МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

# ГРАНИЧНІ ОБЧИСЛЕННЯ

## Комп'ютерний практикум

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для здобувачів ступеня доктора філософії за освітньо-науковою програмою «Автоматизація та комп'ютерно інтегровані технології» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»*

Київ
КПІ ім. Ігоря Сікорського
2021

Граничні обчислення. Комп’ютерний практикум [Електронний ресурс]: навч. посіб. для аспірантів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / Укладачі: О. В. Степанець, А. С. Захарченко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1749 кБайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 44 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 2 від 09.12.2021 р.)*
*за поданням Вченої ради Теплоенергетичного факультету (протокол № 3 від 26.10.2021 р.)*

Електронне мережне навчальне видання

# ГРАНИЧНІ ОБЧИСЛЕННЯ

## Комп’ютерний практикум

Укладачі: *Степанець Олександр Васильович, канд. техн. наук, доцент*
*Захарченко Анастасія Сергіївна, асистент*

Рецензент: *Ковальчук Артем Михайлович, канд. техн. наук, доцент*

Відповідальний редактор: *Баган Тарас Григорович, канд. техн. наук, доцент*

У навчальному посібнику наведено базовий теоретичний матеріал та практичні приклади концепції граничних обчислень. Посібник містить завдання для практичного виконання, здійснення яких дозволить студентам оволодіти методами та техніками граничних обчислень у їх застосуванні до автоматизації технологічних процесів, а також питання для самоперевірки.

Посібник призначений для аспірантів, які навчаються за спеціальністю 151 "Автоматизація та комп’ютерно інтегровані технології".

# Зміст

# Лабораторна робота №1

# Розробка архітектури рішення

**Мета:** отримати практичні навички проектування архітектури системи автоматизації відповідно до особливостей об’єкта автоматизації та впровадження граничних обчислень для розподілення розрахунків і оптимізації роботи системи.

## 1.1 Основні теоретичні відомості

Граничні обчислення — це підхід, що використовує архітектуру розподілених обчислень, яка наближає обчислення та зберігання даних безпосередньо до джерел даних. Граничні обчислення дозволяють перемістити можливості хмарних обчислень у в великих дата-центрах ближче до виробництва. Обробка даних відбувається або всередині пристрою, що генерує дані (IoT-пристрої), або поблизу нього, на окремому девайсі, в результаті чого на центральний сервер надходить менше даних. Використання граничних обчислень призводить до:

- покращення часу реакції системи;
- зменшення навантаження на мережу;
- покращення надійності системи;
- дотримання вимог про конфіденційність і безпеки даних;
- зменшення експлуатаційних витрат.

Основна відмінність між граничними та хмарними обчисленнями полягає в тому, де відбувається обробка даних. Для хмарних обчислень характерна централізоана обробка даних в хмарних дата-центрах.

Для граничних обчислень характерна робота в локальній мережі на межі мережі, обробляючи, фільтруючи, узагальнюючи дані та готуючи їх передачі до хмарних сервісів (рис. 1.1). Граничні обчислення розподілені в залежності від локації розміщення польових пристрої, їх функціонального призначення,

пріоритету даних тощо. Вони ідеально підходять для обчислення даних, чутливих до часу обробки та захищають систему від тимчасової втрати зв'язку з хмарою.


Рис. 1.1 Схема логічної архітектури граничних обчислень

Однак граничні обчислення не є заміною хмарі. Граничні обчислення доповнюють хмарні сервіси, і можуть забезпечити кращу продуктивність для конкретних випадків використання.

Границя визначається скоріше як логічний рівень, а не певний фізичний розподіл, тому вона відкрита для інтерпретації розробником її програмного, логічного та технічної реалізації відповідно до потреб конкретного проекту і конкретного рішення.

Розробляючи комплексне рішення, що відповідає сучасним тенденція цифровізації виробництва, використовуючи багаторівневу архітектуру потрібно розробити структуру майбутньої системи, що може включати:

- Фізичний рівень - включає в себе датчики, виконавчі механізми, контролери тощо.
- Мережеве з’єднання – визначає протоколи і інтерфейси передачі даних між різними девайсами та сервісами.
- Обробка даних – включає програмне та технічне забезпечення для обробки подій та аналізу даних, їх групування, фільтрування тощо.
- Зберігання інформації – використання хмарних та локальних баз даних.
- Представлення Інформації – реалізація людинно-машинного інтерфейсу.
- Сервісне забезпечення – використання хмарних сервісів аналітики, програмних платформ тощо.
- Бізнес аналітика та планування ресурсів підприємства.

## 1.2 Індивідуальне завдання

Розробити архітектуру комплексного рішення системи автоматизації з використанням граничних обчислень відповідно до теми наукової роботи аспіранта в вигляді схеми, що включає фізичну та програмну реалізацію на локальному рівні, мережеві з’єднання, програмні пакети чи сервіси для збереження даних та реалізацію хмарних обчислень.

## 1.3 Контрольні питання

1. Що таке граничні обчислення?
2. Що таке хмарні обчислення?
3. Чим відрізняються хмарні і граничні обчислення?
4. Де відбуваються граничні обчислення?
5. Яке програмне та технічне забезпечення ви використали в розробленій системі?

# Лабораторна робота №2

# Зв’язок з польовим рівнем

**Мета**: ознайомитись з правилами і особливостями роботи в середовищі Node-RED для граничних пристоїв; отримати навички написання програм в візуальному редакторі та реалізації зв’язку з польовим рівнем.

## 2.1 Основні теоретичні відомості

### 2.1.1 Встановлення компонентів Node-RED

Node-RED побудований на Node.js та використовує перевагу своєї неблокуючої моделі, керованої подіями. Це робить його ідеальним для роботи з граничними обчисленнями на недорогих апаратних засобах, таких як Raspberry Pi, а також у хмарі.

Потоки, створені в Node-RED, зберігаються за допомогою JSON, який можна легко імпортувати та експортувати для збереження, повторного використання чи передачі коду. Для роботи з Node-RED використовується браузерний редактор, що полегшує з'єднання потоків за допомогою широкого діапазону вузлів у палітрі, які можуть бути розгорнутими в середовищі виконання одним натисканням кнопки.

Програма Node-RED складається з потоків, що представлені в вигляді окремих вкладок в робочій області програми. Також, цей термін використовують для неформального опису набору пов’язаних між собою вузлів, основних будівельний блоків потоку.

Для інсталяції Node-RED на персональному комп’ютері потрібно мати встановлений Node.js. Завантажити її можна з офіційного сайту Node.js: https://nodejs.org/en/

Користувачам операційної системи Windows для встановлення Node-RED потрібно у командному рядку виконати команду:

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

Детальна інструкція для встановлення програмних компонентів Node-Red та інформацію щодо встановлення на інших платформах і операційних системах можна знайти за посиланням: https://nodered.org/docs/getting-started/

### 2.1.2 Знайомство з середовищем Node-RED

Для запуску Node-RED необхідно виконати команду «node-red» у командному рядку та залишити термінал відкритим, щоб Node-RED працював. Після цього потрібно відкрити редактор у браузері ввівши адресу: http://<ip-address>:1880, де вказати ip-адресу ком'ютера, на якому запущено Node-RED. Загальний вигляд вікна редактора можна побачити на рисунку 2.1. Він складається з чотирьох компонентів:

- Заголовок у верхній частині, що містить кнопку Deploy та головне меню.
- Палітра зліва, що містить вузли, доступні для використання.
- Основна робоча область посередині, де створюються потоки.
- Бічна панель праворуч.

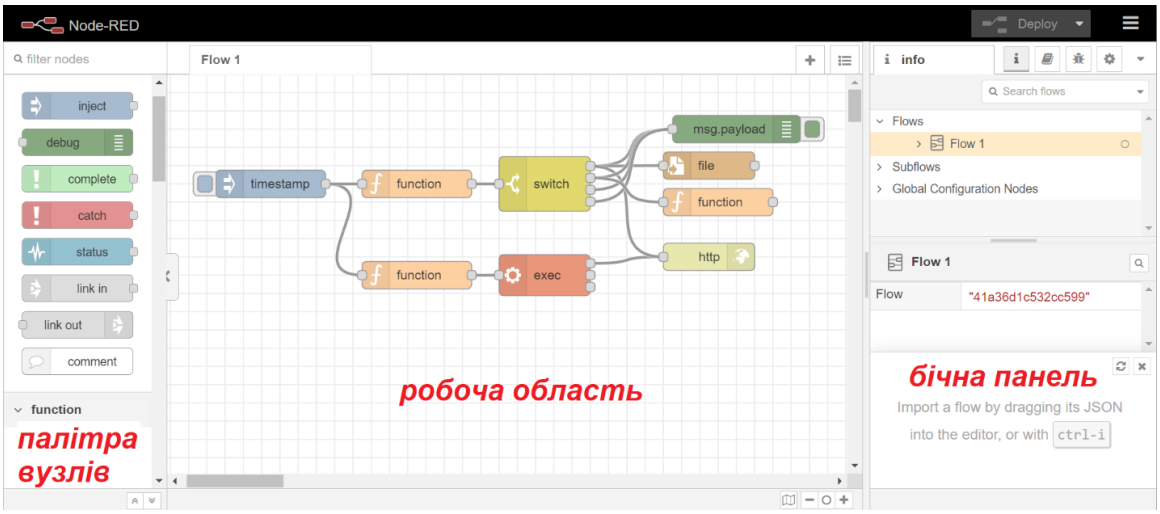


Рис. 2.1 Загальний вигляд редактора Node-RED

Палітра містить усі вузли, які встановлені та доступні для використання. Вони згруповані за кількома категоріями, що включають входи, виходи та функції. Якщо є якісь субпотоки, вони відображаються у верхній частині палітри.

Бічна панель містить вкладки:

- Information - перегляд інформації про вузли
- Debug - перегляд повідомлень, переданих на Debug вузли
- Configuration Nodes - керування вузлами конфігурації
- Context data - перегляд вмісту контексту

Деякі вузли додають власні елементи бічних панелей, наприклад, node-red-dashboard.

Робоча область - це місце, де розробляються потоки. Вона має ряд вкладок, що включають існуючі потоки та відкриті субпотоки.

Вузли можна додати до робочого простору одним із способів:

- витягуючи їх з палітри,
- за допомогою діалогового вікна швидкого додавання,
- імпорту з бібліотеки або буфера обміну.

Вузли запускаються при отриманні повідомлення від попереднього вузла в потоці, або при появі якоїсь зовнішньої події. Вони обробляють це повідомлення або подію і надсилають повідомлення наступним вузлам потоку. Вузли з'єднані між собою інформаційними лініями через свої порти, при цьому вузли можуть мати не більше одного вхідного порту та багатьох вихідних портів, де до кожного порта можуть бути приєднані декілька інформаційних ліній.

Для створення першого тестового потоку необхідно перетягти в робочу область вузли «Inject» та «Debug» та з’єднати їй лінією (рис. 2.2). Після розгортання програми командою Deploy в правому верхньому кутку. Якщо у вузлі є якісь нерозгорнуті зміни, він відображає синє коло над ним. Якщо у його конфігурації є

помилки, він відображає червоний трикутник. Після вдалого розгортання з'являється відповідне повідомлення.


