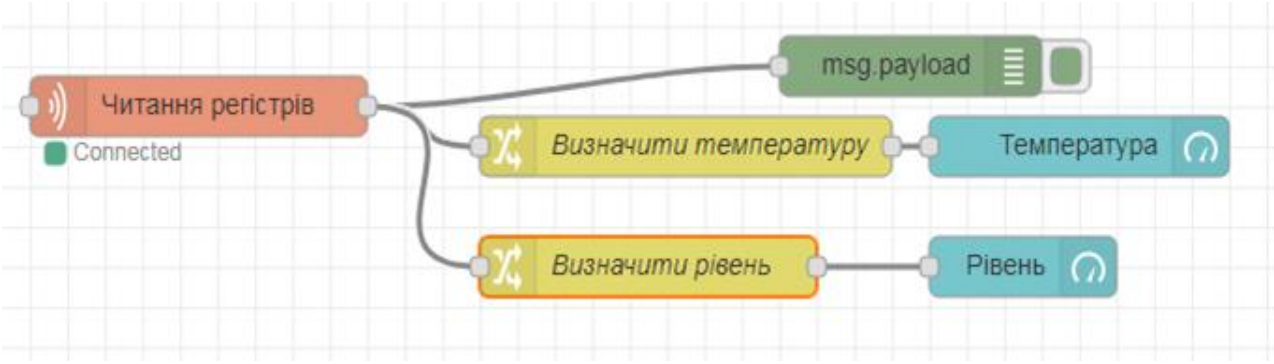

Рисунок 2.14 – Модифікації програми

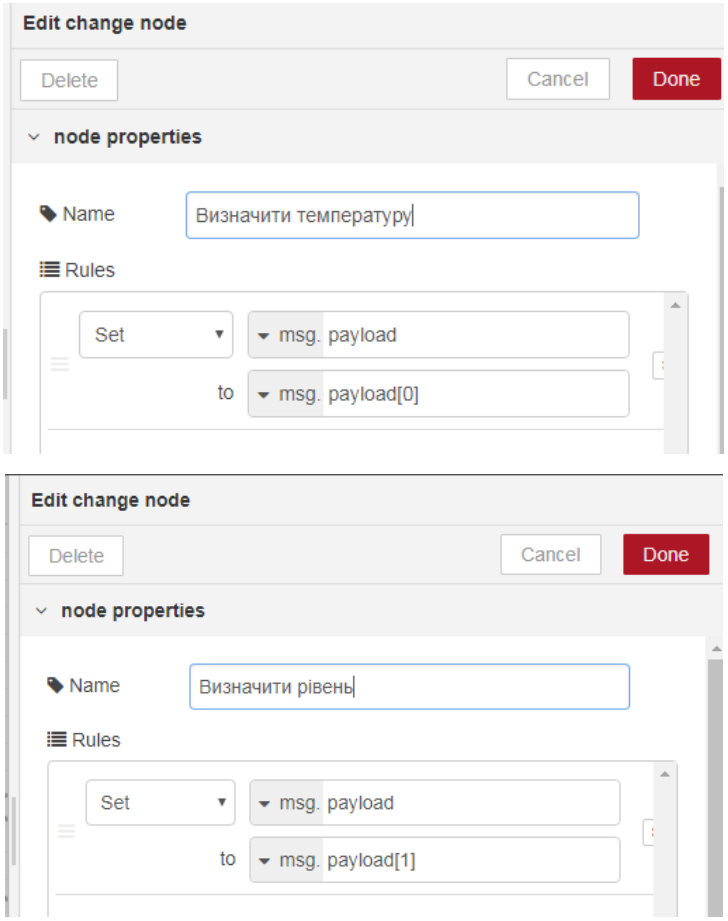

Рисунок 2.15 – Додавання вузла обробки температури та вузла обробки значень рівня

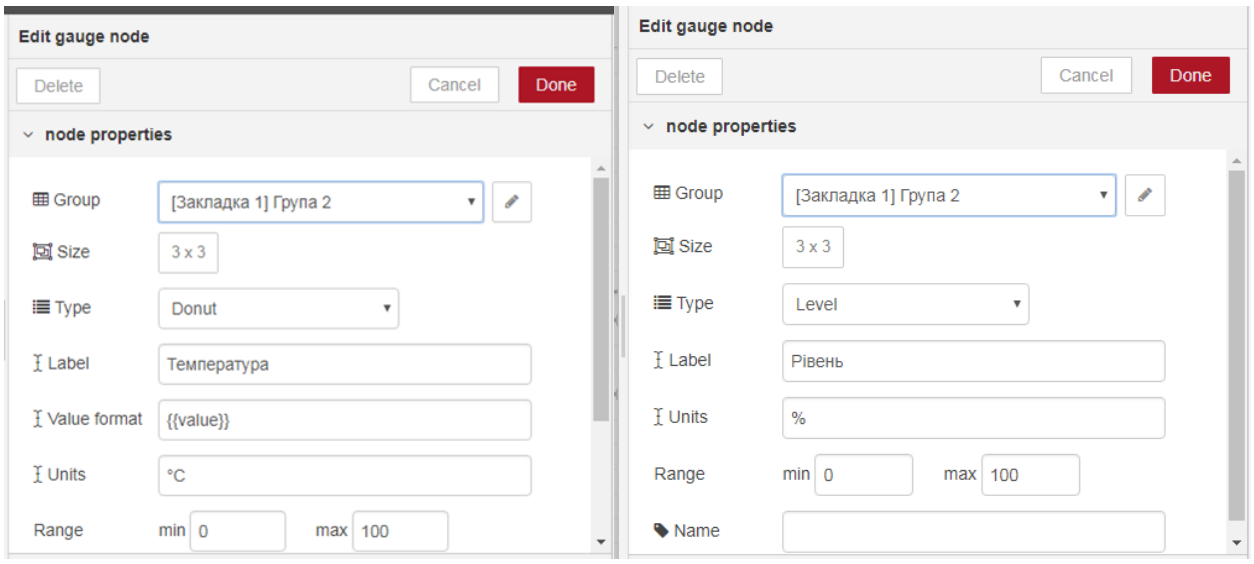

Рисунок 2.16 – Налаштування відображення температури на круговій діаграмі та рівня на лінійній шкалі

6. Запис значень у Holding-регістр

Для запису значень у Holding-регістр слід скористатись вузлом modbustcp-write з розділу Outputs. Створіть відповідний потік згідно зі схемою, поданою на Рис. 2.17. Після успішного розгортання проєкту відкоригуйте значення параметра «Задана температура» у веб-інтерфейсі та переконайтеся, що зміна коректно відображається у відповідному регістрі програми Mod_RSsim.

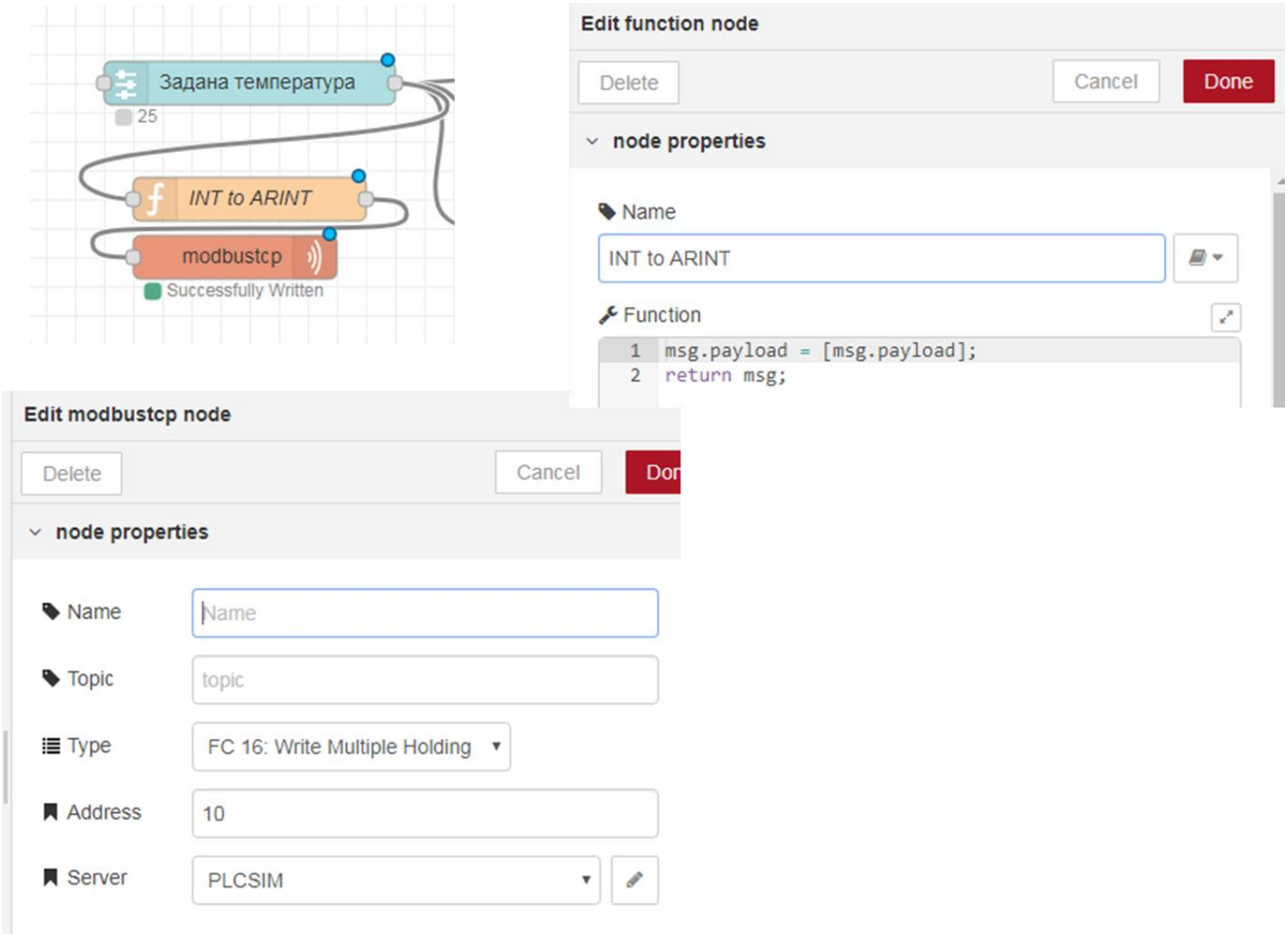

Рисунок 2.17 - Приклад реалізації запису значення у Holding-регістр через вузол modbustcp-write

7. Використання імітатора.

У якості альтернативи програмному пакету **Mod_RSsim**, можна використовувати інші емулятори ПЛК. Наприклад:

– емулятор Unity PRO з демонстраційним проєктом, за посиланням: https://drive.google.com/open?id=0B2FfwwwweBSVRWw4eDVreE1Lb1k;

– емулятор Unity PRO або M221 з проєктом керування роботизованою установкою: http://www.iasu-nuft.pp.ua/virtualnij-trenazer-projectprogrammer

Завдання 3. Підключення та ознайомлення з модулем node-red-dashboard

Node-RED підтримує можливість інсталяції та оновлення палітри вузлів. Для цього використовується інструмент Manage Palette.

На вкладці Install у полі фільтра введіть node-red-dashboard та виконайте наступні дії:

- натисніть кнопку Install;
- підтвердіть інсталяцію у вікні повідомлення;
- після завершення інсталяції закрийте вікно керування палітрою.

