МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

ВОЛОХ Владислав Вікторович

Програмний модуль розгортання сервера інтеграції даних в системі Інтернету речей / Software module for deploying the data integration server in the IoT system

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав: студентка групи КІ-41
ВОЛОХ Владислав Вікторович

Науковий керівник
к.т.н. Дубчак Л.О.

Тернопіль 2025

АНОТАЦІЯ

Волох В.В. Програмний модуль розгортання сервера інтеграції даних в системі Інтернету речей. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2025.

Робота написана обсягом 51 сторінку і містить 10 рисунків, 9 таблиць, 3 додатки та 34 джерела за переліком посилань. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою кваліфікаційної роботи розробка програмного модуля автоматизованого розгортання сервера для інтеграції та обробки даних у системах Інтернету речей. Проведено розробку архітектури програмного модуля автоматичного розгортання, що дало змогу забезпечити структуровану взаємодію між API, бізнес-логікою та базою даних, адаптовану до різних сценаріїв IoT-інтеграції. Проведено реалізацію алгоритмів автоматизації розгортання та налаштування серверів, що дало змогу створити модуль, здатний обчислювати параметри інфраструктури та генерувати TOSCA YAML-шаблони для розгортання систем ThingsBoard з урахуванням реальних умов.

Ключові слова: сервер інтеграції даних, розгортання, TOSCA, YAML, THINGSBOARD.

ANNOTATION

Volokh V.V. Software module for deploying the data integration server in the IoT system. – Manuscript.

Qualification work for the Bachelor degree in specialty 123 “Computer Engineering”, educational and professional program. Western Ukrainian National University, Ternopil, 2025.

The work is written in 51 pages and contains 10 figures, 9 tables, 3 appendices and 34 references. The amount of graphic material is 2 sheets of A3 format.

The purpose of the qualification work is to develop a software module for automated server deployment for data integration and processing in Internet of Things systems. The architecture of the software module for automatic deployment was developed, which made it possible to provide structured interaction between API, business logic and database, adapted to various IoT integration scenarios. The algorithms for automating server deployment and configuration were implemented, which made it possible to create a module capable of calculating infrastructure parameters and generating TOSCA YAML templates for deploying ThingsBoard systems taking into account real conditions.

Keywords: DATA INTEGRATION SERVER, DEPLOYMENT, TOSCA, YAML, THINGSBOARD.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IoT	–	Internet of Things
АІС	–	автоматизовані інформаційні системи
ГІС	–	геоінформаційна система
СІД	–	Система інтеграції даних
TOSCA	–	Topology and Orchestration Specification for Cloud Applications
ETL	–	Extract, Transform, Load

ВСТУП

В умовах стрімкого розвитку технологій Інтернету речей (Internet of Things, IoT) особливого значення набуває проблема ефективного управління потоками даних, що надходять від великої кількості сенсорів, актуаторів та інших пристроїв, розподілених у просторі. Основою для забезпечення цілісності, узгодженості та аналітичної цінності таких даних є сервери інтеграції, які виконують функції приймання, нормалізації, агрегації, зберігання та візуалізації інформації. Збільшення масштабів IoT-систем потребують нових підходів до автоматизації розгортання серверної інфраструктури.

Автоматизоване розгортання серверів інтеграції IoT-даних дозволяє значно знизити час налаштування, підвищити гнучкість адаптації до нових сценаріїв використання та забезпечити стабільність конфігурації в умовах змінного навантаження. Такий підхід особливо важливий для розподілених систем, які функціонують у гетерогенному середовищі edge-, fog- та хмарних обчислень. Проблематика розгортання таких серверів пов’язана з необхідністю точного обліку ресурсних потреб (обсяг телеметрії, пропускна здатність, CPU, RAM, сховище), адаптивного вибору технологічного стеку та коректної генерації конфігурацій відповідно до вимог цільового середовища.