Рис. 2.2 Вигляд тестової програми

Для перевірки роботи програми необхідно відкрити вкладку «Debug» в бічній панелі та натиснути на кнопку вузла «Inject». В результаті має з'явитися повідомлення з часовою міткою. В цьому прикладі перший вузол є ініціатором розрахунку, який формує початкове повідомлення msg. Таким чином потоки Node-RED працюють шляхом передачі повідомлень між вузлами. Повідомлення - це прості об'єкти JavaScript, які можуть мати будь-який набір властивостей. Зазвичай вони включають такі властивості, як:

- topic - використовується для ідентифікації джерела повідомлення або для визначення різних «течій» повідомлень в одному потоці. Також відображається на бічній панелі «Debug» з кожним повідомленням.
- payload – властивість, що, за замовчуванням, містить корисну інформацію, необхідну для роботи більшості вузлів.
- _msgid - це ідентифікатор повідомлення, який можна використовувати для відстеження його проходження через потік.

Значення властивостей повідомлень можуть бути будь-якого типу, що підтримується JavaScript:

- Boolean
- Number
- String
- Array
- Object
- Null

Приклад повідомлення представлений на рисунку 2.3. Для відображення цілого повідомлення в вузлі Debug потрібно змінити налаштування Output з msg.payload, що встановлено за замовчуванням, на «complete msg object»


```
message
TemperatureData : msg : Object
▾object
▾payload: object
    DataType: "Sensor_Data"
    delta_T_CD_h_var: 14.1475584
    delta_T_EV_c_var: 12.42980816
  topic: "TemperatureData"
  _msgid: "1fd691b9.248fae"
▾temperature: array[2]
    0: 14.1475584
    1: 12.42980816
```


Рис. 2.3 Приклад повідомлень у вікні Debug

Для зміни властивостей повідомлення, що передається можна використовувати блок function, що може запускати будь-який код JavaScript. Це дає широкі можливості і гнучкість у роботі з повідомленням, проте вимагає знань JavaScript. В простих завданнях можливе використання блоків change, switch, range, template (рис. 2.4) для пришвидшення роботи, кращої візуалізації потоків та без необхідності писати код.

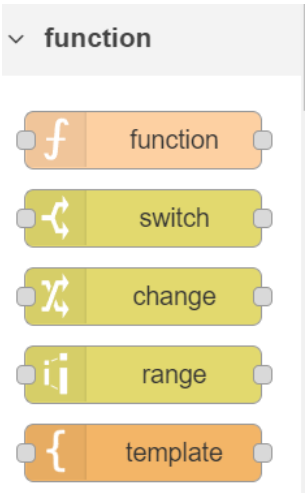


Рис. 2.4 Вузли для редагування повідомлень

### 2.1.3 Пошук модулів

Для пошуку додаткових модулів, необхідних для роботи потрібно викликати меню в правому верхньому куті і обрати пункт “Manage palette” (рис. 2.5).

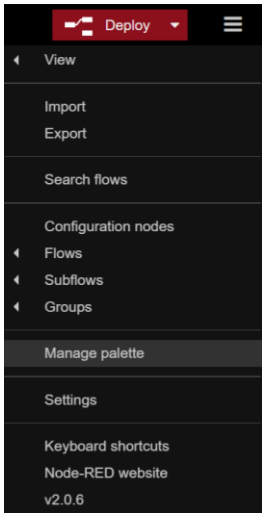


Рис. 2.5 Вікно меню в Node-RED

В вікні, що відкрилося (рис. 2.6), потрібно обрати вкладку “Install” та ввести в пошук ключові слова, наприклад “modbus” та натиснути install навпроти бажаного модуля (node-red-contrib-modbus). В цьому вікні також можна відкрити документацію до модуля, натискаючи на піктограми поруч з назвою модуля.

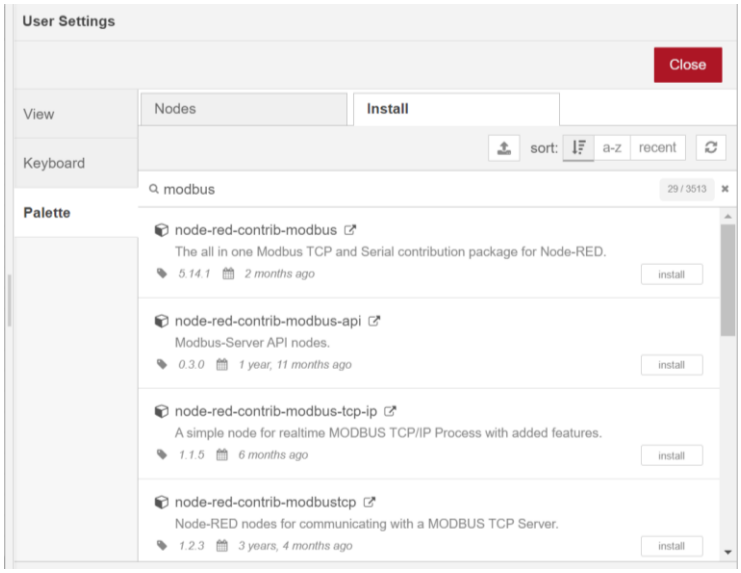


Рис. 2.6 Вікно керування палітрою вузлів в Node-RED

Також пошук модулів можна здійснювати на офіційному сайті Node-RED: https://flows.nodered.org/

### 2.1.4 Зв'язок з польовим рівнем

2.1.4.1 Реалізація зв'язку з польовим рівнем, використовуючи протокол Modbus

Modbus - це комунікаційний протокол обміну повідомленнями, що широко застосовується в промисловості для організації зв'язку типу «клієнт – сервер» між пристроями, підключеними до різних типів шин або мереж. Modbus відноситься до протоколів прикладного рівня мережевої моделі OSI.

Структура пакета для протоколу Modbus представлена на рисунку 2.7 і складається з частини даних протоколу PDU (Protocol Data Unit), однакової для всіх реалізацій протоколу. Для передачі пакетів на конкретних шинах або в мережі до PDU додаються додаткові поля, що відрізняються в залежності від реалізації протоколу. Пакет Modbus вцілому має назву ADU (Application Data Unit).

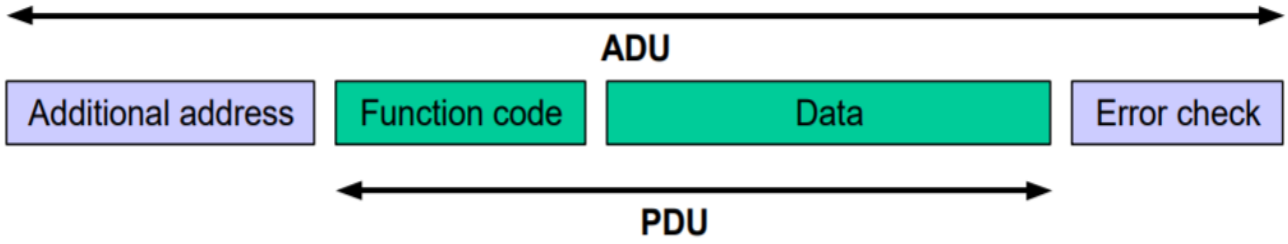


Рис. 2.7 Загальний вигляд пакета Modbus

MODBUS PDU складається з коду функції для зчитування чи запису, яка залежить від виду реєстру, що використовується, моделі даних, що базується на серії таблиць, що мають відмінні характеристики (таблиця 2.1).

З раніше завантаженого модулю використаємо вузол Modbus Server та додамо його до робочого простору (рис. 2.8). Цей вузол використовується для тестування і функціонування TCP-сервера Modbus на основі node-modbus.

Таблиця 2.1 Типи таблиць даних

| Таблиця | Тип елемента | Тип доступу | Опис |
|---|---|---|---|
| Дискретні входи (*Discretes Input*) | 1 bit | Read-Only | Діапазон адрес регістрів: з 10001 по 19999. |
| Регістри котушок (Coils) | 1 bit | Read-Write | Діапазон адрес регістрів: з 20001 по 29999. |
| Регістри вводу (*Input Registers*) | 16-bit word | Read-Only | Діапазон адрес регістрів: з 30001 по 39999. |
| Регістри зберігання (*Holding Registers*) | 16-bit word | Read-Write | Діапазон адрес регістрів: з 40001 по 49999. |

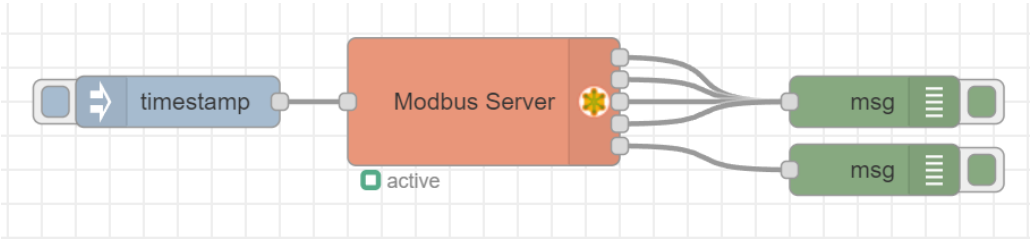


Рис. 2.8 Реалізація потоку з вузлом Modbus Server

Вузли зчитування і запису Modbus потребують конфігурування даних серверу, до якого здійснюється підключення (рис 2.9).

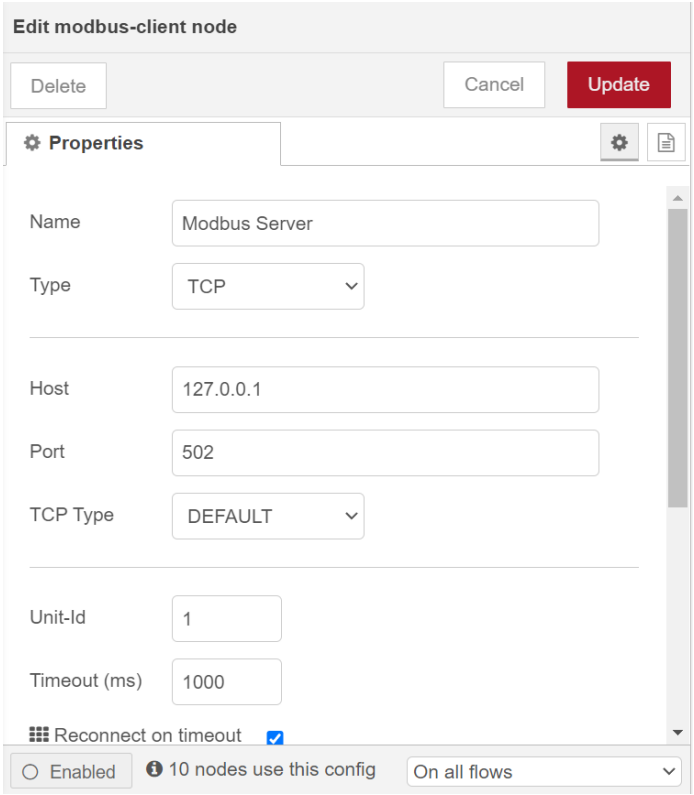


Рис. 2.9 Налаштування конфігураційного вузла modbus-client

Для зчитування даних представлений модуль Node-RED містить вузли:

- *Modbus Read* – використовується для автоматичного зчитування регістрів або котушок із заданою частотою (рис. 2.10 – 2.11).
- *Modbus Getter* – використовується для ручного зчитування регістрів або котушок з наперед визначеними властивостями безпосередньо в вікні налаштувань вузла (рис. 2.10 – 2.11).
- *Modbus Flex Getter* - використовується для ручного зчитування регістрів або котушок відповідно до властивостей, отриманих від попереднього вузла та записаних в msg.payload (рис. 2.11).