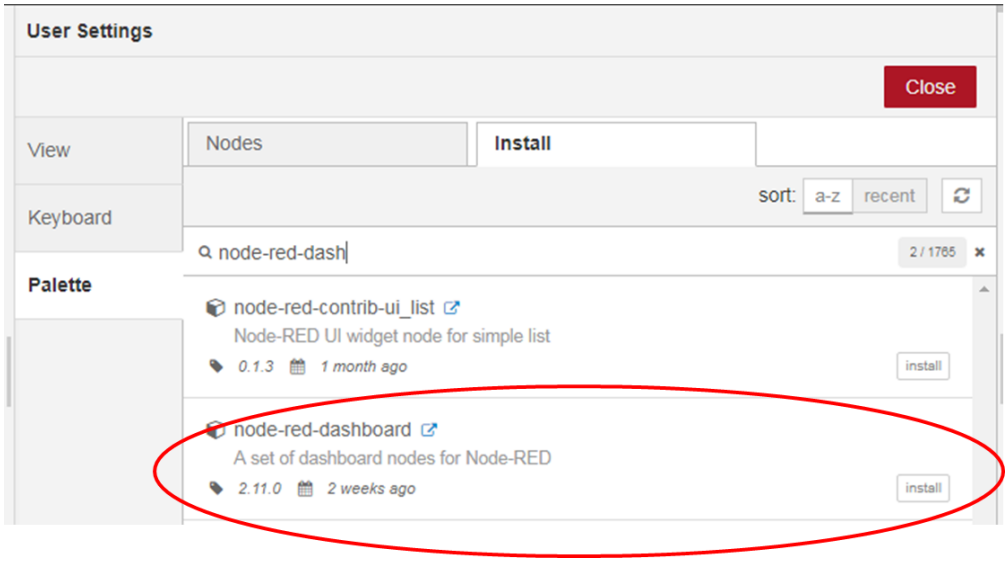

Рисунок 2.18 – Встановлення модуля node-red-dashboard через вкладку Install

Перевірте, чи з’явився в палітрі розділ Dashboard.

1. Додавання закладок у Dashboard Layout

Після встановлення модуля node-red-dashboard, у бічній панелі з’явиться нова іконка з зображенням діаграми – натисніть на неї (Рис. 2.19).

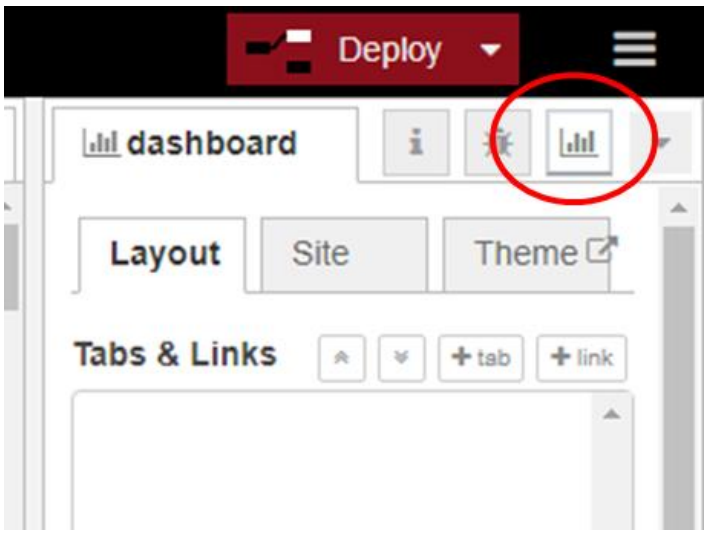

Рисунок 2.19 – Іконка з зображенням діаграми для доступу до Dashboard Layout

У вікні Layout створіть дві закладки (tab). Назвіть першу закладку власним прізвищем та ім'ям (наприклад, «Іваненко Іван»). Додайте другу закладку з довільною назвою.

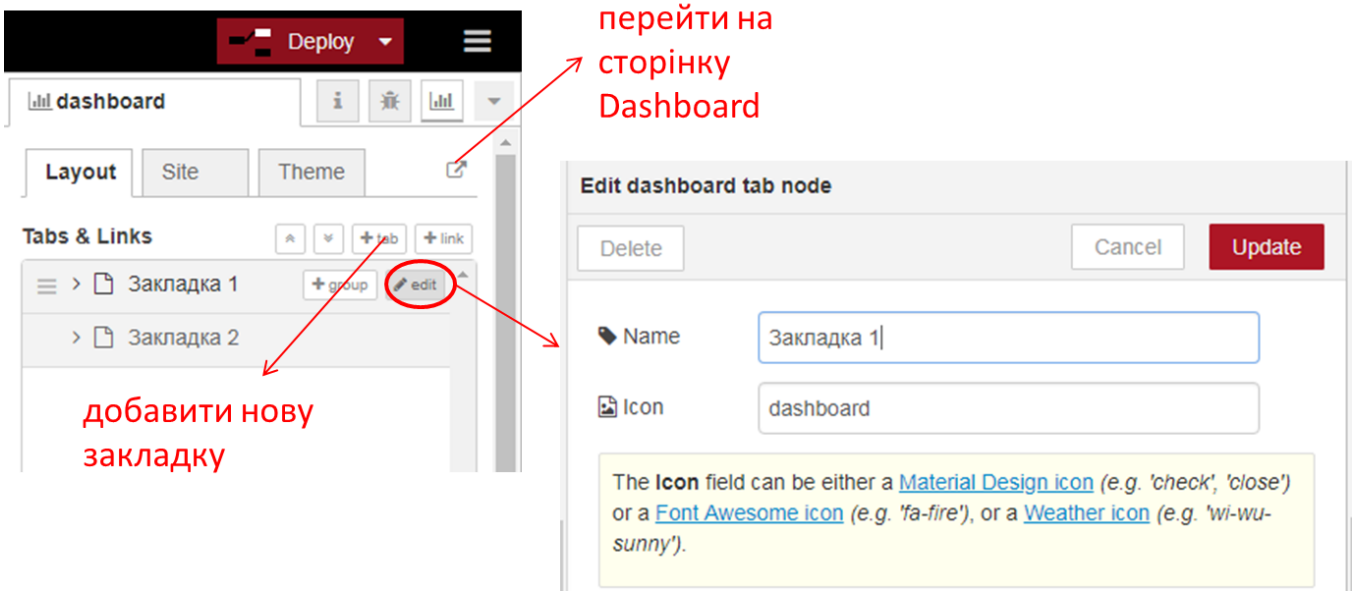

Рисунок 2.20 – Створення нових закладок у Dashboard Layout

3. Вивід дати і часу у Dashboard.

Модифікуйте програму, додавши вузол типу dashboard → text. Під'єднайте його до вузла, який генерує дату і час (наприклад, timestamp). У параметрах вузла

створіть групу з назвою, яка відповідає вашому прізвищу (наприклад, «Іваненко»). Створіть також ще одну групу з назвою, яка відповідає вашому імені (наприклад, «Іван»).

Після всіх налаштувань виконайте розгортання потоку. Для перегляду результату відкрийте веб-інтерфейс Dashboard, натиснувши кнопку переходу (Рис. 2.21) або ввівши в адресному рядку браузера: http://127.0.0.1:1880/ui

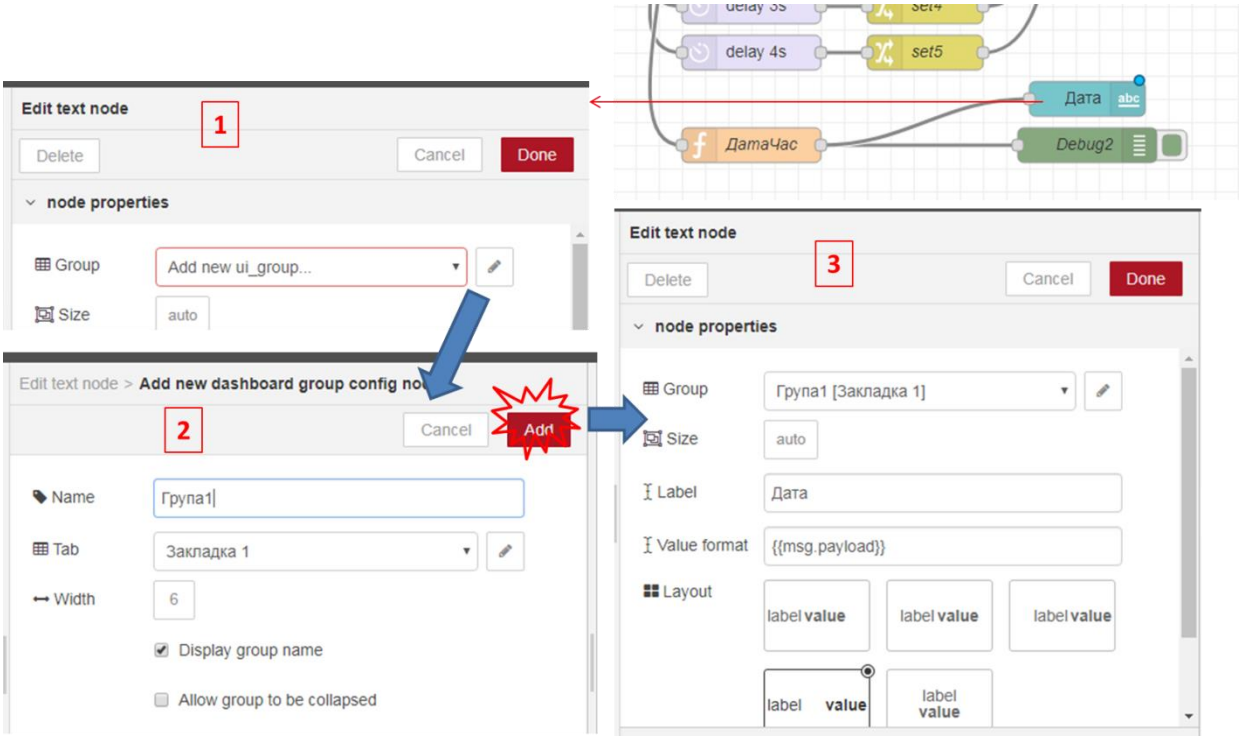

Рисунок 2.21 – Налаштування вузла text для виводу дати й часу у Dashboard

У результаті виконання налаштувань на вкладці веб-інтерфейсу Dashboard має відображатися елемент з поточними датою та часом (Рис. 2.22).

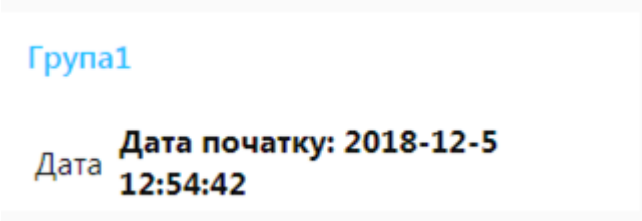

Рисунок 2.22 – Відображення дати й часу у веб-інтерфейсі Dashboard

4. Відображення числа прописом

Аналогічним чином реалізуйте ще один вузол типу text, який відображає числове значення прописом. Під’єднайте його до відповідного вузла обробки числових даних (Рис. 2.23).

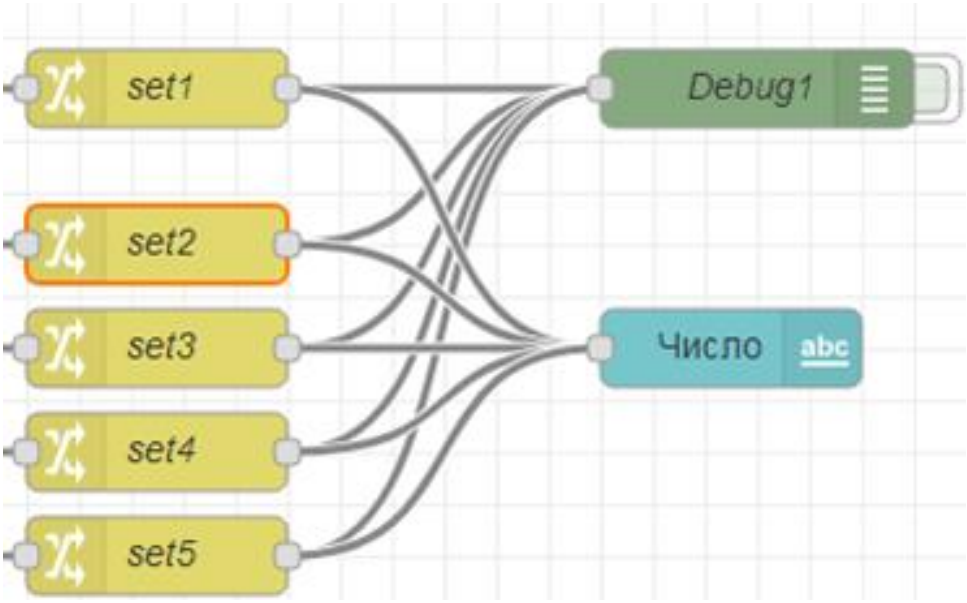

Рисунок 2.23 – Відображення числа у вигляді тексту у Dashboard

5. Додавання логіки порівняння

Додайте до програми фрагмент, як показано на Рис. 2.24. Для реалізації логіки порівняння значень використовуйте вузол switch, а також вузли change, які формують текстові повідомлення та змінюють колір елементів Dashboard.