Зважаючи на тенденцію до зростання кількості IoT-пристроїв, дедалі актуальнішим стає впровадження підходів, що базуються на моделюванні інфраструктури у вигляді декларативних шаблонів розгортання, таких як Topology and Orchestration Specification for Cloud Applications. Використання таких моделей дозволяє автоматизувати розгортання серверів інтеграції даних, зменшити кількість помилок при конфігуруванні, а також забезпечити відтворюваність архітектур рішень у різних середовищах.

Розробка програмного модуля, що забезпечує автоматизоване розгортання серверів інтеграції IoT-даних, є важливим кроком до підвищення ефективності проєктування, впровадження та масштабування систем Інтернету речей у промислових, міських, екологічних та інших критичних застосуваннях.

Об’єкт дослідження: сервери інтеграції даних в системах Інтернету речей. Предмет дослідження: алгоритми та програмні засоби для автоматизованого розгортання серверів інтеграції IoT-даних. Метою кваліфікаційної роботи розробка програмного модуля автоматизованого розгортання сервера для інтеграції та обробки даних у системах Інтернету речей. Для досягнення поставленої мети необхідно вирішити такі завдання:

1. проаналізувати методи інтеграції даних у сучасних IoT-системах.
2. вибрати технологій для розгортання та управління сервером інтеграції даних.
3. розробити програмну архітектуру модуля автоматичного розгортання.
4. реалізувати алгоритми автоматизації розгортання та налаштування сервера.
5. провести тестування роботи створеного програмного модуля.

Результати дослідження апробовано на II Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі» [1] (додаток В).

У першому розділі здійснено огляд сучасних IoT-систем з акцентом на структуру архітектури, де особливу увагу приділено ролі серверів інтеграції даних. Проведено аналіз основних методів збору, обробки, агрегації та нормалізації IoT-даних, розглянуто сучасні підходи до їх інтеграції в умовах гетерогенності пристроїв та протоколів зв’язку.

У другому розділі визначено вимоги до модуля, описано формат вхідних конфігураційних файлів, інтерфейси взаємодії з API платформи та принципи генерації шаблонів розгортання. Розроблено алгоритми вибору пристроїв, шлюзів, протоколів, а також обчислення ресурсів серверної інфраструктури.

У третьому розділі реалізовано запропонований модуль з використанням сучасного технологічного стеку. Побудовано об’єктну модель програмного забезпечення, де виокремлено рівні API, бізнес-логіки та збереження даних. Проведено функціональне тестування системи зокрема перевірку згенерованих конфігураційних файлів.

1. АНАЛІЗ ЗАСОБІВ ІНТЕГРАЦІЇ ДАНИХ В ІОТ-СИСТЕМАХ

1.1. Структура систем Інтернету речей

Система Інтернету речей (IoT) — це сукупність технологій, пристроїв та програмного забезпечення, що забезпечує автоматизовану взаємодію між фізичними об’єктами, які здатні генерувати, передавати, приймати та обробляти дані через мережеві протоколи без участі людини. Головна мета таких систем — інтеграція реального світу з цифровим середовищем задля підвищення ефективності, зручності та керованості процесів.

Розглянемо ключові компоненти IoT-системи.

1. Пристрої збору даних (сенсори та актуатори) фіксують фізичні параметри навколишнього середовища (температуру, вологість, рівень освітлення, тиск, рівень шуму тощо). Актуатори, у свою чергу, здатні змінювати стан об'єктів — наприклад, відкривати клапани, вмикати освітлення або регулювати HVAC-системи.

2. Пристрої передачі даних (мережеві шлюзи) забезпечують передачу даних від сенсорів до серверної інфраструктури через дротові або бездротові канали (Wi-Fi, LoRaWAN, NB-IoT, Zigbee, Ethernet тощо). Часто шлюзи також виконують попередню обробку чи фільтрацію даних.

3. Сервер обробки та зберігання даних (IoT-платформа) - центральна частина системи, де відбувається агрегація, зберігання, аналіз та маршрутизація даних. Платформи реалізують логіку пристроїв, налаштування тригерів, формування дашбордів та API-взаємодію.