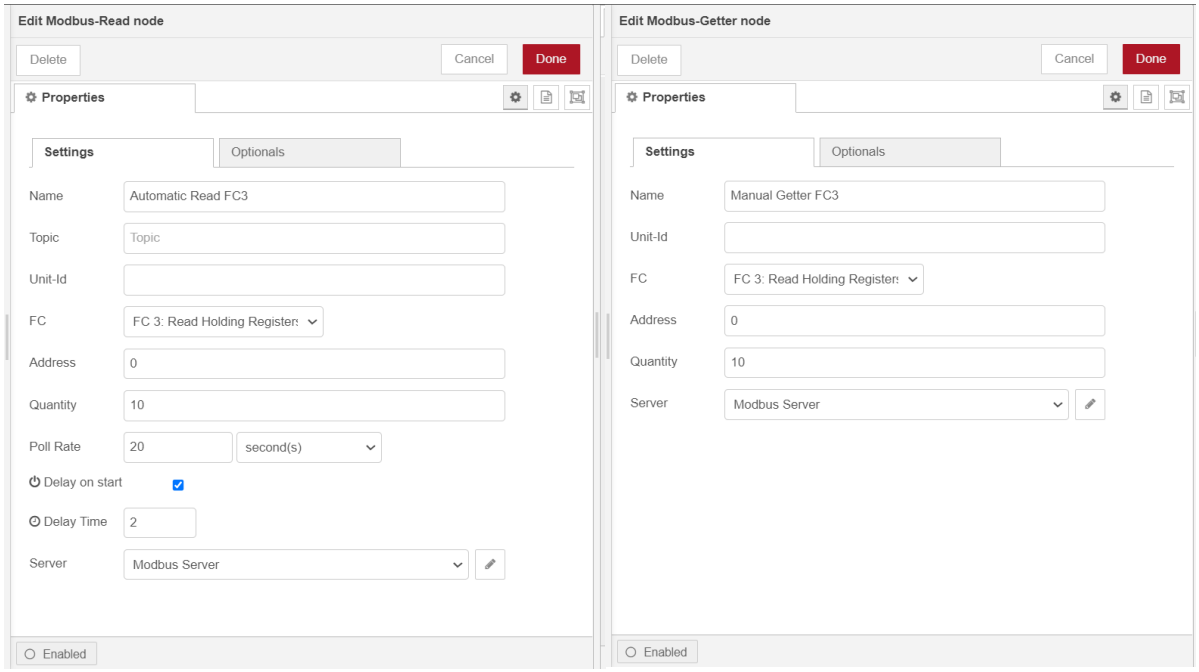


Рис. 2.10 Зовнішній вигляд панелі налашування вузлів Modbus – Read та Modbus – Getter

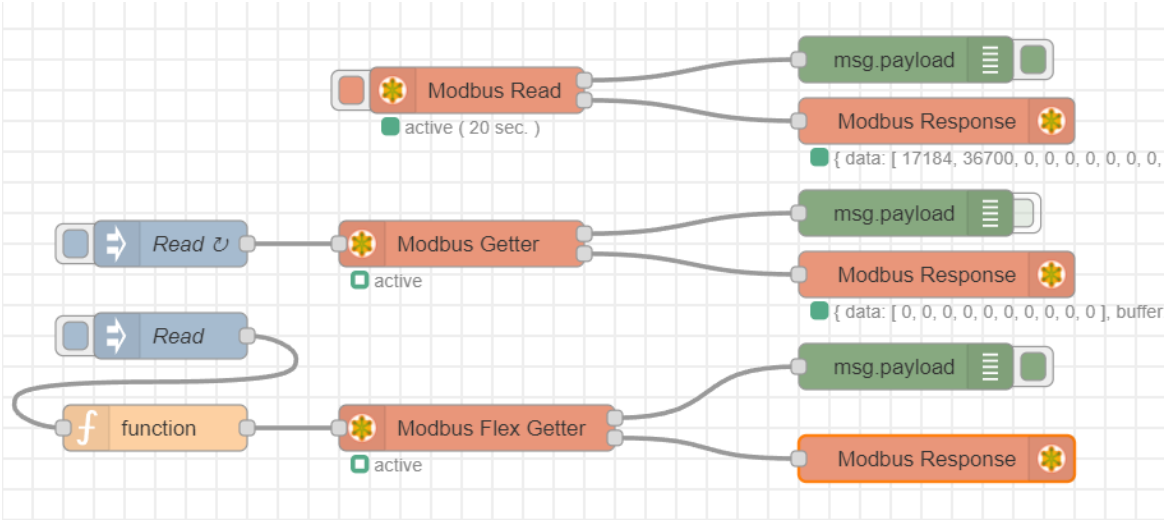


Рис. 2.11 Реалізація зчитування даних за допомогою вузлів Modbus Read, Modbus Getter та Modbus Flex Getter відповідно

Представлений на рисунку вузол function формує повідомлення з властивостями, що є обов’язковим елементом для блоку Modbus Flex Getter, і має наступну структуру:

- unitid (0..255 tcp | 0..247 serial) – ідентифікатор елементу

- fc (1..4) - Код функцій, що включають :
    - FC 1: Зчитування статусу котушки
    - FC 2: Зчитування стану входу
    - FC 3: Зчитування реєстрів зберігання
    - FC 4: Зчитування вхідних реєстрів
- address (0:65535) – початкова адреса
- quantity (1:65535) кількість елементів, що мають бути прочитані

Для коректної роботи описані властивості записуються в вигляді структури в полі msg.payload, використовуючи вузли Change, Inject або Function, як представлено в прикладі:

```
msg.payload = { input: msg.payload, 'fc': 3, 'unitid': 1, 'address':
0 , 'quantity': 10 };
return msg;
```

Вузли передбачають, зчитування 16-бітних регістрів. В випадку використання інших типів потрібно використовувати буфер для перетворення типів. Нижче приведений приклад для зчитування значень з плаваючою точкою:

```
const buf = Buffer.from(msg.payload.buffer);
const value = buf.readFloatBE();
msg.value = value;
return msg;
```

Для запису використовуються вузли Modbus Write і Modbus Flex Write. Як і у випадку читання пакетів, Modbus Write конфігурується безпосередньо в вікні налаштування вузла, а Modbus Flex Write отримує властивості повідомленням, сформованим попередніми вузлами (рис. 2.12).

Для запису використовуються наступні коди функцій:

- FC 5: запис значення одної котушки (Force Single Coil)
- FC 6: запис значення в один регістр зберігання (Preset Single Register)
- FC 15: запис значень в кілька регістрів котушок (Force Multiple Coils)
- FC 16: запис значень в кілька регістрів зберігання (Preset Multiple Registers)

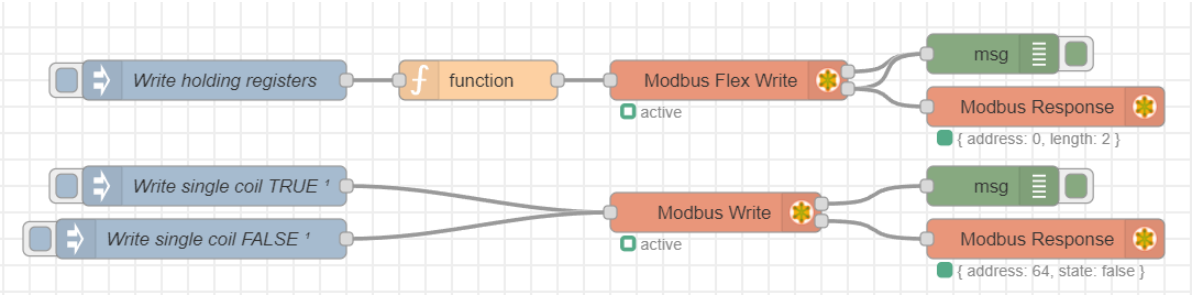


Рис. 2.12 Приклад реалізація потоків з вузлами Modbus Write і Modbus Flex Write

При записі в пакети інформації потрібно пам’ятати про використання 16-бітних регістрів. Приклад обчислення значень реєстрів для числа з плаваючою точкою і формування структури msg.payload приведений нижче:

```
buf=Buffer.alloc(4);
buf.writeFloatBE(msg.payload.value);
var values=[(buf[0]*256+buf[1]),(buf[2]*256)+buf[3]]
msg.slave_ip="192.168.1.31";
msg.payload={"value":values  ,  'fc':  msg.payload.fc,
    'unitid': msg.payload.id, 'address': msg.payload.sa
    , 'quantity': msg.payload.addresses };
return msg;
```

2.1.4.2 Реалізація зв’язку з польовим рівнем, використовуючи OPC UA

OPC UA - це безпечний, відкритий, надійний механізм передачі інформації між серверами та клієнтами. Він широко використовується в промисловій автоматизації.

Для симуляції роботи OPC UA сервера пропонується використати програму Unified Automation - OPC UA C++ Demo Server або MatrikonOPC Simulation Server для роботи в операційній системі Windows.

Розглянемо вузли для роботи з OPC UA в Node-RED, що належать до модулю node-red-contrib-opcua.

OPC UA Client – вузол, що використовується для зв’язку з сервером і виконує наступні дії:

- Read
- Write
- Browse
- Subscribe
- Unsubscribe
- Deletesubscription
- Event
- Info
- Build
- Monitor
- Read Multiple
- Register
- Unregister
- Acknowledge

Загальний вигляд вікна налаштування даного вузла представлений на рисунку 2.13. Воно включає вибір конфігураційного вузла Endpoint, дії, що виконуватиметься та сертифікатів. Для підключення до сервера клієнту потрібна інформація про мережеву адресу, протокол та параметри безпеки.