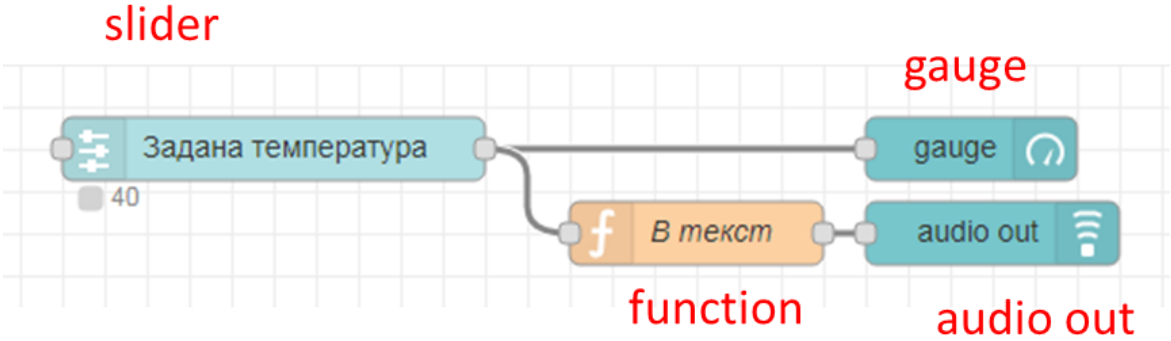

Рисунок 2.24 – Схема додавання вузлів для логіки порівняння значень

Налаштування вузлів, які входять до складу цього фрагменту програми, зображено на Рис. 2.25-2.27.

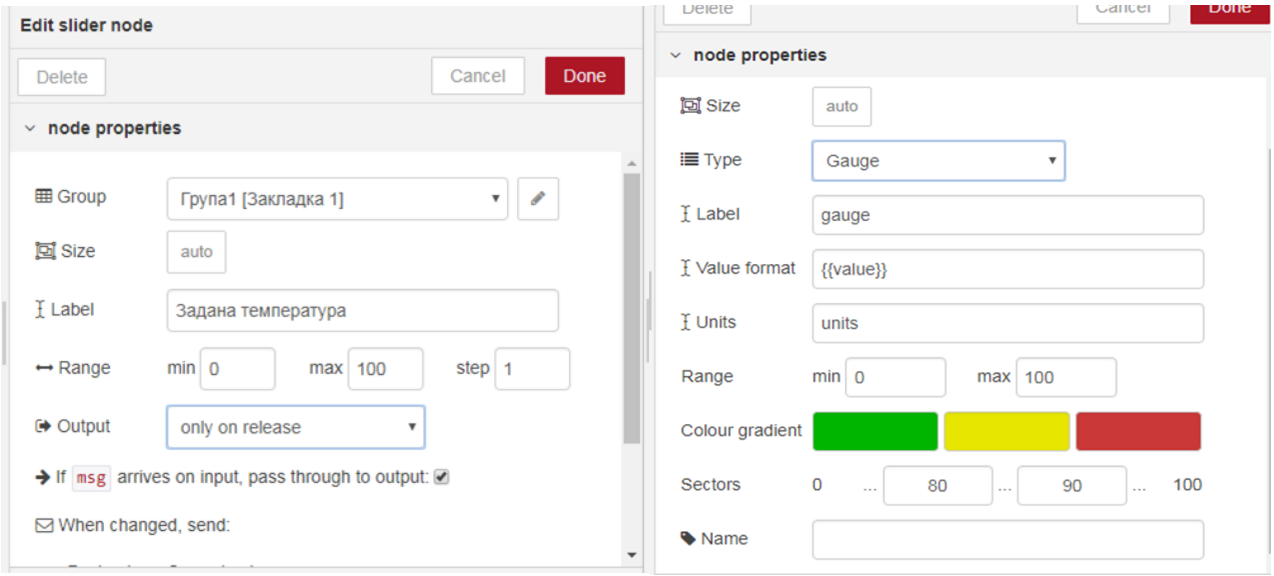

Рисунок 2.25 – Налаштування вузла Slider для регулювання температури та вузла Gauge з градієнтом для відображення значень

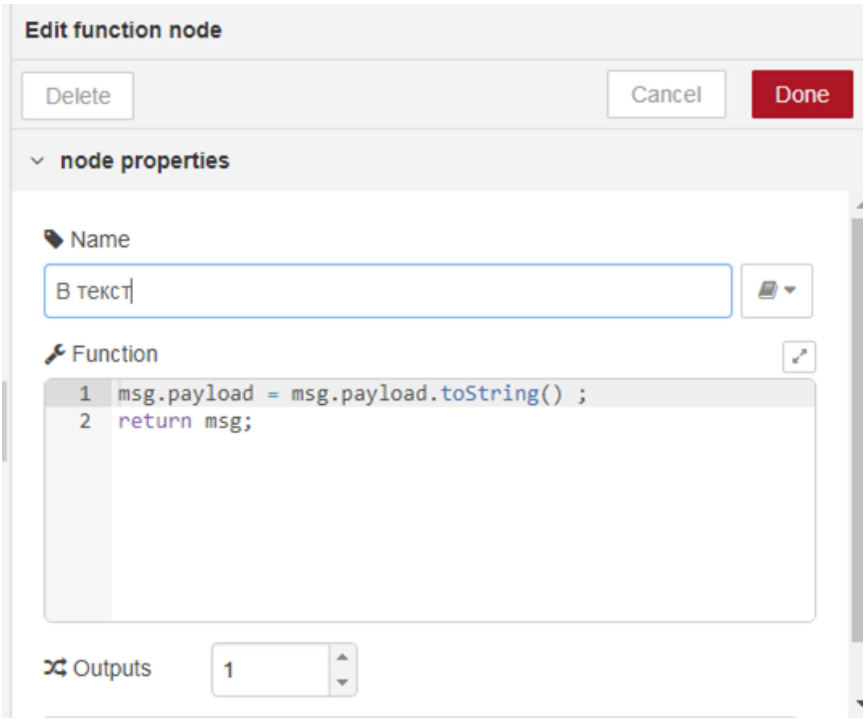

Рисунок 2.26 – Налаштування вузла функції для конвертації даних

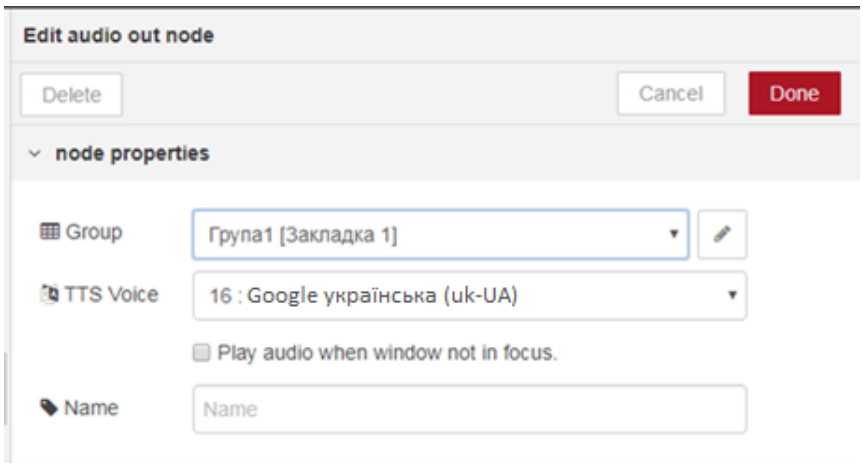

Рисунок 2.27– Налаштування вузла аудіовиходу з TTS голосом

6.Ознайомлення з принципами роботи вузла switch

Модифікуйте програму відповідно до прикладу, що подано нижче. Вузол switch забезпечує умовну обробку вхідного значення. Залежно від виконаної умови, повідомлення буде передане на один із виходів, де воно обробляється відповідними вузлами change.