4. Інтерфейси взаємодії з користувачем (UI/API) призначені для перегляду телеметрії, керування пристроями, візуалізації аналітики та конфігурації системи. Вони реалізуються у вигляді веб- або мобільних застосунків, REST API чи інтеграції з хмарними сервісами.

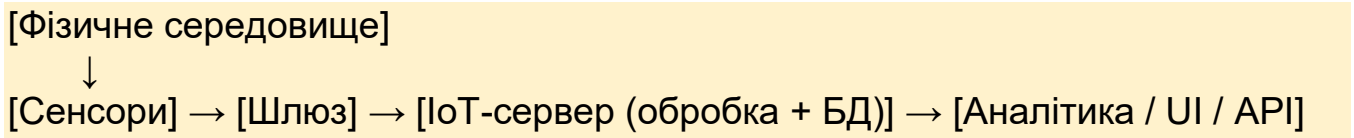


Рисунок 1.1 - Структура типової IoT-системи

У системі «розумного дому» температурні сенсори, вологоміри та сенсори руху постійно передають дані на локальний або хмарний сервер через шлюз (наприклад, ESP32 або Raspberry Pi). Ці дані аналізуються, і відповідно до заданої логіки, автоматично вмикається кондиціонер, освітлення або надсилається сповіщення власнику через мобільний додаток.

У випадку «розумного міста» IoT-система охоплює сотні сенсорів (наприклад, датчики якості повітря, рівня шуму, заповненості сміттєвих контейнерів), розміщених у різних частинах міста. Дані передаються на централізовану платформу, де агрегуються та аналізуються. За результатами аналізу муніципальні служби можуть автоматично змінювати режими вуличного освітлення, маршрути сміттєзбірних машин або формувати публічні карти забруднення.

Архітектура систем Інтернету речей (IoT) є багаторівневою, ієрархічною та поділяється на функціональні шари, кожен з яких виконує специфічні завдання у процесі збору, передавання, обробки та використання даних. Ця архітектура побудована за принципами розподіленої обчислювальної моделі з фокусом на масштабованість, мобільність, асинхронність та обмежену ресурсність окремих компонентів.

Головна особливість архітектури IoT — це тісна інтеграція фізичних об’єктів із цифровою інфраструктурою. На відміну від класичних IT-систем, IoT оперує великим числом пристроїв з обмеженими обчислювальними ресурсами, які взаємодіють у реальному часі, використовують нестійкі бездротові мережі та функціонують у гетерогенному середовищі.

Опишемо детально основні рівні (шари) IoT-системи.

Перший рівень фізичних пристроїв (датчики та актуатори) -це найнижчий рівень архітектури, який забезпечує взаємодію з фізичним середовищем. Особливістю рівня є обмежені ресурси, енергозалежність, нестабільне підключення. Компоненти:

– датчики здійснюють вимірювання фізичних величин (температури, вологості, тиску, рівня шуму, CO_2 тощо);

– актуатори це пристрої, які впливають на середовище (відкривають клапани, перемикають реле, запускають двигуни);

– мікроконтролери (ESP32, STM32) часто поєднані з сенсорами, забезпечують первинну обробку даних.

Другий рівень передавання даних (мережеві шлюзи та протоколи) забезпечує збір даних з пристроїв та їхнє передавання до серверної частини IoT-системи. Сюди входять:

– шлюзи (IoT Gateway) це пристрої або програми, які збирають дані з датчиків і пересилають їх через TCP/IP, LoRaWAN, NB-IoT, Wi-Fi або Ethernet.

– протоколи передачі MQTT, CoAP, HTTP, AMQP, які забезпечують легку та енергоефективну передачу телеметрії.

Шлюзи можуть виконувати фільтрацію, агрегацію або локальну обробку (edge computing).