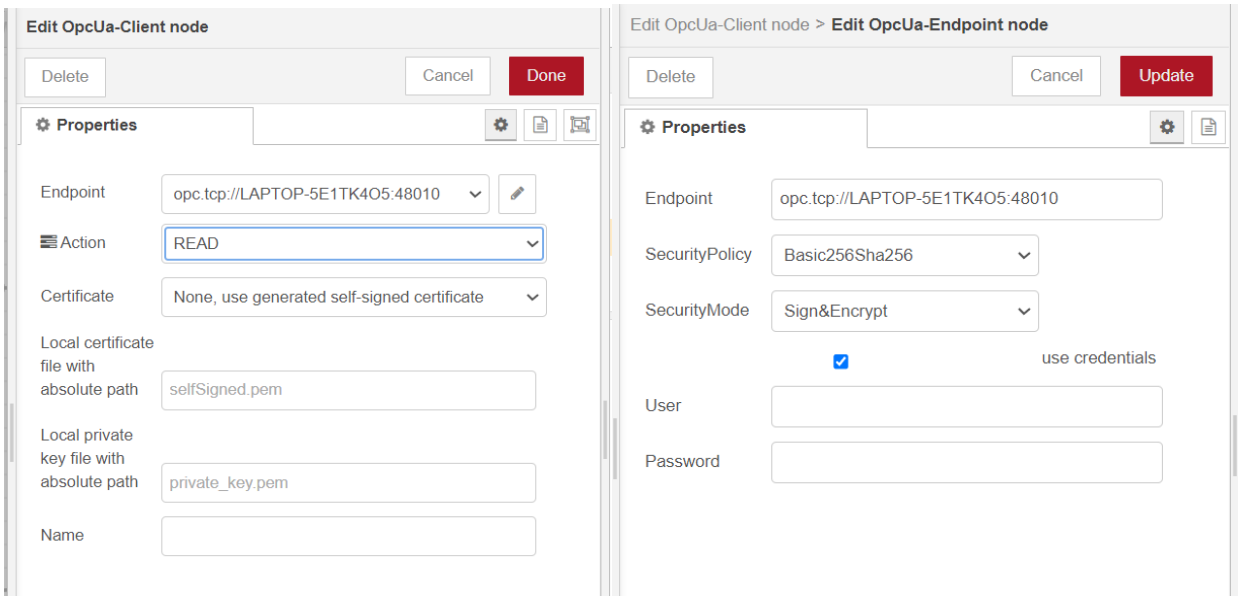


Рис. 2.13 Вигляд вікон налаштування вузлів OPC UA Client і OPC UA Endpoint

Для використання OPC UA Client в режимі читання або запису в вікні налаштування в властивості Action обирається відповідний пункт. Для виконання запиту потрібно додати в тему повідомлення адресу (NodeId) та тип даних, що

зчитуються або записуються (рис. 2.14). В випадку запису значення, його значення вказують в msg.payload.

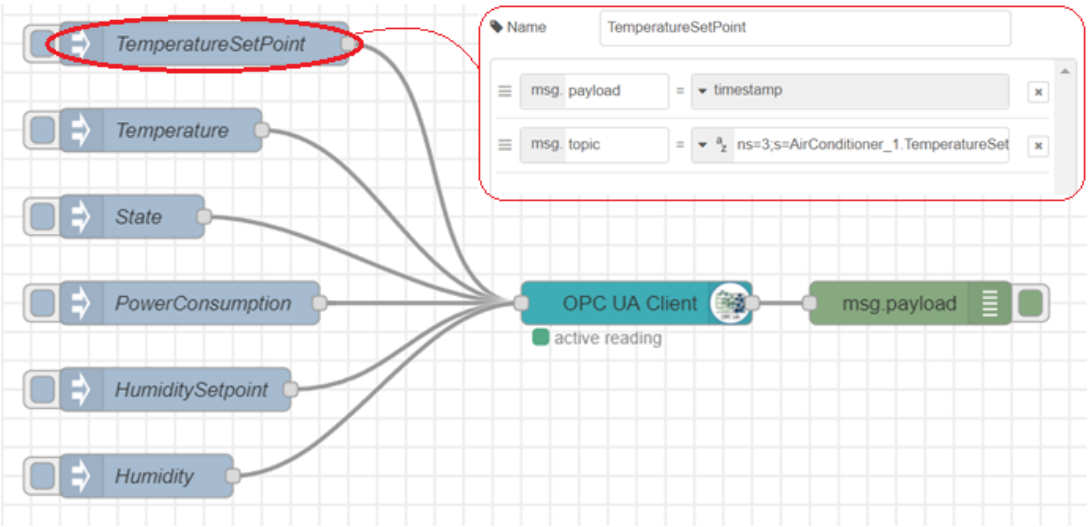


Рис. 2.14 Приклад потоку для запису даних з використанням вузла Inject

Для роботи з клієнтом також можна використовувати вузол OPC UA Item (рис. 2.15), що включає інформацію про адресу, тип даних та їх значення. В випадку відсутності даних в полі Value, використовує дані, що містяться в повідомленні msg.payload.

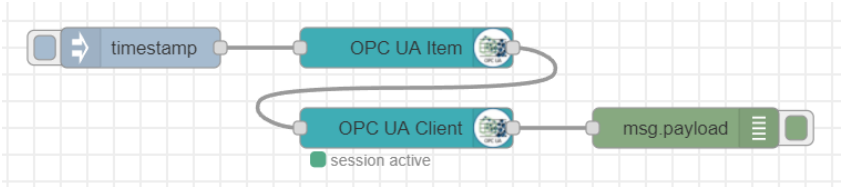


Рис. 2.15 Приклад потоку для запису даних з використанням вузла OPC UA Item

Acknowledge в OPC UA Client може бути використано для підтвердження події чи стану. В такому випадку повідомлення має містити тему (alarm nodeId), conditionId (ідентифікатор вузла eventId) та коментар.

Для створення підписки (Subscribe) потрібно вказати інтервал публікації (Interval) та одноразово запустити вузол Inject для кожного відстежуваного

значення. Для зміни значення інтервалу вибірки (за замовчуванням – 100 мс), потрібно задати нове значення в msg.payload.

Функція Monitor відрізняється від описаного вище Subscribe наявністю додаткових параметрів, таких як абсолютне або відсоткове значення та значення нечутливості.

Множинне зчитування (Read Multiple) спочатку зберігає всі вхідні значення nodeId в масиві. Читання виконується після отримання команди readmultiple з вхідного повідомлення. Якщо вхідне повідомлення має команду clearitems, тоді масив nodeId очищається.

Функція Browse надає весь адресний простір починаючи із введеної msg.topic nodeId. В випадку якщо msg.collect == true, то результат буде зібрано в одне повідомлення в вигляді масиву.

Окрім роботи в якості клієнта Node-RED дозволяє розгорнути власний OPC UA сервер із власними змінними, структурами об’єктів та методами для кінцевої точки: opc.tcp: // localhost: Port/ResourcePath/ за допомогою вузла OPC UA Server (рис 2.16).

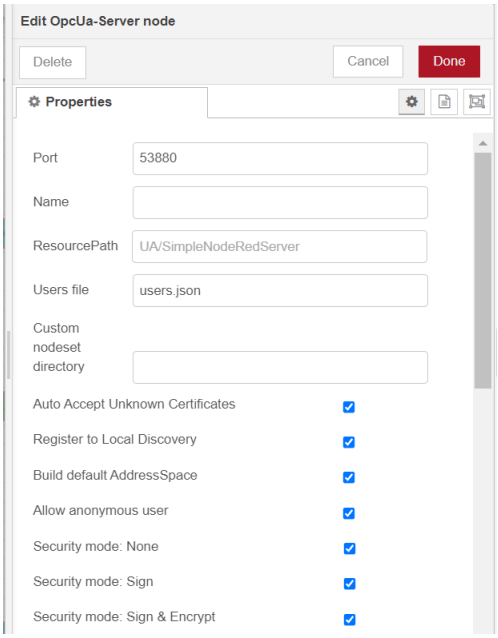


Рис. 2.16 Вікно налаштування вузла OPC UA Server

Команди сервера OPC UA:

- restartOPCUAServer
- addEquipment
- addPhysicalAsset
- setFolder
- addFolder
- addVariable
- installHistorian
- installDiscreteAlarm
- registerNamespace
- getNamespaceIndex
- getNamespaces
- setUsers

Приклади запису JSON в msg.payload для вузла Injects:

```
{ "opcuaCommand": "addEquipment",  "nodeName": "Machine" }
{ "opcuaCommand": "addEquipment",  "nodeName": "Machine" }
{ "opcuaCommand": "deleteNode",  "nodeId": "ns=4;s=PhysicalAsset2" }
```

## 2.2 Індивідуальне завдання

Реалізувати симуляцію або фізичне підключення до польового рівня системи, відповідно до теми накових досліджень аспіранта та розробленої архітектури рішення; розробити програму, використовуючи Node-RED, що виконує підключення до польового рівня системи та зробити висновки про ефективність і роботу запропонованого рішення.

## 2.3 Контрольні питання

1. Що таке Node-RED?
2. Які принципи роботи в редакторі Node-RED?
3. Які основні вузли ви використовувати для виконання лабораторної? Поясніть принципи їх роботи.
4. Які принципи функціонування протоколу Modbus?
5. Як реалізована робота з протоколом Modbus в Node-RED?
6. Які принципи роботи технології OPC UA?
7. Як реалізована комунікація OPC UA клієнта в Node-RED?

# Лабораторна робота №3

# Зберігання та обробка даних

**Мета**: ознайомитися з інструментами Node-RED для роботи з базами даних, розглянути особливості підготовки даних в Node-RED для запису і зчитування інформації та роботи із SQL – запитами.

## 3.1 Основні теоретичні відомості

### 3.1.1 Робота з реляційними базами даних

В якості прикладу використаємо MySQL або MariaDB Server, що є одними з найпопулярніших у світі реляційних баз даних з відкритим вихідним кодом, доступною для використання на операційних системах Windows, Linux і Mac OS. Завантажити програмне забезпечення можна за лінком:

- https://mariadb.org/download/
- https://dev.mysql.com/downloads/mysql/

Для перевірки роботи бази даних використаємо клієнтську утиліту HeidiSQL, що встановлюється разом з MariaDB, або MySQL Workbench для MySQL. Для керування базою даних з HeidiSQL або MySQL Workbench, користувач під’єднується до локального або віддаленого серверу та відкривають сесію, використовуючи потрібний логін і пароль.

Для роботи з реляційними базами даних потрібно знання SQL (Structured Query Language) – мови програмування, що використовується для формування запитів, оновлення та адміністрування баз даних.

Сучасні системи керування базами даних (СКБД) можуть використовувати власні видозміни мови SQL, але основні принципи формування SQL-запитів відповідають стандартам ANSI та ISO.