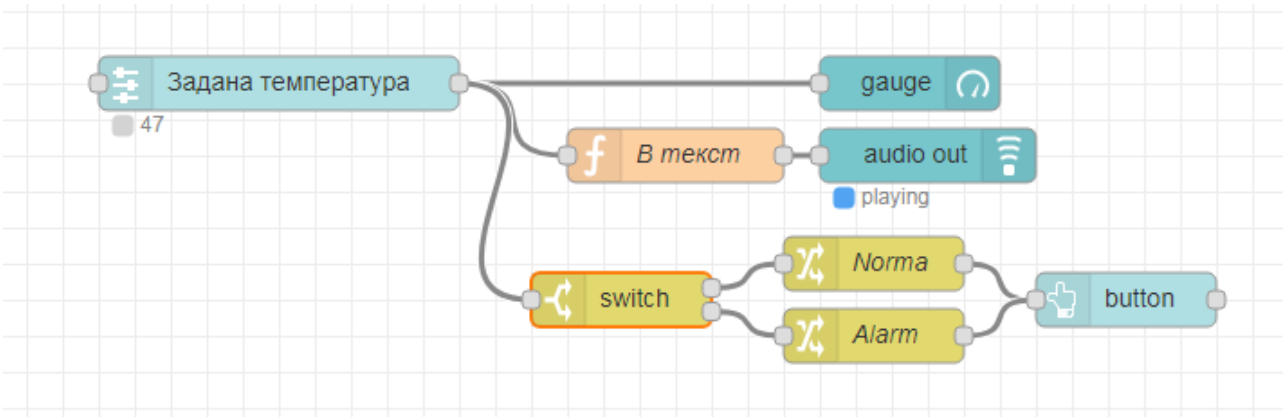

Рисунок 2.28 - Схема керування температурою з використанням вузла switch

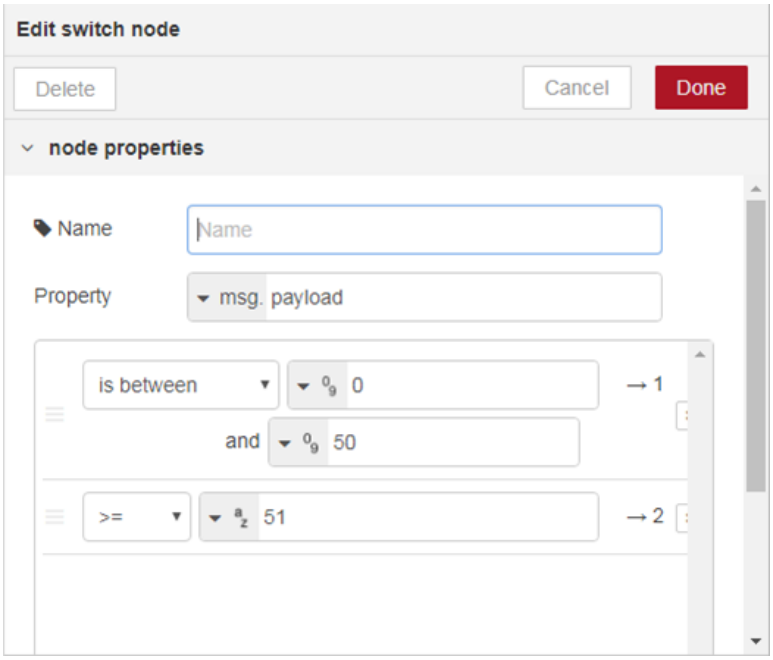

Рисунок 2.29 – Налаштування вузла switch для обробки значень температури

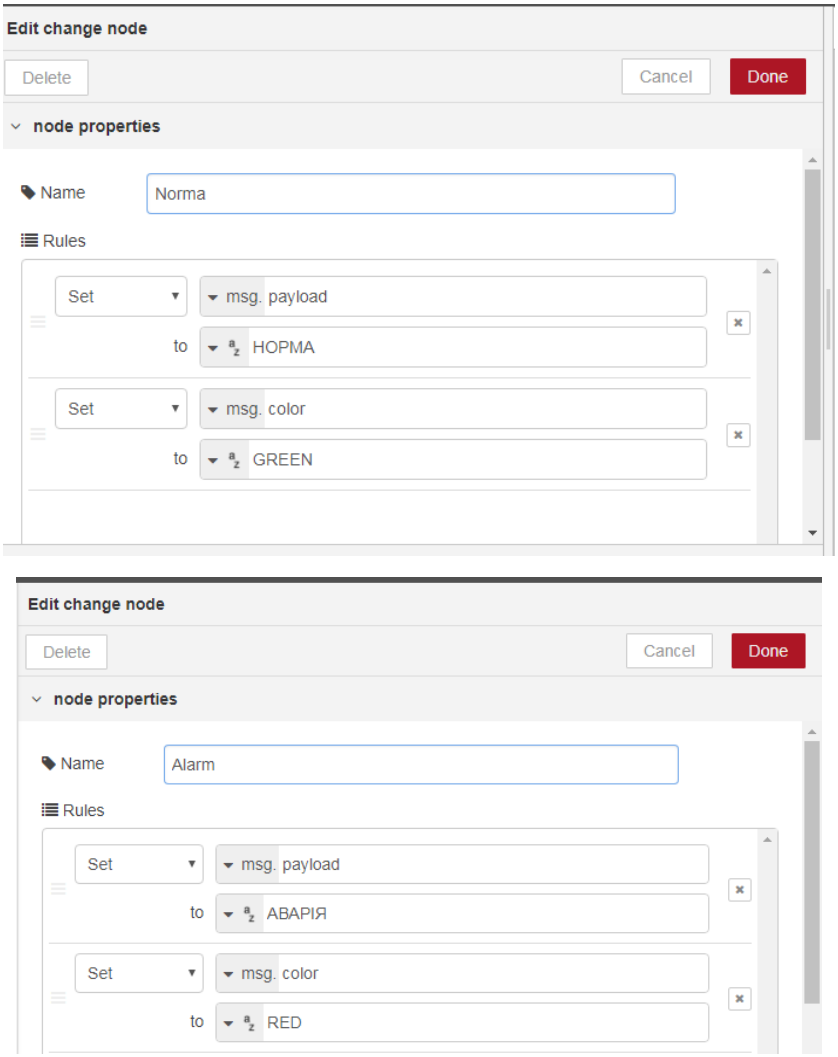

Рисунок 2.30 – Налаштування вузла change для статусу «Норма» та «Аварія»

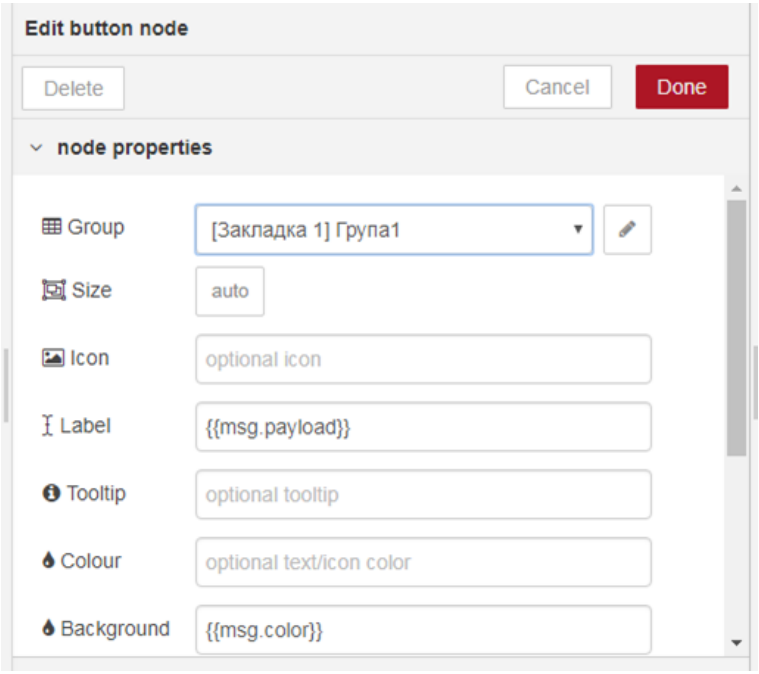

Рисунок 2.31 – Налаштування вузла button для відображення статусу

Виконайте розгортання проєкту та перевірте функціональність програми. Для цього на сторінці веб-інтерфейсу змініть значення заданої температури у діапазоні від 0 до 50, а також вище 50.

Ця частина програми функціонує наступним чином. При зміні значення температури дані передаються в поле msg.payload для обробки вузлом switch. У цьому вузлі, залежно від виконаної умови, формується повідомлення, яке спрямовується на один із двох виходів. У разі виконання умови 0 < msg.payload < 50 (оператор is between), повідомлення передається на перший вихід, до якого підключено вузол Norma (тип function->change). Цей вузол встановлює текстове значення властивості msg.payload як «НОРМА» та створює нову властивість msg.color зі значенням «GREEN». Далі повідомлення надходить до вузла button, який використовується для відображення тексту в прямокутнику. Значення тексту визначається в полі Label, а колір – у полі Background. Для динамічної підстановки значень у вузлах застосовується формат Angular-фільтрів із використанням подвійних фігурних дужок.

Аналогічний процес відбувається при виконанні умови msg.payload > 50 у вузлі switch. У цьому випадку повідомлення генерується на другому виході, що

активує перерахунок вузла Alarm, який формує відповідний текст і колір для кнопки.

Контрольні питання

1. Які сервіси електронної пошти підтримує Node-RED?
2. Як відправити повідомлення на email за допомогою вузлів Node-RED?
3. У чому особливості протоколу Modbus TCP?
4. Які вузли Node-RED призначені для роботи з Modbus?
5. Чим відрізняються Modbus TCP і Modbus RTU?
6. Як організувати читання та запис даних через Modbus у Node-RED?
7. Що таке Dashboard у Node-RED і для чого його використовують?
8. Які базові елементи інтерфейсу можна створювати у Node-RED Dashboard?
9. Як реалізується відображення графіків і показників у Dashboard?
10. Чому інтеграція поштових сервісів та Modbus є важливою для IoT-рішень?
11. Які приклади промислових застосувань Node-RED ви можете навести?

ПРАКТИЧНА РОБОТА № 3

Аналіз промислових IoT-даних із застосуванням кластеризації та технології MapReduce

Мета роботи. Ознайомлення з технологією паралельної обробки великих обсягів даних у контексті Промислового Інтернету Речей (IIoT) та набуття практичних навичок побудови моделі кластеризації з використанням шаблону MapReduce на базі програмної платформи Apache Hadoop.

Теоретичні відомості

Однією з ключових особливостей промислового Інтернету Речей (IIoT) є генерація значних обсягів даних сенсорними пристроями, системами моніторингу та керування. Для ефективної обробки таких даних використовуються спеціалізовані технології, зокрема MapReduce – модель паралельного програмування, яка забезпечує масштабоване опрацювання інформації у розподілених обчислювальних середовищах.

MapReduce є однією з найпоширеніших технологій розподіленої обробки даних великих обсягів. Її принципова особливість полягає у декомпозиції програмної логіки на дві незалежні підзадачі: розбиття вхідних даних та агрегування проміжних результатів з підтримкою необмеженого горизонтального масштабування за рахунок нарощування кількості обчислювальних вузлів у кластері.

Робота MapReduce передбачає два основні етапи: **розділення (map)** та **згортання (reduce)**. На етапі map вузол-координатор розбиває вхідний набір даних на частини і розподіляє їх між робочими вузлами для паралельної обробки. На етапі reduce агрегуються проміжні результати: головний вузол збирає їх від усіх виконавців і формує підсумковий розв'язок задачі [4].

Принцип роботи задачі MapReduce наочно представлений на Рис. 3.1.

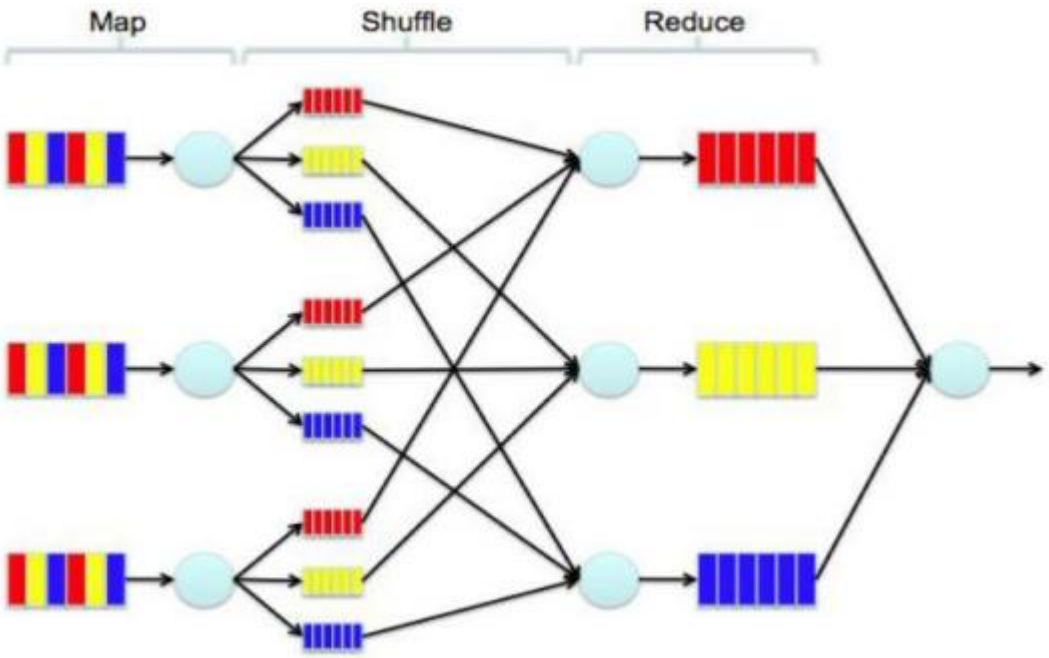

Рисунок 3.1 – Схема функціонування MapReduce

Серед найбільш поширених і практично затребуваних реалізацій парадигми MapReduce виділяють такі:

ApacheHadoop – відкрита платформа розподілених обчислень, призначена для опрацювання даних петабайтного масштабу. Вона реалізує парадигму MapReduce: кожна задача розбивається на велику кількість підзадач, що виконуються паралельно на окремих вузлах кластера.

Невід'ємною складовою Hadoop є розподілена файлова система HDFS (Hadoop Distributed Filesystem), що забезпечує автоматичне резервування даних на декількох вузлах та оптимізована для ефективного виконання MapReduce-задач у розподіленому середовищі. Завдяки такій архітектурі система залишається стійкою до відмов окремих вузлів без втрати даних. Для зручної роботи з інформацією, що зберігається у сховищі Hadoop, було розроблено низку додаткових інструментів: колонкову базу даних HBase, орієнтовану на роботу з розрідженими структурованими даними, а також мову запитів Hive – свого роду SQL-інтерфейс для MapReduce. Запити Hive автоматично транслюються у MapReduce-задачі та виконуються паралельно одразу на кількох вузлах Hadoop-кластера, що суттєво прискорює аналіз великих масивів даних.

Практичний приклад архітектури та роботи Apache Hadoop зображено на Рис. 3.2 [5].