Третій Рівень обробки та зберігання (сервери, хмарні платформи [2]) відповідає за агрегацію, зберігання, обробку даних та керування пристроями. IoT-платформи, наприклад, ThingsBoard, AWS IoT, FIWARE, Azure IoT Hub реалізують:

– авторизацію пристроїв;

– отримання телеметрії;

– запуск правил обробки;

– зберігання історичних даних;

– інтеграцію з базами даних, хмарними API, ML-модулями.

Четвертий рівень застосунків реалізує аналітику, візуалізацію, керування. Рівень реалізує взаємодію з користувачем або зовнішніми системами. Використовуються веб-інтерфейси, мобільні додатки, API для візуалізації даних, керування пристроями, створення звітів та прийняття рішень. Використовуються Аналітичні сервіси для виявлення відхилень, прогнозування поведінки, прийняття рішень (AI/ML). Особливості: адаптивний інтерфейс, рольовий доступ, інтеграція з зовнішніми платформами (SCADA, ERP, GIS тощо).

Дані, зібрані сенсорами, передаються через шлюзи на сервер обробки, де вони зберігаються, аналізуються і передаються далі до застосунків. Застосунки, у свою чергу, можуть не лише відображати ці дані, але й ініціювати зворотний сигнал (наприклад, включення пристрою), який прямує у зворотному напрямку — від хмари до актуатора.