### 3.1.2 Особливості роботи з базами даних в Node-RED

В Node-RED для роботи з запропонованими базами даних можна використовувати модуль node-red-node-mysql. Він включає в себе два вузли:

- MySQLdatabase – конфігураційний вузол, містить облікові дані для доступу до бази даних (рис. 3.1)

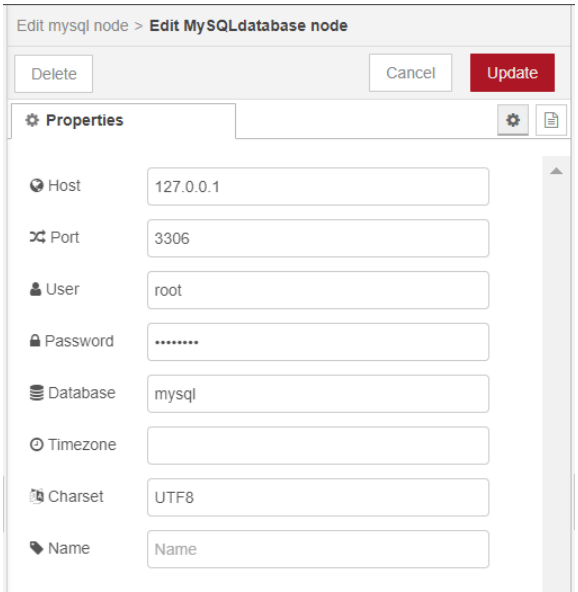


Рис. 3.1 Вікно налаштувань вузла MySQLdatabase

- mysql – надає доступ до бази даних MySQL. Цей вузол здійснює SQL запити до налаштованої бази даних. Недоліком роботи даного модуля в тому, що він дозволяє ін'єкції SQL, що потрібно враховувати при реалізації проектів. Вхідне повідомлення msg.topic має містити запит до бази даних, а результат повертається у msg.payload. Якщо для ключа нічого не знайдено, повертається null.

Для створення бази даних з Node-RED потрібно використовувати вузол MySQLdatabase з підключенням до існуючої бази даних на сервері, наприклад, таких як системна mysql (рис.15). Формування повідомлення може здійснюватися вузлами Inject, Change, Function тощо, в залежності від реалізації, проекту і завдань, що виконуються (рис. 3.2). В msg.topic записуємо запит:

```
CREATE DATABASE `BuildingAutomation`;
```

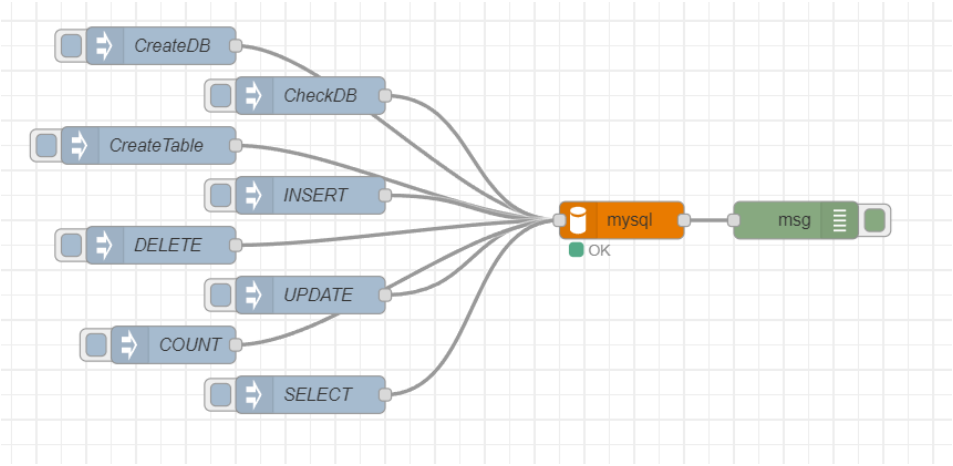


Рис. 3.2 Приклад роботи з базою даних використовуючи вузли mysql та Inject

Для перевірки наявності бази даних можна виконувати наступний запит:

```
SHOW DATABASES LIKE 'BuildingAutomation'
```

Якщо вказана база даних існує, вузол повертає в msg.payload[0].Database (BuildingAutomation) текстову змінну "buildingautomation". Таким чином можна реалізувати перевірку при запуску програми Node-RED на наявність відповідної бази даних і в разі відсутності створити її та підготувати необхідні для роботи таблиці (рис. 3.3).

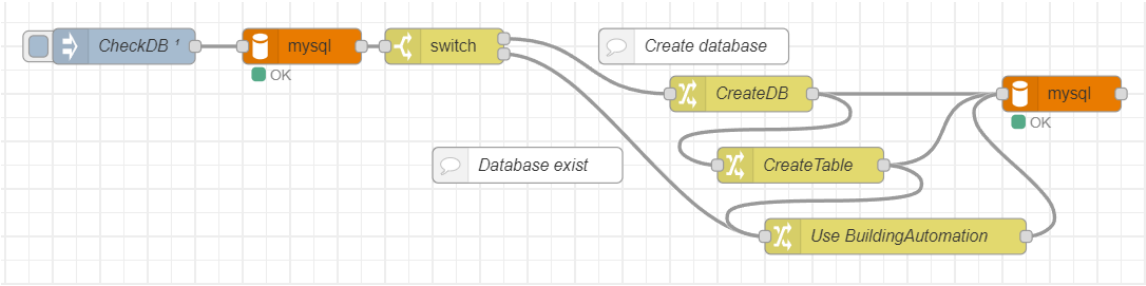


Рис. 3.3 Перевірка наявності бази даних при запуску програми

Щоб вибрати базу даних для роботи, можна використовувати оператор USE:

```
USE BuildingAutomation;
```

Запит для створення таблиці має наступний вигляд:

```
CREATE TABLE `room` (
    `Ts` TIMESTAMP NOT NULL DEFAULT current_timestamp(),
    `Troom` FLOAT NULL DEFAULT NULL,
    `TroomSP` FLOAT NULL DEFAULT NULL,
```

```
    `Humidity` FLOAT NULL DEFAULT NULL,
    `HumiditySP` FLOAT NULL DEFAULT NULL,
PRIMARY KEY (`Ts`) USING BTREE
);
```

Для створеної таблиці запис даних буде мати наступний вигляд:

```
INSERT INTO room (Troom,TroomSP,Humidity,HumiditySP) VALUES ('27.9',
  '28', 58.9', '60');
```

Видалення останнього запису:

```
DELETE FROM `room` ORDER BY TS DESC LIMIT 1;
```

Редагування запису:

```
UPDATE `room` SET `TroomSP`='25' ORDER BY TS DESC LIMIT 1;
UPDATE `room` SET `TroomSP`='29' WHERE TS = '2021-03-19 11:34:34';
```

Використовуючи дані запити можна реалізувати динамічний запис даних в базу даних. В прикладі, на рисунку 3.4, виконується зчитування даних з OPC сервера, використовуючи функцію «Read Multiple».

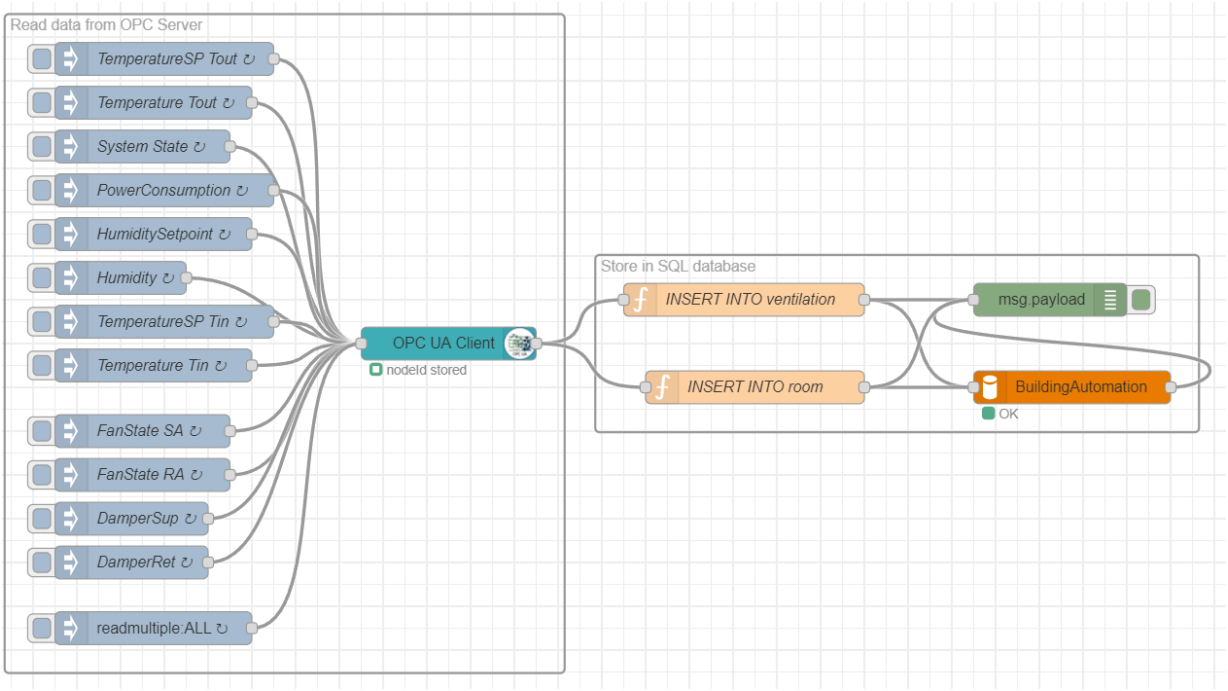


Рис. 3.4 Зчитування даних з OPC серверу та їх запис в базу даних

Це означає, що вузол OPC UA Client збирає всі запити, які пригодять на його вхід і після приходу команди, яка містить значення msg.topic = «readmultiple» і

msg.payload = «ALL», виконує послідовне зчитування всіх значень і формує одне повідомлення в вигляді масиву. Використовуючи вузол function створюємо відповідний запит INSERT:

```
topic = "INSERT INTO room (Troom,TroomSP,Humidity,HumiditySP) VALUES
  ('";
topic = topic + msg.payload[1].value.value +"', '";
topic = topic + msg.payload[0].value.value +"', '";
topic = topic + msg.payload[5].value.value +"', '";
topic = topic + msg.payload[4].value.value + "')";
msg.topic = topic;
return msg;
```

Аналогічно до попередніх прикладів відбувається вибірка даних за допомогою оператора SELECT. Він використовується для отримання рядків, вибраних з однієї або кількох таблиць, і може включати оператори UNION та підзапити. Розглянемо найбільш часто використовувані елементи оператора SELECT:

```
SELECT
    select_expr [, select_expr] ...
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

- Кожен select_expr вказує стовпець, який потрібно отримати. Має бути принаймні один select_expr.
- table_references вказує на таблицю або таблиці, з яких потрібно отримати рядки.
- SELECT підтримує явний вибір розділів за допомогою речення PARTITION зі списком розділів або підрозділів після імені таблиці в table_references. У

цьому випадку рядки вибираються лише з перелічених розділів, а будь -які інші розділи таблиці ігноруються.

- Речення WHERE, якщо воно задане, вказує на умову або умови, яким рядки повинні задовольняти для вибору. where_condition — це вираз, який оцінюється як істина для кожного рядка, який потрібно вибрати. Оператор вибирає всі рядки, якщо немає пропозиції WHERE. У виразі WHERE можна використовувати будь -які функції та оператори, які підтримує MySQL, за винятком агрегатних (групових) функцій.
- Щоб відсортувати обрані дані використовується вираз ORDER BY. За замовчуванням він виконує сортування в порядку зростання. Це можна вказати явно за допомогою ключового слова ASC. Для сортування в зворотному порядку, додається ключове слово DESC (за спаданням) до імені стовпця в вираз ORDER BY.
- Вираз LIMIT можна використовувати для обмеження кількості рядків, які повертає оператор SELECT. LIMIT приймає один або два числові аргументи, які мають бути невід'ємними цілочисельними константами.

Приклад запитів для таблиці room:

```
SELECT * FROM `BuildingAutomation`.`room` ORDER BY TS DESC LIMIT 100;
SELECT Ts, Troom FROM `BuildingAutomation`.`room` ORDER BY TS LIMIT
100;
```

Підрахунок записів в таблиці:

```
SELECT COUNT(*) FROM `room`;
```

Детальну інформацію про роботу з SQL запитами в середовищі MySQL можна отримати за посиланням:

https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html

## 3.2 Індивідуальне завдання

Розробити базу даних відповідно до теми наукової роботи аспіранта та розробленої архітектури рішення для системи автоматизації. Реалізувати запис даних, що отримуються з польового рівня до створеної бази даних, використовуючи програмне середовище Node-RED.

## 3.3 Контрольні питання

1. Розкажіть про особливості роботи з реляційними базами даних.
2. Що таке мова SQL і для чого вона використовується?
3. Поясніть принципи роботи з SQL базами даних в Node-RED?
4. Які вузли ви використовували для роботи базами даних?
5. Поясніть як здійснюється запис даних в базу даних. Наведіть приклад SQL-запиту.
6. Поясніть як здійснюється зчитування даних з бази даних. Наведіть приклад SQL-запиту.

# Лабораторна робота №4

# Людино-машинний інтерфейс

**Мета**: ознайомитись зі способами графічного представлення стану системи на граничних девайсах та взаємодії з користувачами; отримати навички використання графічних елементів Node-RED.

## 4.1 Основні теоретичні відомості

### 4.1.1 Робота з Node-RED Dashboard

Для реалізації людинно-машинного інтерфейсу на базі Node-RED будемо використовувати модуль node-red-dashboard. Цей модуль надає набір вузлів у Node-RED для швидкого створення інформаційної панелі даних у реальному часі.

Стандартно URL-адреса для інформаційної панелі базується на існуючому шляху Node-RED з додаванням /ui. Це можна змінити у файлі Node-RED settings.js.

Макет інформаційної панелі можна представити як сітку, до якої прив’язуються елементи, одиниця якої шириною 48px з 6px пробілом. Налаштування відображення інформаційної панелі знаходяться на бічній панелі dashboard, що містить вкладки:

- Layout – дозволяє додати вкладки та лінки в меню
- Site – включає налаштування заголовку, вибір теми, формату дати та налаштування основної сітки макету вікна в пікселях тощо.
- Theme – відповідає за налаштування кольорової схеми вікна та шрифтів

Модуль Dashboard містить віджети, які можна додавати на інформаційну панель. Кожен віджет розміщується в певній групі за допомогою вікна налаштування, де визначаються його розміри, заголовки та інші властивості. Модуль має наступні вузли-віджети:

- Audio out
- Button
- Chart
- Colour Picker

- Date Picker
- Dropdown
- Form
- Gauge
- Notification
- Numeric
- Slider
- Switch
- Template
- Text
- Text input
- UI-Control

Більшість віджетів мають мітку (lable) і значення (value) – окрім статичного запису в вікні налаштувань, обидва вони можуть бути вказані властивостями вхідного повідомлення, якщо це необхідно, і модифікуватися за допомогою ангулярних виразів. Наприклад, для lable можна встановити значення `{{msg.topic}}`, а значення - `{{value | number:1}}%`, щоб округлити значення до одного знака після коми і додати знак %. Таким чином, кожен вузол може проаналізувати отримане повідомлення msg.payload, щоб зробити його придатним для відображення.

Будь-який віджет можна вимкнути, передавши властивість msg.enabled з значенням false.

Розглянемо декілька прикладів реалізації елементів інформаційної панелі.

Використовуючи вузли Switch розробимо панель керування освітлення в приміщені (рис 4.1). Кожна зміна перемикача буде генерувати повідомлення msg.payload, котре ми записуватимемо до OPC UA серверу.

В вікні налаштування Switch (рис 4.2) можна змінювати групу, що якої входить елемент, розмір та піктограму елемента на панелі, текстовий підпис, спливаюче вікно підказки та налаштовувати генерацію вихідних повідомлень.

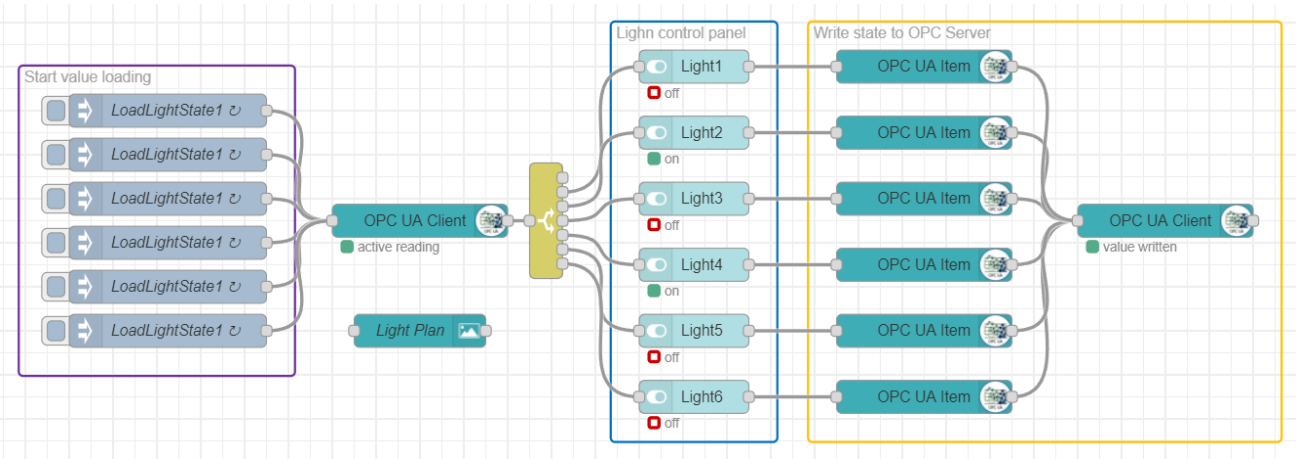


Рис. 4.1 Реалізація панелі освітлення приміщення

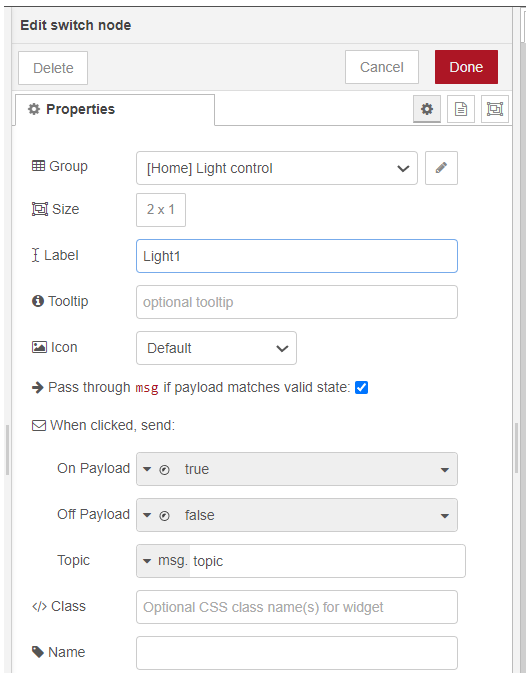


Рис. 4.2 Вікно налаштування вузла Switch

Також стан перемикача можна змінювати програмно, надсилаючи на нього повідомлення з булевим значенням стану, записаним в msg.payload. Для моніторингу ввімкнення та вимкнення освітлення вручну ми зчитуємо поточний стан системи з OPC сервера при завантажені Node-RED та продовжуємо це робити з вказаним інтервалом.

До панелей в Node-RED можна додавати мультимедійні файли. Один з найпростіших варіантів додавання графічних елементів – використання модуля node-red-contrib-ui-media.

Представлені елементи управління на рисунку 4.1 утворюють панель управління освітленням приміщення для користувача (рис 4.3).

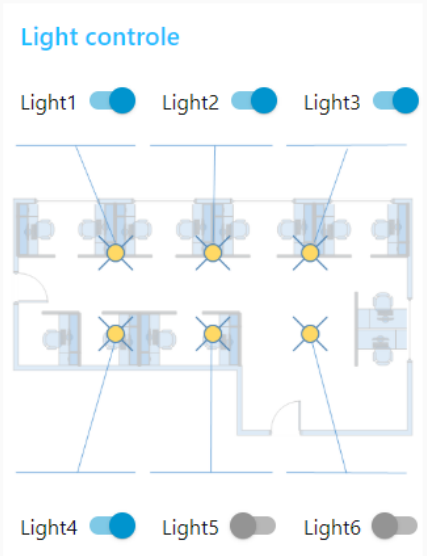


Рис. 4.3 Панель управління освітленням

Щоб додати на панель керування інформацію в вигляді лінійної, стовпчастої або кругової діаграм, використовується  вузол Chart. Цей вузол може отримувати дані в msg.payload динамічно, в процесі роботи системи і кожне вхідне значення буде перетворено в число. Якщо перетворення не вдається, повідомлення ігнорується.

Також вузол Chart може отримувати масив даних для відображення. Це зручно використовувати для представлення історичних трендів (рис. 4.4).

В такому випадку дані потрібно привести до вигляду:

```
msg.payload.data: [
    [{ "x": {timestamp}, "y": {data} },  // Серія 1
     { "x": {timestamp},, "y": {data} },
```