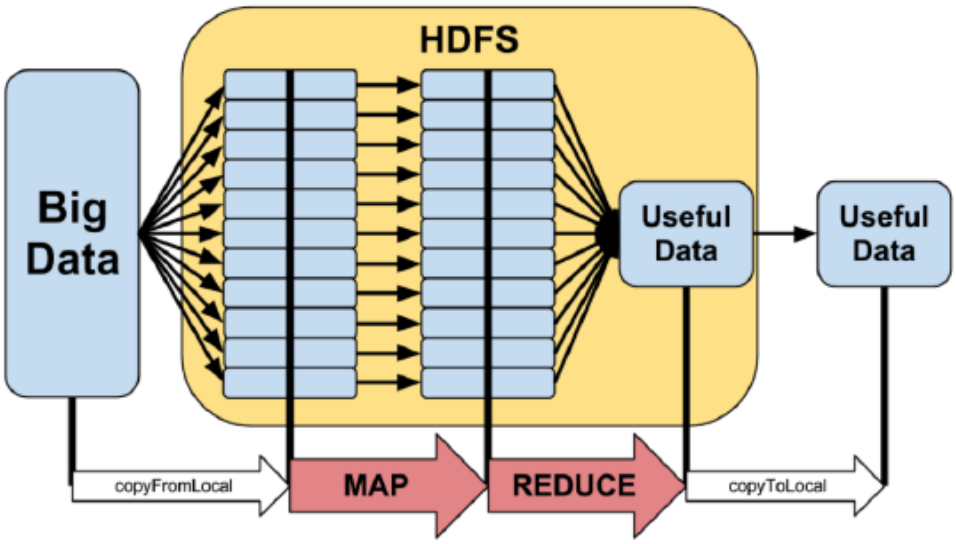

Рисунок 3.2 – Схема функціонування Apache Hadoop

Алгоритм кластеризації. У практиці розв'язання задач кластеризації найбільшого поширення набули три основні підходи:

- кластеризація методом k-середніх;
- ієрархічна кластеризація;
- кластеризація у графах (Graph Community Detection) [6].

На Рис. 3.3 наведено приклад трьох «чітких» кластерів при значенні $k = 3$.

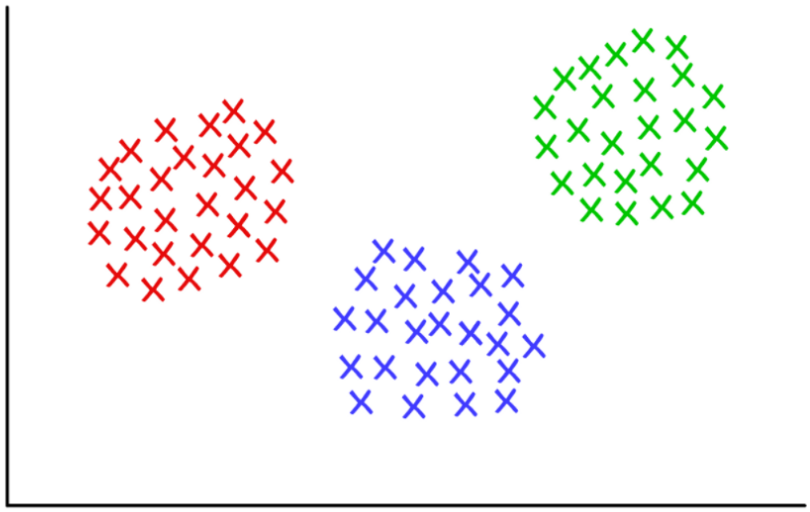

Рисунок 3.3 – Приклад трьох «чітких» кластерів при значенні $k = 3$

Метод k-середніх (k-means clustering) – один із найпоширеніших алгоритмів розбиття даних на відносно однорідні групи (кластери). Він був незалежно запропонований у 1950-х роках математиком Гуґо Штайнгаузом та інженером Стюартом Ллойдом і з того часу залишається базовим інструментом кластерного аналізу. Задача методу формулюється наступним чином: розбити множину з n спостережень на k непересічних підмножин таким чином, щоб кожен об'єкт був приписаний до того кластера, центроїд якого знаходиться до нього найближче у просторі ознак. Критерієм оптимальності розбиття є мінімізація внутрішньокластерної дисперсії – тобто сумарного квадрату евклідових відстаней між кожним спостереженням і центром відповідного кластера.

Алгоритм методу k -середніх:

1. Початок.
2. Обрати *k* точок даних як початкові центроїди кластерів.
3. Виконувати ітеративно до стабілізації положення центрів кластерів:
    a. Віднести кожну точку даних до того кластера, центроїд якого є найближчим до неї у просторі ознак.
    b. Перевірити наявність хоча б однієї точки у кожному кластері. За необхідності доповнити порожній кластер довільно обраною точкою, що максимально віддалена від решти.
    c. Перерахувати центр кожного кластера як середнє арифметичне координат усіх точок, що до нього належать.
4. Закінчення.

Завдання до практичної роботи

1. Загальна постановка задачі кластеризації для всіх розглянутих алгоритмів формулюється таким чином:

Вхідні дані: X – простір об'єктів; $X_l = \{x_i\}_{i=1}^{l}$ – вибірка елементів; $d: X \times X \to [0, \infty)$ – функція відстані між об‘єктами.

Результат: побудова множини кластерів Y та відображення $a: X \to Y$, яке

визначає алгоритм кластеризації, та задовольняє умову, що кожен кластер об’єднує об’єкти, близькі між собою за метрикою d, тоді як об’єкти, що належать до різних кластерів, суттєво відрізняються.

2. Побудувати модель кластеризації даних із застосуванням технології MapReduce.

Хід роботи

Як ілюстрацію практичного застосування моделі кластеризації розглянуто задачу підрахунку частоти входження слів, реалізовану засобами технології MapReduce на локальному кластері Hadoop. Розгортання кластера Hadoop здійснено з використанням платформи контейнеризації Docker, а програмна реалізація виконана мовою Python.

Розгортання кластера Hadoop.

Для реалізації системи використано архітектуру **master-slave**, де:

- master – вузол NameNode;
- slaves – вузли DataNodes_N.

У рамках даної роботи реалізовано конфігурацію з одним master-вузлом та трьома slave-вузлами (1 master : 3 slave).

```
D:\custom_hadoop>docker-compose up -d
Creating network "custom_hadoop_default" with the default driver
Creating custom_hadoop_hadoop-cluster-base_1 ... done
Creating custom_hadoop_yarn-nodemanager-4_1   ... done
Creating custom_hadoop_yarn-resourcemanager_1 ... done
Creating custom_hadoop_yarn-nodemanager-2_1   ... done
Creating custom_hadoop_hdfs-datanode-3_1      ... done
Creating custom_hadoop_hdfs-datanode-4_1      ... done
Creating custom_hadoop_gateway_1              ... done
Creating custom_hadoop_yarn-nodemanager-1_1   ... done
Creating custom_hadoop_hdfs-datanode-1_1      ... done
Creating custom_hadoop_yarn-nodemanager-3_1   ... done
Creating custom_hadoop_hdfs-datanode-2_1      ... done
Creating custom_hadoop_hdfs-namenode_1        ... done
```

```
D:\custom_hadoop>docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED
3cdc8fdaec73        custom_hadoop_hdfs-datanode-1        "/bin/sh -c '[ -z \"$…"   18 seconds ago
1ad80a7371d0        custom_hadoop_hdfs-datanode-2        "/bin/sh -c '[ -z \"$…"   18 seconds ago
05e48cca8872        custom_hadoop_yarn-resourcemanager   "/bin/sh -c '/opt/ha…"    18 seconds ago
1b7023101e7c        custom_hadoop_hdfs-namenode          "/bin/sh -c '/opt/ha…"    18 seconds ago
73a220c31870        custom_hadoop_yarn-nodemanager-2     "/bin/sh -c '[ -z \"$…"   18 seconds ago
763f870b0de9        custom_hadoop_yarn-nodemanager-1     "/bin/sh -c '[ -z \"$…"   18 seconds ago
a074dfb87345        custom_hadoop_yarn-nodemanager-3     "/bin/sh -c '[ -z \"$…"   18 seconds ago
5e69a42fff0e        custom_hadoop_hdfs-datanode-3        "/bin/sh -c '[ -z \"$…"   18 seconds ago
```

Рисунок 3.4 – Створення та запуск контейнерів Hadoop за допомогою Docker Compose

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✔hdfs-datanode-2:9866 (192.168.16.12:9866)	http://hdfs-datanode-2:9864	0s	32m	324 KB	7.68 GB	58.42 GB	4	324 KB (0%)	3.3.0
✔hdfs-datanode-3:9866 (192.168.16.2:9866)	http://hdfs-datanode-3:9864	0s	96m	108 KB	7.68 GB	58.42 GB	5	108 KB (0%)	3.3.0
✔hdfs-datanode-1:9866 (192.168.16.4:9866)	http://hdfs-datanode-1:9864	0s	69m	380 KB	7.68 GB	58.42 GB	7	380 KB (0%)	3.3.0

Рисунок 3.5 – Стан вузлів DataNode у веб-інтерфейсі HDFS

Копіювання локальних mapper.py, reducer.py в custom_hadoop_hdfs-namenode_1 (Рис. 3.6).

```
D:\Теф\Оброблення_надвеликих_масивів_даних\Lab1>docker cp mapper.py custom_hadoop_hdfs-namenode_1:mapper.py
D:\Теф\Оброблення_надвеликих_масивів_даних\Lab1>docker cp reducer.py custom_hadoop_hdfs-namenode_1:reducer.py
```

Рисунок 3.6 – Копіювання скриптів MapReduce у контейнер Hadoop NameNode

Контейнер custom_hadoop_hdfs-namenode_1

```
D:\Теф\Оброблення_надвеликих_масивів_даних\Lab1>docker exec -it custom_hadoop_hdfs-namenode_1 bash
hadoop@hdfs-namenode:~$
```

Рисунок 3.7 – Вхід у контейнер custom_hadoop_hdfs-namenode_1 через Bash

Вхідні дані:

```
hadoop@hdfs-namenode:~$ mkdir input
hadoop@hdfs-namenode:~$ echo "Hello World" >input/f1.txt
hadoop@hdfs-namenode:~$ echo "Hello Docker" >input/f2.txt
hadoop@hdfs-namenode:~$ echo "Hello Hadoop" >input/f3.txt
hadoop@hdfs-namenode:~$ echo "Hello MapReduce" >input/f4.txt
hadoop@hdfs-namenode:~$ ls
input
hadoop@hdfs-namenode:~$ cd input
hadoop@hdfs-namenode:~/input$ ls
f1.txt  f2.txt  f3.txt  f4.txt
hadoop@hdfs-namenode:~/input$
```

```
hadoop@hdfs-namenode:~/input$ cat f1.txt
Hello World
hadoop@hdfs-namenode:~/input$ cat f2.txt
Hello Docker
hadoop@hdfs-namenode:~/input$ cat f3.txt
Hello Hadoop
hadoop@hdfs-namenode:~/input$ cat f4.txt
Hello MapReduce
hadoop@hdfs-namenode:~/input$
```

Рисунок 3.8 – Створення папки з вхідними даними

Програма MapReduce зчитує вхідні дані безпосередньо з розподіленої файлової системи Hadoop (HDFS).