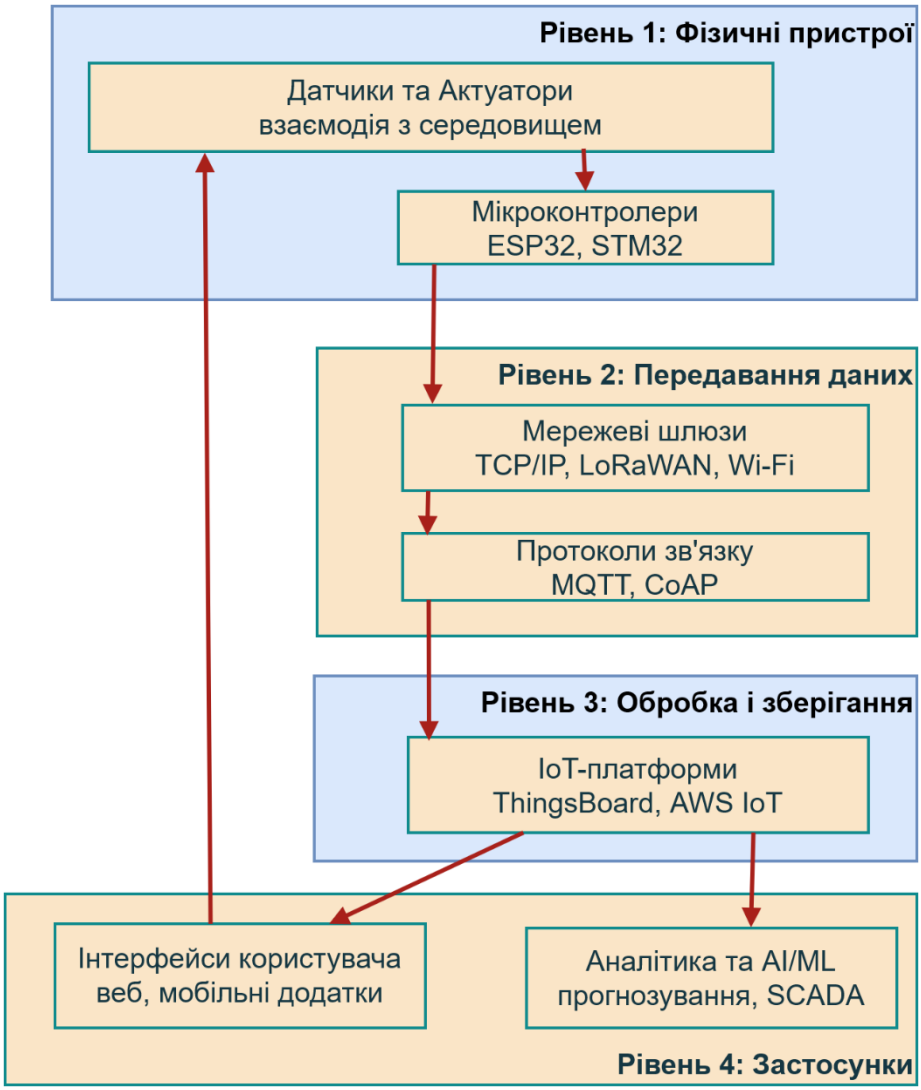


Рисунок 1.1 - Основні рівні (шари) IoT-системи

1.2 Роль серверів інтеграції даних у загальній архітектурі IoT

Сервери інтеграції даних у системах Інтернету речей (IoT) відіграють центральну роль у забезпеченні узгодженої взаємодії між гетерогенними пристроями, обробці та маршрутизації даних, а також у реалізації логіки реагування на події. Їхня наявність є критично важливою для перетворення сирих даних, отриманих від сенсорів, у структуровану, інтерпретовану та доступну інформацію, придатну до подальшого аналізу, керування або візуалізації.

Сервер інтеграції даних виступає проміжною ланкою між польовим рівнем (датчики, шлюзи) та прикладними рівнями (аналітика, інтерфейси користувача,

зовнішні системи). Він уніфікує вхідні потоки даних із різних джерел, виконує їх попередню обробку (нормалізацію, агрегацію, фільтрацію), ініціює події або реакції згідно із заданими правилами, а також забезпечує доступ до цих даних через API.

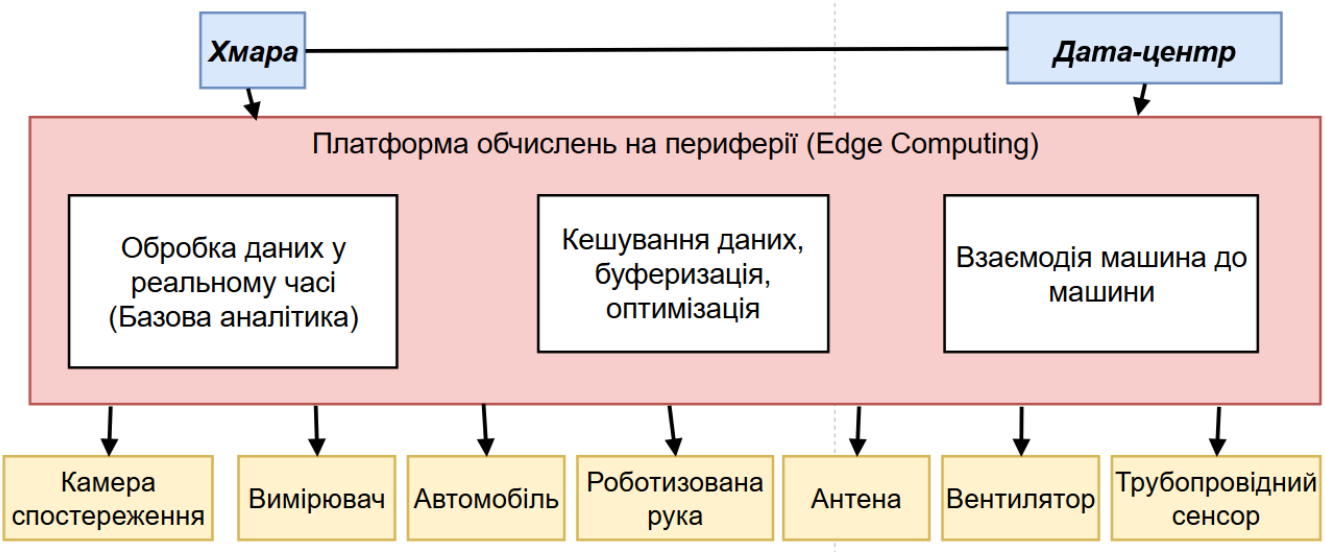


Рисунок 1.2 - Платформа обчислень на периферії (Edge Computing)

Типові завдання серверів інтеграції в IoT:

1. Сервер виступає як брокер або endpoint, що приймає телеметрію з пристроїв через протоколи MQTT, HTTP, CoAP.