```
        ],
        [{ "x": {timestamp},, "y": {data} }, // Серія 2
         { "x": {timestamp},, "y": {data} },
        ],
…
]
```

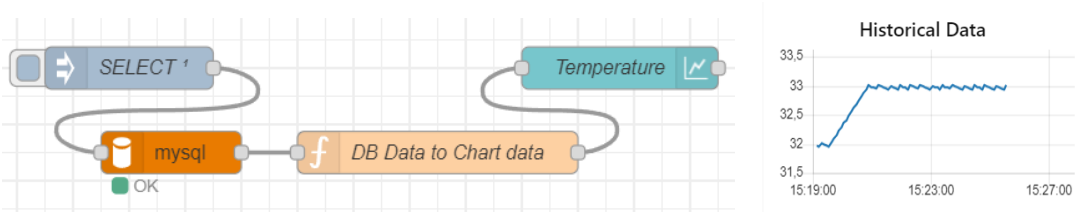


Рис. 4.4 Використання вузла Chart для відображення історичних трендів

Мінімальні та максимальні значення осі Y є необов’язковими. Графік автоматично масштабується до будь-яких отриманих значень. Декілька серій можна відобразити на одній діаграмі, використовуючи різне значення msg.topic для кожного вхідного повідомлення. Для конфігурування вісі X вказується максимальний часовий проміжок або максимальну кількість точок для відображення. Інші дані будуть автоматично видалятися з графіка.

Вихід вузла містить масив стану діаграми, який можна зберегти, якщо потрібно і використовувати для повторного відображення даних.

Окрім виводу даних в вигляді графіків, можна використовувати вузол Gauge для представлення даних у вигляді датчиків, або текстовий вивід даних, що можуть форматуватися за допомогою ангулярних виразів.

Для отримання даних від користувача можна використовувати вузли Slider, Form, Text input, Date picker, DropDown. При введені даних користувачем ці вузли відправляють вихідне повідомлення з відповідним значенням в msg.payload.

Віджет вузла Template може містити будь-який дійсний код html. Цей вузол можна використовувати для створення елементів динамічного інтерфейсу

користувача, який змінює свій вигляд на основі вхідного повідомлення і може надсилати повідомлення до Node-RED.

Для прикладу представимо реалізацію панелі керування панелі керування технологічними системами будинку, що складається з елементів, розглянутих раніше (рис. 2.5-2.6)

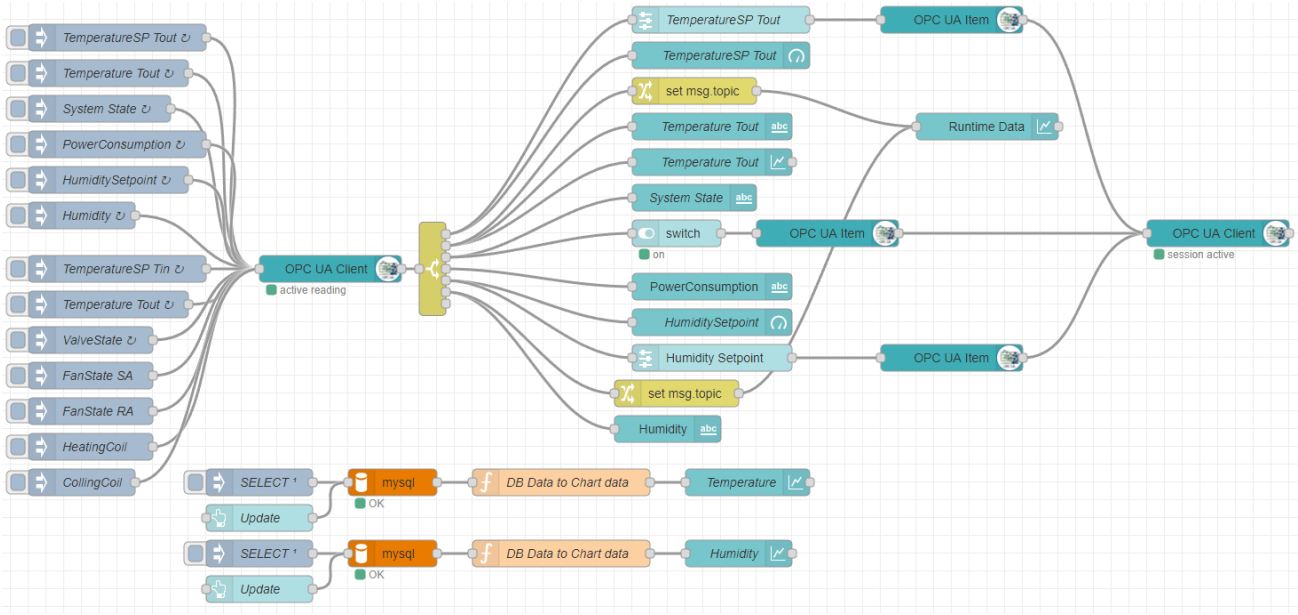


Рис. 4.5 Реалізація панелі керування технологічними системами будинку

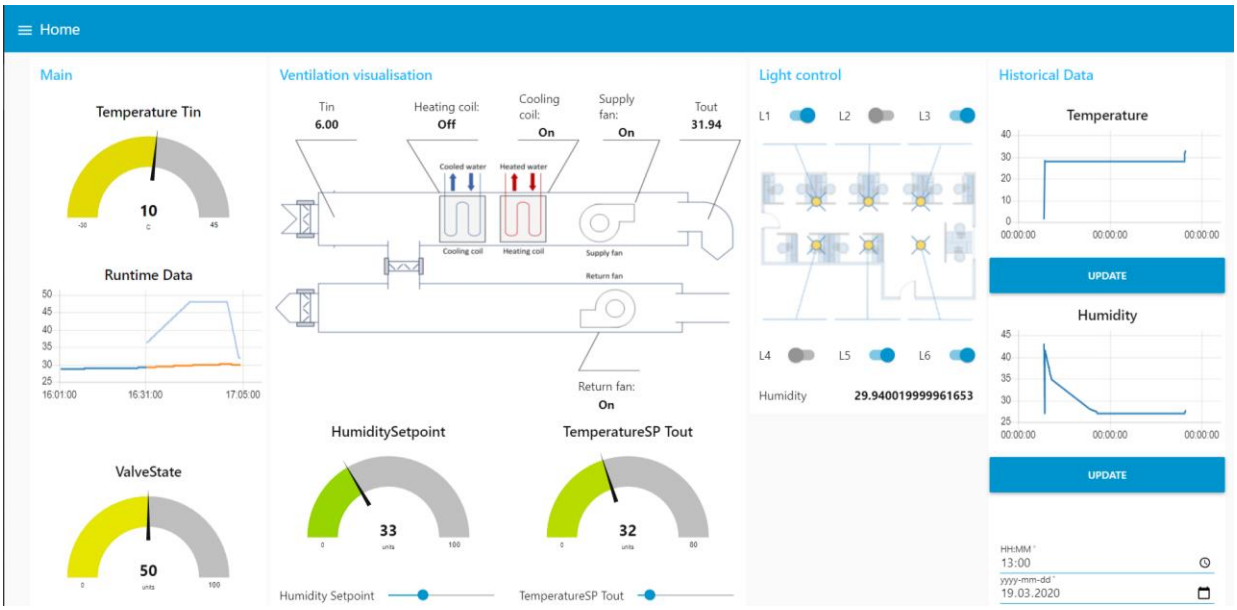


Рис. 4.6 Панель керування технологічними системами будинку

### 4.1.2 Використання додаткових вузлів для роботи з Dashboard

Окрім запропоновонах віджетів в модулі dashboard, на сьогоднішній день розроблена велика кількість dashboard-сумісних модулів розроблених іншими користувачами. Їх можна шукати на офіційному сайті Node-RED flows, вводячи пошуковий запит «node-ui-» або «contrib-ui-».

## 4.2 Індивідуальне завдання

Розробити панель керування, використовуючи модуль Dashboard в Node-RED, відповідно до розробленої архітектури системи автоматизації. Реалізувати зв'язок елементів віджетів з польовим рівнем та базою даних для представлення історичних трендів.

## 4.3 Контрольні питання

1. Що таке людино-машинний інтерфейс?
2. Розкажіть про призначення модуля "dashboard".
3. Як відбувається розміщення віджетів на панелі керування?
4. Які вузли ви використовували для розробки людино-машинний інтерфейс в Node-RED?
5. Яке призначення вузла Template в Node-RED?
6. Які вузли використовуються для отримання інформації від користувача?

# Лабораторна робота №5

# Зв’язок з хмарою. Маніпуляція з даними в хмарі

**Мета**: отримати практичні навички роботи з хмарними застосунками та навчитися передавати дані з граничних пристроїв до хмарних сервісів.

## 5.1 Основні теоретичні відомості

### 5.1.1 Зв’язок з хмарними сервісами

В якості хмарного середовища використаємо IBM Cloud, що можна знайти за посиланням:

https://cloud.ibm.com/

Для роботи з хмарою необхідно зареєструватися, вказуючи свої особисті дані. Після успішної реєстрації відкриється вікно Dashboard, що дозволяє керувати сервісами, на які користувач підписаний.

Для роботи в хмарі з середовищем Node-RED, потрібно перейти на вкладку “Catalog” і в полі пошуку ввести запит “Node-red app”. Відкриється вікно, що дозволить створити попередньо налаштовану програму Node-RED, включаючи службу Cloudant для зберігання конфігурації потоку програми.

Візуально вікно Node-RED в IBM Cloud не відрізняється від розглянутого раніше, проте містить інший набір вузлів в палітрі.

Для зв’язку між хмарним застосунком та програмою на граничному девайсі будемо використовувати WebSocket —протокол, що призначений для обміну інформацією в режимі реального часу. В Node-RED для цієї мети можна використовувати вузли

- websocket in – отримує дані, отримані через WebSocket і записує їх до msg.payload.
- websocket out – надсилає msg.payload через WebSocket.

- websocket-client - підключає клієнта WebSocket до вказаної URL-адреси.
- websocket-listener - створює кінцеву точку сервера WebSocket, використовуючи вказаний шлях.

Для налаштування зв'язку додамо вузол websocket in в робочу область і в вікні налаштування виберемо тип "Listen on", а в пункті шлях оберемо варіант "Add new websocket-listener" (рис. 5.1).

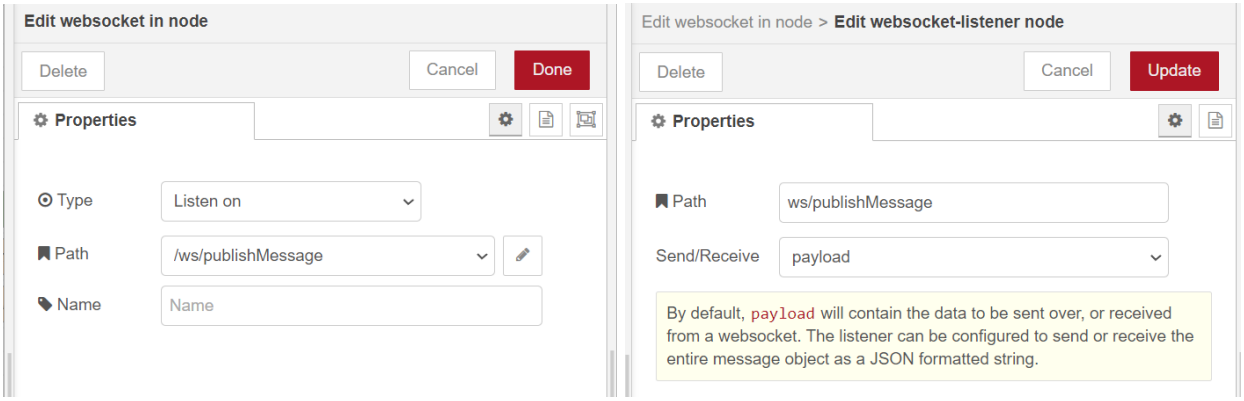


Рис. 5.1 Вікна налаштувань вузлів websocket in та websocket-listener відповідно

Аналогічно налаштуємо websocket на граничному девайсі. В якості шляху введемо url адресу Node-RED застосунку, включно з шляхом, вказаним в вікні налаштування websocket-listener, наприклад:

```
ws://node-red-pbgxi-2021-03-11.eu-gb.mybluemix.net/ws/publishMessage
```

Для відправки даних до хмари (рис. 5.2) сформуємо вихідне повідомлення msg.payload в вигляді JSON структури за допомогою вузла function:

```
msg.payload ={
    "DataType": "room",
     "Troom":msg.payload[1].value.value,
    "TroomSP":msg.payload[0].value.value,
    "Humidity":msg.payload[5].value.value,
    "HumiditySP":msg.payload[4].value.value
}
return msg;
```

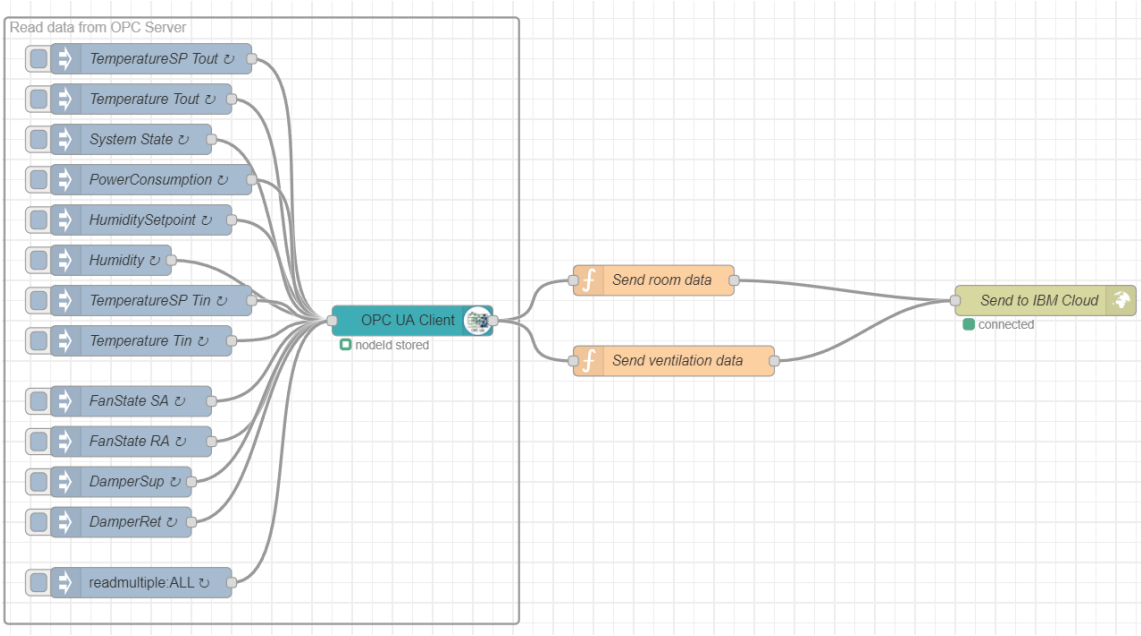


Рис. 5.2 Реалізація зв'язку з IBM Cloud

З боку хмари ми отримуємо повідомлення в вигляді JSON структур. Для обробки різних наборів даних в вихідне повідомлення було додано елемент “DataType”, за значенням якого можна розподіляти повідомлення по різних потоках.

### 5.1.2 Підключення до хмарних баз даних

Для запису даних використаємо базу даних часових рядів InfluxDB Cloud, що можна знайти за посиланням:

https://www.influxdata.com/products/influxdb-cloud/

Створивши акаунт, для роботи потрібно перейти на вкладку Data і обрати пункт Buckets, натиснути кнопку “Create new bucket” та ввести назву бази даних.

В вкладці API-tokens натиснути на кнопку “Generate API Token” та обрати створену базу даних.

В Node-Red IBM Cloud додамо вузол influxdb batch з модуля node-red-contrib-influxdb (рис. 5.3). В вікні налаштування оберемо нове підключення до

сервера. В якості URL введемо адресу веб-клієнта InfluxDB, з яким працювали раніше та створений token.

В якості організації потрібно ввести ідентифікатор, який можна знайти в вікні Users, вкладці About веб-клієнта InfluxDB

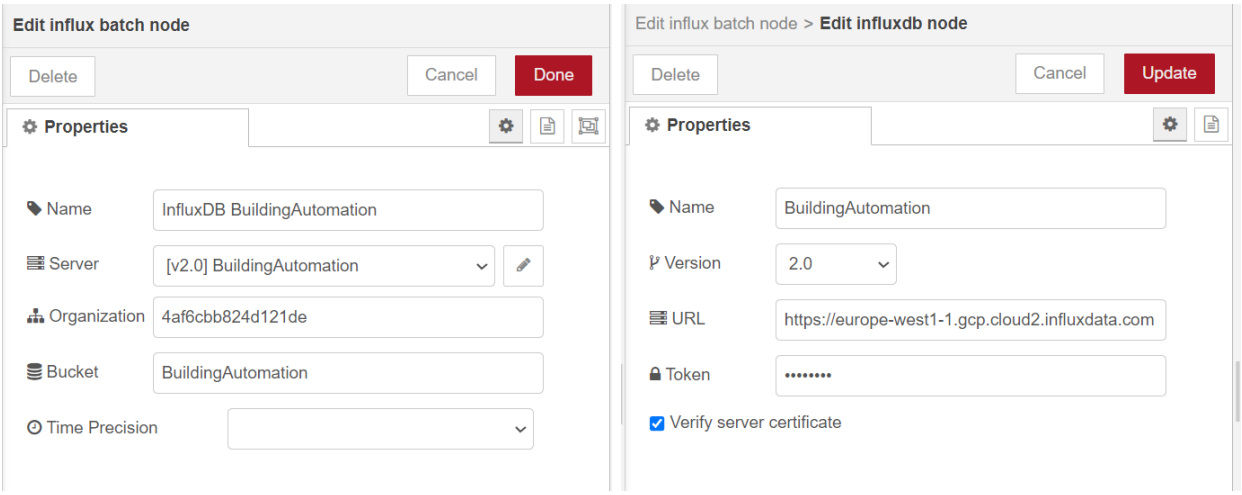


Рис. 5.3 Вікна налаштування вузлів influx batch node та influxdb

Flux — це функціональна скриптова мова в InfluxData, призначена для запитів, аналізу та дії з даними.

Для запису в InfluxDB (рис. 5.4) потрібно підготувати msg.payload в вигляді структури:

```
msg.payload = [
    {
        measurement: "RoomTemperature",
        tags:{
            UnitType:"Room",
            UnitID:"ID1",
            Device: "Sensor",
            DeviceID: "Troom"
        },
        fields: {
            Temperature: msg.payload.Troom,
            TemperatureSP:msg.payload.TroomSP
```

```
            },
            timestamp: new Date()
        },
        {
            measurement: "RoomHumitidy",
            tags:{
                UnitType:"Room",
                UnitID:"ID1",
                Device: "Sensor",
                DeviceID: "Hroom"
            },
            fields: {
                Humidity: msg.payload.Humidity,
                HumiditySP: msg.payload.HumiditySP
            },
            timestamp: new Date()
        }
    ];
    return msg;
```

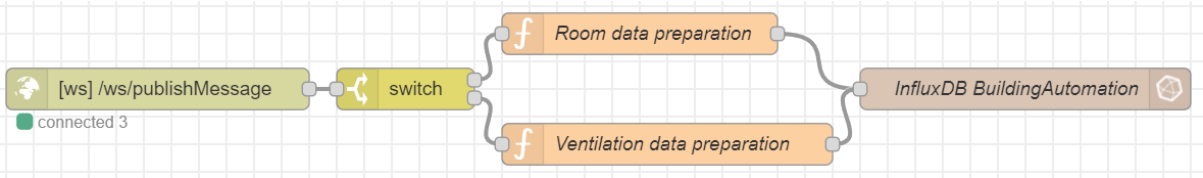


Рис. 5.4 Реалізація запису в InfluxDB

Для зчитування даних з InfluxDB сформуємо повідомлення msg.query, використовуючи вузол function:

```
msg.query = [
  'from(bucket: "BuildingAutomation")',
  '  |> range(start: -1d, stop: now())',
  '  |> filter(fn: (r) => r._measurement == "RoomHumidity")',
  '  |> filter(fn: (r) => r._field == "Humidity")',
  '  |> yield(name: "1")',
```

```
    'from(bucket: "BuildingAutomation")',
    '  |> range(start: -1d, stop: now())',
    '  |> filter(fn: (r) => r._measurement == "RoomTemperature")',
    '  |> filter(fn: (r) => r._field == "Temperature")',
    '  |> yield(name: "2")',
  ].join('\n');
  return msg;
```

## 5.2 Індивідуальне завдання

Розробити програму для хмарного середовища Node-RED в IBM Cloud та реалізувати зв’язок з граничним рівнем. Реалізувати обробку інформації в хмарі та збереження даних в хмарних сховищах даних, відповідно до розробленої архітекрути.

## 5.3 Контрольні питання

1. Що таке WebSocket? Які принципи роботи?
2. Яким чином виконується з'єднання WebSocket в Node-RED?
3. В якому форматі передаються дані при використанні WebSocket?
4. Які принципи роботи бази даних часових рядів?
5. Що таке API token?
6. В якому форматі відбувається комунікація з InfluxDB?
7. В якому вигляді відбувається запис даних в InfluxDB?
8. В якому вигляді відбувається зчитування даних з InfluxDB?

# Література


1. Krogh A. An Introduction to the Internet of Things. Bookboon. 2020. P. 172.
2. Vermesan O., Bacquet J. Internet of Things – The Call of the Edge Everything Intelligent Everywhere. River Publishers. 2020. P. 365.
3. Nitulescu I.-V., Korodi A. Supervisory Control and Data Acquisition Approach in Node-RED: Application and Discussions. IoT. 2020. № 1. P. 76–91.
4. The Edge Computing Advantage: An Industrial Internet Consortium White Paper. [Електронний ресурс] / 2019. P. 11. – Режим доступу: https://www.iiconsortium.org/pdf/IIC_Edge_Computing_Advantages_White_Paper_2019-10-24.pdf. – Дата доступу 20.10.2021.
5. Node.js. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://nodejs.org/en/. – Дата доступу 20.10.2021. – Назва з екрану.
6. Node-RED Documentation. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://nodered.org/docs/. – Дата доступу 20.10.2021. – Назва з екрану.
7. Modbus Specifications and Implementation Guides. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://modbus.org/specs.php. – Дата доступу 20.10.2021. – Назва з екрану.
8. IBM Cloud Docs. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://cloud.ibm.com/docs/. – Дата доступу 20.10.2021. – Назва з екрану.
9. InfluxDB Cloud. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://www.influxdata.com/products/influxdb-cloud/. – Дата доступу 20.10.21 – Назва з екрану.
10. Get started with Flux. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://docs.influxdata.com/flux/v0.x/get-started/. – Дата доступу 20.10.2021. – Назва з екрану.