Завантаження вхідного каталогу та файлів до файлової системи HDFS:

```
hadoop@hdfs-namenode:~$ hadoop fs -mkdir -p input
hadoop@hdfs-namenode:~$ hdfs dfs -put ./input/* input
put: `input/f1.txt': File exists
put: `input/f2.txt': File exists
put: `input/f3.txt': File exists
put: `input/f4.txt': File exists
```

Рисунок 3.9 – Завантаження файлів у каталог файлової системи HDFS

Ініціалізація та виконання програми MapReduce.

```
hadoop@hdfs-namenode:~$ hadoop jar ../../opt/hadoop-3.3.0/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file ../../
mapper.py    -mapper mapper.py -file ../../reducer.py   -reducer reducer.py -input input -output output
```

Рисунок 3.10 – Запуск задачі MapReduce у середовищі Hadoop

Результуючий вихідний файл (part-r-00000):

```
hadoop@hdfs-namenode:~$ hdfs dfs -cat output/*
Docker  1
Hadoop  1
Hello   4
MapReduce       1
World   1
```

Рисунок 3.11 – Результати виконання Hadoop MapReduce (WordCount)

Лістинг програми

mapper.py

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# вхідні дані надходять зі стандартного введення (STDIN)
```

```
    for line in sys.stdin:
        # видаляємо пробіли на початку та в кінці рядка
        line = line.strip()
        # розбиваємо рядок на слова
        words = line.split()
        # підраховуємо входження
        for word in words:
            #  виводимо  результати  до  стандартного  виведення
(STDOUT);
            # цей вивід стане вхідними даними для етапу Reduce,
            # тобто для reducer.py
            #
            #  формат:  слово[TAB]лічильник;  початкове  значення
лічильника — 1
            print('%s\t%s' % (word, 1))
```

reducer.py

```
    #!/usr/bin/env python
    """reducer.py"""

    from operator import itemgetter
    import sys

    current_word = None
    current_count = 0
    word = None

    # вхідні дані надходять зі стандартного введення (STDIN)
    for line in sys.stdin:
        # видаляємо пробіли на початку та в кінці рядка
        line = line.strip()

        # розбираємо рядок, отриманий від mapper.py
        word, count = line.split('\t', 1)

        # перетворюємо лічильник із рядка на ціле число
        try:
            count = int(count)
        except ValueError:
            # якщо значення не є числом — мовчки пропускаємо рядок
            continue

        # цей IF-блок працює коректно, оскільки Hadoop сортує
        # вивід mapper за ключем (тут — словом) перед передачею до
reducer
        if current_word == word:
            current_count += count
        else:
            if current_word:
                #  виводимо  результат  до  стандартного  виведення
(STDOUT)
                print('%s\t%s' % (current_word, current_count))
```

```
        current_count = count
        current_word = word

# не забуваємо вивести останнє слово, якщо воно є
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

Приклад демонструє базову обробку даних за допомогою MapReduce на Hadoop, використовуючи синтетичний текстовий набір для підрахунку слів. У реальних умовах Smart Factory ці методи застосовуються для аналізу великих обсягів IIoT-даних, таких як сенсорні показники чи журнали подій, із можливістю реалізації кластеризації для групування станів обладнання.

Контрольні питання

1. Що таке Hadoop і які його основні компоненти?
2. У чому полягає ідея парадигми MapReduce?
3. Які є етапи виконання алгоритму MapReduce?
4. Які задачі найкраще вирішуються за допомогою MapReduce?
5. Як у MapReduce реалізується кластеризація даних?
6. Що таке ключ (key) і значення (value) в MapReduce?
7. Як Hadoop забезпечує масштабованість і відмовостійкість?
8. Чим MapReduce відрізняється від класичної обробки даних?
9. У чому переваги MapReduce для аналізу великих обсягів IoT-даних?
10. Які недоліки MapReduce у порівнянні з іншими технологіями Big Data?
11. Які приклади задач кластеризації у промисловому IoT ви можете назвати?

ПРАКТИЧНА РОБОТА № 4

Інтелектуальний пошук і керування даними у Smart Factory за допомогою Apache Spark

Мета роботи. Отримати навички роботи з даними у середовищі Smart Factory. Навчитися застосовувати Spark SQL для обробки та аналізу даних.

Теоретичні відомості

Платформа Spark забезпечує підтримку трьох мов програмування: Scala, Python та Java. Основні структурні компоненти Spark представлені на рисунку 4.1 [7].

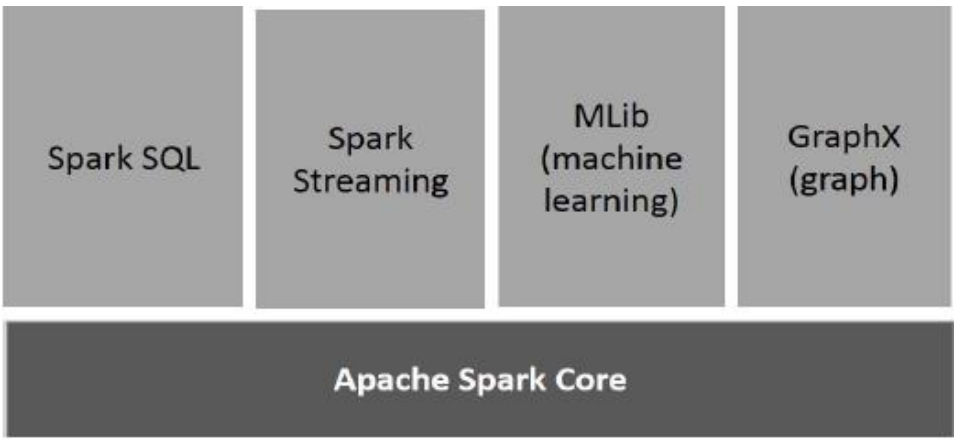

Рисунок 4.1 – Основні компоненти Spark

Apache Spark Core є фундаментальним рушієм платформи Spark, на якому ґрунтуються всі інші її модулі. Саме Spark Core відповідає за виконання обчислень у оперативній пам'яті та забезпечує механізми посилань на набори даних, що зберігаються у зовнішніх системах.

Spark SQL. Spark SQL - модуль, що надбудовується над Spark Core і вводить абстракцію SchemaRDD для роботи зі структурованими та напівструктурованими даними. Він дозволяє виконувати SQL-запити безпосередньо до датасетів у розподіленому середовищі, поєднуючи декларативний синтаксис SQL із потужністю розподілених обчислень Spark.

Spark Streaming. Spark Streaming реалізує потокову обробку даних, спираючись на механізм швидкого планування задач Spark Core. Вхідний потік розбивається на невеликі часові фрагменти - міні-пакети, над кожним із яких застосовуються стандартні RDD-перетворення. Такий підхід дозволяє уніфікувати обробку потокових і пакетних даних у межах єдиного програмного інтерфейсу.

MLlib (Бібліотека машинного навчання). MLlib - вбудована бібліотека машинного навчання для Spark, що повністю використовує його розподілену in-memory архітектуру. Завдяки цьому алгоритми MLlib демонструють суттєво вищу продуктивність порівняно з дисковими реалізаціями: зокрема, за результатами бенчмарків на задачі Alternating Least Squares (ALS) Spark MLlib перевершує дискову версію Apache Mahout для Hadoop приблизно у дев'ять разів. Бібліотека охоплює широкий спектр алгоритмів: класифікацію, регресію, кластеризацію, колаборативну фільтрацію та методи зниження розмірності.

Graphx. GraphX - компонент Spark для розподіленої обробки графових структур. Він надає програмний інтерфейс для опису графових обчислень і підтримує модель Pregel – абстракцію для ітеративних алгоритмів на графах. Окрім виразності API, GraphX забезпечує оптимізоване виконання графових задач завдяки внутрішнім структурам зберігання та плануванню обчислень, що мінімізують накладні витрати на комунікацію між вузлами.

Центральною абстракцією Spark є RDD (Resilient Distributed Dataset) – відмовостійкий розподілений набір даних, що зберігається у пам'яті вузлів кластера. Вся робота з RDD зводиться до послідовного застосування двох типів операцій: трансформацій і дій, що утворює основний програмний патерн Spark-додатків. (див. рисунок 4.2)[8].

Трансформації – це ліниві операції над RDD, результатом яких завжди є новий RDD. Вони не виконуються негайно, а лише формують граф обчислень, який матеріалізується при виклику дії. Серед найпоширеніших трансформацій:

- **map (function)** - застосовує задану функцію до кожного елемента датасету і повертає новий RDD із перетворених значень;

- **filter (function)** - відбирає лише ті елементи, для яких функція повертає істинне значення;
- **distinct ([numTasks])** - повертає RDD із дедублікованими елементами вихідного датасету.

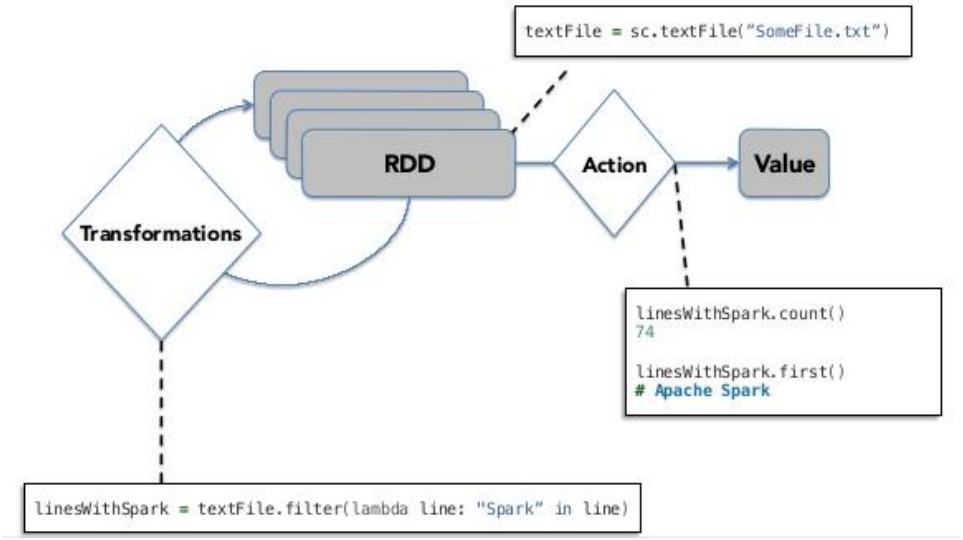

Рисунок 4.2 – Принцип роботи з RDD

Окремо виділяють операції над множинами, семантика яких випливає з їхніх назв:

- **union (otherDataset)** об'єднує два RDD,
- **intersection (otherDataset)** повертає їхній перетин,
- **cartesian (otherDataset)** формує декартів добуток - новий RDD із усіма парами (A, B), де A належить вихідному датасету, а B - аргументу.**Дії**

Дії - операції, що ініціюють фактичне виконання всього накопиченого графа трансформацій і матеріалізують результат: записують його на диск або повертають у драйвер-програму. Саме виклик дії запускає планувальник Spark і розподіляє обчислення між вузлами кластера. Серед найпоширеніших дій над RDD:

• **saveAsTextFile (path)** - записує вміст датасету у текстовий файл у підтримувану файлову систему (HDFS, локальний диск або інші сумісні сховища; повний перелік наведено в документації);

• **collect ()** - повертає всі елементи датасету у вигляді масиву в пам'ять драйвер-програми. Використовується після застосування фільтрів і трансформацій, коли обсяг даних вже невеликий, — наприклад, для подальшої візуалізації засобами бібліотеки Pandas;

• **take (n)** - повертає перші n елементів датасету у вигляді масиву, не завантажуючи весь RDD у пам'ять;

• **count ()** - повертає загальну кількість елементів у датасеті;

• **reduce (function)** - агрегує елементи датасету за допомогою бінарної асоціативної та комутативної функції, що приймає два аргументи і повертає одне значення. Семантика операції аналогічна reduce у парадигмі MapReduce.

Завдання до практичної роботи

1. Сформувати індекс у форматі RDD[(A, B)], де ключем виступає окреме слово, а відповідним значенням — перелік посилань на сторінки, на яких це слово зустрічається.
2. Створити тимчасову таблицю Spark SQL із двома стовпцями: значення метаданих та посилання на сторінку, якій це значення відповідає.
3. Реалізувати функціональність повнотекстового пошуку за ключовими словами на сторінках.
4. Реалізувати функціональність пошуку сторінок за значеннями їхніх метаданих.

Хід роботи

Приклад демонструє інтелектуальний пошук у синтетичних текстових даних, що імітують журнали подій обладнання, за допомогою RDD і Spark SQL для створення індексу слів та таблиці метаданих. У реальних умовах Smart Factory ці методи застосовуються для пошуку та аналізу виробничих логів, сенсорних даних чи метаданих обладнання.

Робота з RDD. Вхідними даними програми є асоціативний масив, у якому ключами слугують URL-адреси сторінок, а значеннями — їхній текстовий вміст. Фрагмент програми, що відповідає за виконання першого пункту завдання, створює RDD та послідовно застосовує до нього ланцюжок трансформацій із використанням функцій flatMap, groupBy та map.

На наступному етапі здійснюється пошук заданого слова у побудованому RDD. Результатом виконання пошукового запиту є список посилань на сторінки, що містять шукане слово.