2. Реєстрація та автентифікація пристроїв. Кожен пристрій повинен бути зареєстрованим, автентифікованим і пов'язаний із відповідним користувачем, групою або сервісом.

3. Нормалізація та структуризація даних. Сирі потоки телеметрії перетворюються у стандартизовану форму: форматування JSON, перетворення одиниць, типізація.

4. Обробка подій (rule engine). Сервер може запускати сценарії у відповідь на умови: перевищення порогів, втрата зв'язку, поява нових пристроїв.

5. Маршрутизація даних до баз даних, аналітичних модулів, зовнішніх API. Наприклад, передача даних до TimescaleDB, Elasticsearch, Grafana, або сторонніх REST API для зберігання чи аналізу.

6. Публікація даних для споживачів. Користувацькі дашборди, API-запити, сповіщення користувачам, генерація звітів.

Як сервер інтеграції може виступати платформа ThingsBoard:

– приймає MQTT/HTTP-повідомлення від пристроїв;

– використовує Rule Engine для генерації подій;

– зберігає дані в TimescaleDB або PostgreSQL;

– публікує інформацію через REST API або WebSocket;

– відображає динамічні панелі для користувачів.

Процес збирання даних розпочинається з того, що сенсори, встановлені у фізичному середовищі, здійснюють перетворення аналогових параметрів (наприклад, температури, вологості, рівня освітлення) у цифрові сигнали. Ці сигнали або безпосередньо, або через проміжні шлюзи надсилаються до сервера за допомогою телекомунікаційних протоколів, таких як MQTT, CoAP або HTTP. У багатьох випадках шлюзи не лише передають дані, але й виконують базову фільтрацію або агрегацію, що зменшує обсяг переданої інформації та знижує навантаження на сервер.

Після надходження дані проходять через стадію нормалізації. Це означає, що сервер виконує уніфікацію формату вхідної інформації, наприклад, перетворення одиниць виміру, переформатування структури JSON або переведення показників у стандартизовані типи. Такий підхід дозволяє системі оперувати уніфікованим уявленням про дані, незалежно від виробника чи специфікації пристрою.

Наступним етапом є обробка даних, що реалізується через внутрішню логіку сервера. Вона може включати аналіз умов, виявлення відхилень, запуск тригерів і сценаріїв реагування. Наприклад, якщо температура перевищує заданий поріг, система автоматично формує сповіщення або активує відповідну реакцію, таку як увімкнення системи охолодження. Також часто використовується історичне порівняння значень, що потребує доступу до бази даних, яка зберігає телеметричну інформацію за попередні періоди.

Після обробки результати передаються на прикладні сервіси. Це можуть бути веб-інтерфейси візуалізації, мобільні застосунки, аналітичні панелі або зовнішні інформаційні системи. Передача може здійснюватися як у форматі потокових даних, так і у вигляді запитів через REST API. У деяких випадках сервер також інтегрується

з хмарними платформами або великими дата-центрами для подальшої обробки, машинного навчання чи довгострокового зберігання.

1.3. Методи інтеграції та обробки IoT-даних

У системах IoT існує кілька принципово різних методів інтеграції та обробки даних. Найбільш поширеними серед них є використання технологій ETL, MQTT-брокерів та REST-шлюзів. Кожен із цих підходів має свої переваги та обмеження у контексті швидкості обміну, масштабованості та стійкості до відмов, що визначає доцільність їхнього застосування в конкретних архітектурних рішеннях.

Extract, Transform, Load (ETL) [3] — класична технологія обробки даних, яка широко використовується в дата-орієнтованих системах. У області IoT вона застосовується для періодичного отримання, трансформації та завантаження великих обсягів телеметрії у сховища даних. Цей підхід орієнтований на пакетний режим роботи й не придатний для сценаріїв з низькою затримкою або високою частотою оновлення. До недоліків також належить затримка оновлення, потреба у зовнішніх джерелах для ініціації завантаження, а також складність масштабування при роботі з нестабільними або гетерогенними джерелами IoT-даних.

MQTT-брокери (наприклад, Mosquitto, EMQX, HiveMQ) є стандартним засобом обміну даними в IoT середовищах. Протокол MQTT забезпечує публікацію та підписку на теми з надзвичайно низькою затримкою, мінімальним накладним трафіком і підтримкою механізмів повторної доставки. Це робить його оптимальним для реального часу, обмежених каналів зв’язку та великої кількості розподілених сенсорів. Основною перевагою MQTT є висока масштабованість, ефективність при малій пропускній здатності та стабільність при нестабільному зв’язку. Водночас, MQTT передбачає **стейтлесну** модель зберігання, і сам брокер не виконує складної трансформації даних — це вимагає окремого обробника (наприклад, rule engine на ThingsBoard). Крім того, MQTT не є оптимальним для масового запиту архівних або агрегованих даних, оскільки не забезпечує інтерактивний інтерфейс доступу до них.

REST-шлюзи використовуються для інтеграції пристроїв або сторонніх систем за допомогою HTTP-запитів, які включають структури типу POST або GET із передачею JSON або XML-даних. Цей підхід простий у реалізації, добре підтримується більшістю сучасних технологій і зручний для інтеграції з веб-сервісами або мобільними додатками. Основною перевагою REST-шлюзів є сумісність з існуючими корпоративними API, а також наявність багатьох інструментів для тестування, валідації та логування трафіку. Проте, REST має вищу затримку, залежить від відкриття та завершення з'єднання для кожного запиту, що робить його менш ефективним для потокових або подієвих сценаріїв. До того ж, REST не забезпечує автоматичної повторної доставки при збої з'єднання, що обмежує його надійність у мобільних або енергозалежних IoT-сценаріях.

1.4. Постановка задач роботи

Платформи для обробки потокових даних забезпечують маршрутизацію, перетворення, фільтрацію та аналіз телеметричних даних у реальному часі. Найбільш популярними та функціональними серед них є Node-RED, ThingsBoard, FIWARE та Apache NiFi. Кожна з цих платформ має власну архітектурну модель, специфічні інструменти інтеграції та рівень зручності для розробників.

Node-RED [4] — це потужна візуальна платформа на базі Node.js, орієнтована на створення потокових сценаріїв за допомогою блок-схем. Її ключова перевага — інтуїтивно зрозумілий графічний інтерфейс, де обробка даних реалізується шляхом з'єднання модулів ("нодів") у логічні ланцюги. Node-RED підтримує величезну кількість готових інтеграцій із пристроями, API, протоколами (MQTT, HTTP, WebSocket, Modbus).

Платформа менш придатна для великих розподілених систем, де потрібна централізована автентифікація, багатокористувацька підтримка або масштабування.

ThingsBoard [5] — це повнофункціональна IoT-платформа з відкритим вихідним кодом, яка об’єднує телеметричну обробку, керування пристроями,

візуалізацію та rule engine. Вона побудована за мікросервісною архітектурою та підтримує протоколи MQTT, HTTP, CoAP, SNMP, а також зовнішні бази даних (PostgreSQL, Cassandra). Система містить вбудований Rule Engine, який дозволяє створювати складні сценарії обробки подій на рівні платформи без необхідності писати код. Інтерфейс адміністратора та дашборди користувачів налаштовуються через веб-інтерфейс, що значно спрощує конфігурацію. ThingsBoard чудово підходить як для пілотних проєктів, так і для промислових впроваджень завдяки масштабованій архітектурі, ролям доступу, інтеграції з системами безпеки й підтримці мультиорендарності (multi-tenant). Її недоліком є складність інсталяції в порівнянні з Node-RED, а також потреба у знаннях DevOps для адміністрування.