Приклад виконання програми:

Вхідні дані:

```
inputData = {
'link1': 'Foo Bar Baz Boo Foo',
'link2': 'Foo Foo',
'link3': 'Bar Baz Baz'
}
```

Побудований RDD має вигляд:

```
[
{'Foo': ['link1', 'link2']},
{'Bar': ['link3', 'link1']},
{'Baz': ['link3', 'link1']},
{'Boo': ['link1']}
]
```

Шукане слово: Baz

Результат виконання програми:

```
[
{
"Baz": [
"link3",
"link1"
]
}
]
```

Робота з інструментом SQL в Spark. На вхід програма отримує асоціативний масив, у якому ключі - посилання на сторінки, значення - текст, що містить метадані цих сторінок.

Приклад:

```
inputData = {
'link1': '<meta name="Foo Baz">',
'link2': '<meta content="Foo Bar"><meta content="Baz">',
```

```
'link3': '<meta content="Baz">'
}
```

Програма формує тимчасову таблицю Spark SQL із двома стовпцями: перший містить значення метатегу, другий — посилання на сторінку, якій це значення належить. До створення таблиці вхідні дані попередньо приводяться до необхідного формату шляхом застосування відповідних функцій трансформації.

Структура сформованої таблиці:

```
[
{"MetaData: FooBaz": "Link: link1"},
{"MetaData: Foo Bar": "Link: link2"},
{"MetaData: Baz": "Link: link2"},
{"MetaData: Baz": "Link: link3"}
]
```

Знайдемо у даній таблиці посилання на сторінки які мають значення метаданого Baz.

Результат роботи програми:

```
[
{"MetaData: FooBaz": "Link: link1"},
{"MetaData: Baz": "Link: link2"},
{"MetaData: Baz": "Link: link3"}
]
```

Лістинг програми

```
from __future__ import print_function

import sys
import json
from random import random
from operator import add
import re

from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql.types import *

if __name__ == "__main__":

    spark = SparkSession\
        .builder\
        .appName("lab2 Apache Spark")\
        .getOrCreate()

    sc = spark.sparkContext

#################### Task 1 ####################
# Побудувати індекс типу RDD[(A, B)], де ключем є слово, а
```

```
# значенням є посилання, на сторінках яких зустрічається це слово.

inputData = {
    'link1': 'word1 word2 word3 word1',
    'link2': 'word2 word3',
    'link3': 'word1 word3'
}

words_rdd = sc.parallelize(inputData.items()) \
    .flatMap(lambda line: [(word, line[0]) for word in line[1].split("
")]) \
    .groupByKey() \
    .map(lambda word_link: (word_link[0], list(set([link for link in
word_link[1]]))))

result = words_rdd.collect()
print('====> Task 1 ------------------------------------')
print(result)
print(json.dumps(result, indent=2))
print('<==== Task 1 ------------------------------------')


#################### Task 3 ####################
# Реалізувати пошук за словами на сторінці.

word_to_find = 'word1'

words_index_rdd = words_rdd.filter(
    lambda word_links: word_to_find in word_links[0]
)

result = words_index_rdd.collect()
print('====> Task 3 ------------------------------------')
print(result)
print(json.dumps(result, indent=2))
print('<==== Task 3 ------------------------------------')


#################### Task 2 ####################
# На основі словника створити Spark SQL таблицю з 2 стовпців: (значення
# мета-тегу, посилання на сторінку, на якій це значення знаходилось).

inputData = {
    'link1': '<meta name="Foo" content="Bar">',
    'link2': '<meta name="FooBar" content="Baz">',
    'link3': '<meta name="BarFoo" content="Baz">'
}

meta_link_rdd = sc.parallelize(inputData.items()) \
    .flatMap(lambda line: [Row(meta_data=re.search('content="(.*)"',
line[1]).group(1),
                               link=line[0]) for meta_word in
re.findall('content="(.*)"', line[1])])

schemaMetaLink = spark.createDataFrame(meta_link_rdd)
schemaMetaLink.createOrReplaceTempView("meta_link")

# SQL can be run over DataFrames that have been registered as a table.
```

```
meta_link_sql = spark.sql("SELECT * FROM meta_link")

# the results of SQL queries are DataFrame objects.
rdd = meta_link_sql.rdd.map(
    lambda p: ({"metaData": p.meta_data, "Link": p.link})
)

result = meta_link_rdd.collect()
print('====> Task 2 -------------------------------------')
print(result)
print(json.dumps(result, indent=2))
print('<==== Task 2 -------------------------------------')


#################### Task 4 ####################
# Реалізувати пошук за метаданими сторінок.

meta_data_to_find = 'Baz'

# SQL can be run over DataFrames that have been registered as a table.
result = spark.sql(
    "SELECT * FROM meta_link WHERE meta_data LIKE \'%" +
meta_data_to_find + "%\'"
)

# the results of SQL queries are DataFrame objects.
rdd = result.rdd.map(
    lambda p: ({"metaData": p.meta_data, "Link": p.link})
)

result = meta_link_rdd.collect()
print('====> Task 4 -------------------------------------')
print(result)
print(json.dumps(result, indent=2))
print('<==== Task 4 -------------------------------------')

spark.stop()
```

Контрольні питання

1. Що таке Apache Spark і які його ключові модулі?
2. Що таке DataFrame у Spark і чим він схожий на таблицю SQL?
3. Як створюється DataFrame у Spark?
4. Які операції можна виконувати над DataFrame?
5. Чим відрізняється DataSet від DataFrame?
6. Як у Spark SQL реалізується виконання запитів SQL?
7. Які переваги Spark SQL у порівнянні з класичними СУБД?
8. Як Spark SQL використовується для обробки IoT-даних?
9. Що таке Catalyst Optimizer і яка його роль?

10. Як можна інтегрувати Spark SQL із зовнішніми джерелами даних?

11. Які приклади застосувань Spark SQL у Smart Factory ви можете навести?

ПРАКТИЧНА РОБОТА № 5

Задачі класифікації та регресії з використанням бібліотеки машинного навчання MLlib у контексті розумного виробництва

Мета роботи. Навчитися вирішувати задачі класифікації та регресії та створювати моделі з використанням бібліотеки машинного навчання MLlib у сфері розумного виробництва.

Теоретичні відомості

Машинне навчання, охоплює як підходи з учителем, так і без нього та є однією з центральних дисциплін сучасної інтелектуальної обробки даних. В його основі лежить побудова математичних моделей, що використовують методи статистичного аналізу та навчальні алгоритми для формування прогнозів на підставі накопиченого масиву спостережень [10]. У контексті Smart Factory застосування таких методів дозволяє автоматизувати процеси прийняття рішень у виробничих системах там, де традиційні аналітичні підходи виявляються обмеженими або недостатньо точними.

Статистичний аналіз відіграє ключову роль у дослідженні виробничих даних і є необхідною основою для побудови ефективних моделей машинного навчання. Він охоплює різноманітні методи: зокрема, аналіз часових рядів, що дає змогу виявляти стійкі закономірності та тренди у динаміці процесів, а також A/B-тестування (split testing) – метод контрольованого порівняння, у якому базові показники зіставляються з результатами тестових груп, де цілеспрямовано змінено один або кілька параметрів системи чи технологічного процесу. Такий підхід дозволяє кількісно оцінити, які саме модифікації справді позитивно впливають на якість продукції або продуктивність обладнання, а не є наслідком випадкових коливань.

Поняття про кореляцію ознак. Важливим елементом статистичного аналізу є вивчення кореляції між ознаками. Методи статистики дозволяють аналізувати окремі характеристики обладнання ізольовано, проте у Smart Factory

першочерговим завданням є виявлення залежностей між кількома змінними (наприклад, температура та тиск у процесі лиття металу). У таких випадках рідко спостерігаються строго функціональні зв’язки, частіше мають місце стохастичні або кореляційні залежності, що проявляються лише при масових спостереженнях. Кореляції можуть класифікуватися за напрямом (пряма чи обернена), формою, силою та кількістю встановлених зв’язків.

Класифікація – найбільш поширене завдання машинного навчання, що являє собою процес віднесення вхідних даних до певних категорій або класів. Алгоритм класифікації визначає принцип, за яким вхідним об’єктам призначаються мітки класів.

Навчальна вибірка складається з об’єктів, для яких відома їхня класова належність. Завдання алгоритму полягає у побудові моделі, здатної коректно класифікувати нові, невідомі об’єкти. Таким чином, результатом класифікації є віднесення кожного об’єкта до певного класу, що виражається у вигляді номера або найменування цього класу.

Класифікувати об'єкт – означає однозначно визначити, до якої категорії він належить, та присвоїти йому відповідну мітку. Таким чином, результатом класифікації конкретного об'єкта є порядковий номер або найменування класу, що повертається алгоритмом після його застосування до вхідних даних цього об'єкта.

Класифікація відноситься до задач навчання з учителем. У такому підході між множиною об'єктів та їхніми відповідями існує прихована залежність, яка апріорі невідома дослідникові. Для навчання алгоритму використовується обмежена вибірка пар «об'єкт – відповідь», що отримала назву навчальної вибірки або прецедентної бази. Спираючись на ці дані, будується модель, здатна з прийнятною точністю відтворювати приховану закономірність і генерувати коректні прогнози для раніше не бачених об'єктів. Для оцінювання якості побудованої моделі вводиться функціонал якості, що характеризує точність і надійність отриманих рішень на тестових даних. Роль «учителя» у цьому процесі

відіграє або сама навчальна вибірка, або експерт-розмітник, який встановив правильні відповіді для відомих прикладів.

Окрім навчання з учителем, існує також навчання без учителя, коли у вибірці відсутні мітки класів, і алгоритм самостійно виявляє структуру даних, наприклад, здійснює кластеризацію.

Серед методів, які застосовуються у задачах класифікації, окреме місце займає **логістична регресія**. Цей алгоритм належить до категорії статистичних моделей і дозволяє ефективно розв'язувати задачі двійкової класифікації, тобто віднесення об'єктів до однієї з двох груп. Зокрема, API Spark забезпечує інструментарій для реалізації логістичної регресії, що робить цей метод практичним та зручним для обробки великих обсягів даних.

Завдання до практичної роботи

Необхідно вирішити задачу класифікації виробничих подій у Smart Factory на основі даних сенсорів. Система повинна розпізнавати категорії подій, наприклад: «Нормальна робота», «Аварійне відхилення», «Профілактичне обслуговування».

При вирішенні задачі необхідно:

1. Провести дослідний аналіз даних.
2. Проаналізувати статистичні властивості ознак у даних, порівняти їх між навчальною та тестовою вибіркою.
3. Провести початковий відбір ознак на базі статистичних властивостей.
4. Сформувати остаточний набір ознак, навчити модель і перевірити її якість.
5. Проаналізувати внутрішній устрій моделі для ідентифікації точок зростання.

Хід роботи

У практичній роботі для прикладу вирішується задача багатокласової класифікації даних сенсорів обладнання на категорії, необхідно визначити, які з

записів належать до певної категорії «Нормальна робота», «Аварійне відхилення» та «Профілактичне обслуговування».

Такий приклад є навчальною ілюстрацією, яка дозволяє відпрацювати методику побудови моделей у Spark MLlib.

Оскільки класифікація відноситься до навчання з учителем, у початкових даних повинні бути мітки – категорії, до яких належать повідомлення. Саме завдяки цим міткам модель навчається знаходити залежності між текстами та класами.

На початковому етапі така залежність невідома, проте після навчання алгоритм зможе визначати закономірності у даних та відносити нові об’єкти до потрібної категорії. Модель буде здатна визначати залежності між об’єктами класифікації та класами після того як буде навчена.

Перед тим як розпочати навчання, дані необхідно очистити та підготувати.

Кроки очищення:

1. Видалити всі нерелевантні елементи (наприклад, пропущені значення або аномальні записи за межами очікуваних діапазонів).

2. Нормалізувати числові ознаки, якщо потрібно, для забезпечення однакового масштабу (наприклад, температура, тиск, вібрація).

3. Видалити нерелевантні ознаки, якщо вони не впливають на класифікацію.

4. Перевірити на баланс класів та, за потреби, застосувати методи балансування.

5. Розглянути можливість генерації додаткових ознак, якщо базові недостатньо інформативні (наприклад, обчислення середнього або дисперсії).

Після того, як будуть виконі ці кроки, а також виконано перевірку на додаткові помилки, можливо розпочати використовувати «чисті» підготовлені дані для навчання моделі.

Моделі машинного навчання оперують числовими значеннями. Наприклад, моделі, що працюють із сенсорними даними, приймають вектори, що відображають значення ознак, таких як температура, тиск чи вібрація.

Так як датасет є масивом числових значень (в даному випадку), тому для того, щоб алгоритм міг «витягти» патерни даних, спочатку необхідно знайти метод підходящий представлення даних.

Метод Vector Assembler («збірник векторів»). Його сутність полягає у створенні векторів ознак з усіх релевантних числових колон датасету. Кожне спостереження відображається у вигляді вектора, довжина якого дорівнює кількості ознак, а значення у відповідних позиціях відповідають числовим значенням ознак. Такий підхід дозволяє оцінити залежності між ознаками.

На рисунку 5.1 наведено приклад реалізації цього методу.

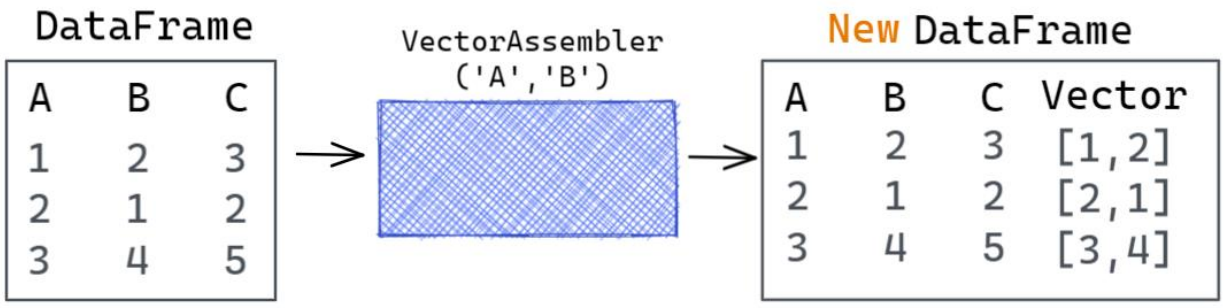

Рисунок 5.1 – Ілюстрація методу Vector Assembler

Наступним етапом є вибір алгоритму для побудови моделі класифікації. Одним із найпоширеніших методів, що застосовуються для розв’язання задач класифікації числових даних, є логістична регресія. Цей підхід дозволяє відносити об’єкти до кількох класів та добре підходить для роботи з підготовленими векторними представленнями даних.

Для коректної оцінки роботи алгоритму датасет необхідно розділити на дві частини – тренувальну вибірку та тестову вибірку. Тренувальна частина використовується для побудови моделі, тоді як тестова застосовується для перевірки її точності на нових даних, які не брали участі у процесі навчання. Такий підхід дозволяє уникнути перенавчання та об’єктивно оцінити якість класифікації.

Далі відбувається навчання моделі за допомогою методу логістичної

регресії. Алгоритм поступово формує залежності між вхідними ознаками та вихідними класами на основі навчальної вибірки. Після завершення навчання модель готова до перевірки та використання.

На фінальному кроці здійснюється оцінювання результатів роботи моделі. Для цього виводиться показник точності (accuracy), який відображає частку правильно класифікованих об’єктів у тестовій вибірці. Отримане значення точності дозволяє зробити висновок щодо ефективності застосованого методу.

Лістинг програми

```
# Імпорт необхідних бібліотек
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, DoubleType,
StringType
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.stat import Correlation
import pandas as pd
import random

# Ініціалізація Spark-сесії
spark =
SparkSession.builder.appName("SensorClassification").getOrCreate()

# Визначення схеми для синтетичного датасету
schema = StructType([
    StructField("temperature", DoubleType(), True),
    StructField("pressure", DoubleType(), True),
    StructField("vibration", DoubleType(), True),
    StructField("status", StringType(), True)
])

# Генерація синтетичних даних (100 записів для прикладу)
data = []
for i in range(100):
    if random.random() < 0.4:  # Нормальна робота
        temp = random.uniform(24, 26)
        press = random.uniform(1.1, 1.3)
        vib = random.uniform(0.4, 0.6)
        status = "Normal"
    elif random.random() < 0.7:  # Аварійне відхилення
        temp = random.uniform(29, 31)
        press = random.uniform(1.4, 1.6)
        vib = random.uniform(0.9, 1.2)
        status = "Anomaly"
    else:  # Профілактичне обслуговування
        temp = random.uniform(27, 29)
        press = random.uniform(1.25, 1.35)
        vib = random.uniform(0.65, 0.75)
        status = "Maintenance"
```

```
    data.append((temp, press, vib, status))

# Створення DataFrame
dataset = spark.createDataFrame(data, schema)

# У реальному випадку: читання з файлу (розкоментувати за потреби)
# file_location = "/FileStore/tables/sensors_dataset.csv"
# file_type = "csv"
# infer_schema = "true"
# first_row_is_header = "true"
# delimiter = ";"
# dataset = spark.read.format(file_type) \
#     .option("inferSchema", infer_schema) \
#     .option("header", first_row_is_header) \
#     .option("sep", delimiter) \
#     .load(file_location)

# Дослідний аналіз: виведення статистики датасету
print("Опис датасету:")
dataset.describe().show()

# Розділення на тренувальну (80%) та тестову (20%) вибірки
(train, test) = dataset.randomSplit([0.8, 0.2], seed=123)

# Аналіз кореляції між ознаками
assembler_for_corr = VectorAssembler(inputCols=["temperature",
"pressure", "vibration"], outputCol="features")
df_vector = assembler_for_corr.transform(dataset)
correlation_matrix = Correlation.corr(df_vector, "features").head()[0]
print("Матриця кореляції:\n", correlation_matrix)

# Порівняння статистики між тренувальною та тестовою вибірками
print("Статистика тренувальної вибірки:")
train.describe().show()
print("Статистика тестової вибірки:")
test.describe().show()

# Підготовка даних: конвертація міток у числові значення
indexer = StringIndexer(inputCol="status", outputCol="label")

# Об'єднання ознак у вектор
assembler = VectorAssembler(inputCols=["temperature", "pressure",
"vibration"], outputCol="features")

# Налаштування моделі логістичної регресії
lr = LogisticRegression(featuresCol="features", labelCol="label",
maxIter=10)

# Створення пайплайну
pipeline = Pipeline(stages=[indexer, assembler, lr])

# Навчання моделі
model = pipeline.fit(train)

# Прогнозування на тестовій вибірці
predictions = model.transform(test)

# Оцінка якості моделі
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction")
accuracy = evaluator.evaluate(predictions, {evaluator.metricName:
"accuracy"})
f1 = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})
print(f"Точність (Accuracy): {accuracy:.3f}")
print(f"F1-score: {f1:.3f}")

# Аналіз коефіцієнтів моделі
coefficients = model.stages[-1].coefficientMatrix.toArray()
intercept = model.stages[-1].interceptVector.toArray()
feature_names = ["temperature", "pressure", "vibration"]
print("Коефіцієнти моделі для кожного класу:")
for i, class_name in enumerate(model.stages[0].labels):
    print(f"Клас {class_name}:")
    for j, feature in enumerate(feature_names):
        print(f"  {feature}: {coefficients[i][j]:.4f}")
    print(f"  Intercept: {intercept[i]:.4f}")

# Аналіз важливості ознак (на основі абсолютних значень коефіцієнтів)
importance_data = []
for i, class_name in enumerate(model.stages[0].labels):
    for j, feature in enumerate(feature_names):
        importance_data.append((class_name, feature,
abs(coefficients[i][j])))
importance_df = pd.DataFrame(importance_data, columns=["Class",
"Feature", "Weight"])
print("\nВажливість ознак (абсолютні значення коефіцієнтів):")
print(importance_df.sort_values(by="Weight", ascending=False))

# Зупинка Spark-сесії
spark.stop()
```

Навчальний приклад демонструє класифікацію синтетичних сенсорних даних для Smart Factory, що дозволяє автоматизовано виявляти стани обладнання з використанням методів Spark MLlib. У реальних умовах ці методи застосовуються для аналізу справжніх виробничих даних, таких як сенсорні сигнали чи журнали подій обладнання, для прогнозування станів процесів.

Контрольні питання

1. Що таке MLlib і які алгоритми вона реалізує?
2. Яка різниця між задачами класифікації та регресії?
3. Які алгоритми класифікації підтримуються у Spark MLlib?
4. Які алгоритми регресії доступні у MLlib?
5. Що таке навчальна вибірка та тестова вибірка?

6. Які показники якості моделей класифікації ви знаєте?

7. Що таке overfitting і як його уникнути?

8. Як у MLlib реалізується побудова пайплайнів (pipelines)?

9. Які приклади регресійних задач можна навести у промисловості?

10. Як моделі машинного навчання можуть бути інтегровані у Smart Factory?

11. Які приклади класифікаційних задач у Smart Factory можна назвати?

ПЕРЕЛІК ПОСИЛАНЬ

1. Жураковський Б. Ю., Зенів І. О. Технології Інтернету речей : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 271 с. URL: https://ela.kpi.ua/items/03f7d3c6-aa4a-4615-b864-0b4a6ab409e4

2. Довідник Node-RED з елементами опису JavaScript, JSON, JSONata / Перекладач О. Пупена. URL: https://github.com/pupenasan/NodeREDGuidUKR

3. Пулеко І. В., Єфіменко А. А. Архітектура та технології Інтернету речей : навч. посіб. Житомир : Житомирська політехніка, 2022. 260 с. URL: https://eztuir.ztu.edu.ua/123456789/8093

4. Невлюдов І. Ш., Новоселов С. П., Сичова О. В. Node-RED та технологія промислового Інтернету речей : навч. посіб. 2-ге вид., перероб. і доп. Харків : ХНУРЕ, 2024. 271 с. URL: https://publish.nure.ua/catalog/book/506

5. Onwuegbuzie I. U., Olowojebutu A. O., Akomolede K. K. Node-RED and IoT Analytics: A Real-Time Data Processing and Visualization Platform // Tech-Sphere Journal of Pure and Applied Sciences (TSJPAS). 2024. Vol. 1, No. 1. P. 1–12. URL: https://doi.org/10.5281/zenodo.13856860

6. Alotaibi B. A Survey on Industrial Internet of Things Security: Requirements, Attacks, AI-Based Solutions, and Edge Computing Opportunities // Sensors. 2023. Vol. 23, No. 17. P. 7470. URL: https://doi.org/10.3390/s23177470

7. Miner, D., Shook, A. MapReduce Design Patterns. Beijing: O'Reilly, 2012.

8. Holmes, A. Hadoop in Practice. 2nd ed. Shelter Island, NY: Manning, 2015. 513 p.

9. Chambers B., Zaharia M. Spark: The Definitive Guide: Big Data Processing Made Simple. Sebastopol : O'Reilly Media, 2018. 607 p.

10. Damji J., Wenig B., Das T., Lee D. Learning Spark: Lightning-Fast Data Analytics. 2nd ed. Beijing : O'Reilly Media, 2020. 350 p.

11. Apache Spark Tutorial with Examples. URL: https://sparkbyexamples.com/

12. Haines S. Modern Data Engineering with Apache Spark. San Jose, CA: Apress Media, 2022. 592 p.

13. Choudhry M. D., Jeevanandham S., Rose B., Mol P. S. Machine Learning Frameworks for Industrial Internet of Things (IIoT): A Comprehensive Analysis // Proc. 2022 First International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT). Trichy, India, 16–18 Feb. 2022. IEEE, 2022. P. 1–6. Available at: https://doi.org/10.1109/ICEEICT53079.2022.9768630

14. Raschka, S., Mirjalili, V. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. 4th ed. Packt Publishing, 2022. 771 p. URL: https://www.packtpub.com/product/python-machine-learning-fourth-edition/9781801819312

15. Marsland, S. Machine Learning: An Algorithmic Perspective. 2nd ed. Chapman & Hall/CRC, 2015. 452 p.

16. Mohri M., Rostamizadeh A., Talwalkar A. Foundations of Machine Learning. 2nd ed. Cambridge, MA: MIT Press, 2018. 505 p.

17. Черняк О. І., Захарченко П. В. Інтелектуальний аналіз даних: підручник. – Київ: Знання, 2010. – 837 с.