FIWARE [6] — це IoT-платформа, орієнтована на смарт-міста, промислові та громадські сервіси. Вона побудована як набір мікросервісів, відомих як "Generic Enablers", що взаємодіють через стандартні NGSI-протоколи. Ключовим компонентом є Orion Context Broker [7], який відповідає за збереження, оновлення та підписку на зміни контексту в режимі реального часу. FIWARE активно підтримує інтеграцію з різними сенсорними платформами, геопросторовими сервісами,. Серед її переваг — модульність, відкритість до стандартів та потужна підтримка від спільноти Smart Cities. З точки зору функціональності, FIWARE є найкращим вибором для масштабних урядових або інфраструктурних проектів, проте вона має високий поріг входу, складну документацію й потребує налаштування багатьох компонентів навіть для базової роботи.

Apache NiFi [8] — це високопродуктивна система маршрутизації та обробки потокових даних, спроєктована для обробки великих обсягів інформації з надійною гарантією доставки. Вона дозволяє візуально створювати потокові схеми (flow-based programming) з контролем за темпом обробки, чергами, обмеженням трафіку, відмовостійкістю та аудиторськими журналами.

NiFi має вбудовану підтримку для MQTT, HTTP, TCP/UDP, JDBC, SFTP та інших протоколів. Окремою перевагою є підтримка розподіленої обробки, можливість інтеграції з Apache Kafka та потужні інструменти з моніторингу продуктивності. Ця платформа ідеально підходить для критичних систем з великою кількістю джерел, потребою в гарантованій доставці й ретельному контролі потоку

даних. Її недоліком є складність початкового налаштування, відсутність нативної підтримки IoT-інтерфейсів на прикладному рівні та велика потреба в обчислювальних ресурсах.

Таблиця 1.1 – Порівняння платформ

Платформа	Масштабо-ваність	Підтримка протоколів	Призначення
Node-RED	Середня	MQTT, HTTP, WebSocket	Просте з’єднання та обробка даних
ThingsBoard	Висока	MQTT, CoAP, HTTP	Повноцінна IoT-платформа з rule engine
FIWARE	Висока	NGSI, HTTP, MQTT	Контекстно-орієнтовані IoT-сервіси
Apache NiFi	Висока	MQTT, TCP, REST	Надійна обробка великих потоків даних

У сучасних IoT-системах використовується багато протоколів і форматів даних — залежно від рівня (фізичний, мережевий, транспортний, прикладний), енергоспоживання, складності пристроїв та архітектури (edge/fog/cloud). Нижче систематизовано найпоширеніші протоколи та формати.

Таблиця 1.2 – Порівняння Прикладних протоколів

Протокол	Призначення	Особливості
MQTT	Легкий publish/subscribe	Надзвичайно легкий, ідеальний для sensor-to-cloud
CoAP	REST-подібний для constrained devices	Працює поверх UDP, підтримує multicast
HTTP/ HTTPS	Стандарт для веб-сервісів	Простий, але ресурсоємний для деяких пристроїв
AMQP	Обмін повідомленнями між підприємствами	Надійний, складніший за MQTT
WebSockets	Двостороння комунікація	Відкрита сесія, зручно для реального часу
LwM2M	Менеджмент IoT-пристроїв	використовує CoAP

В таблиці 1.3 підсумовано потрібні характеристики протоколів і форматів даних надісланих сенсорами .

Таблиця 1.3 - Характеристики протоколів і формати даних

Мережеві протоколи	
IPv6 / 6LoWPAN	Оптимізоване IPv6 для low-power мереж
RPL	Протокол маршрутизації для IoT (з IPv6)
Thread	Мережа на базі IPv6 для домашньої автоматизації
Фізичні та канальні протоколи	
Протокол	Призначення
Bluetooth LE	Низьке енергоспоживання, персональні пристрої
Zigbee	Сітка для домашніх і промислових мереж
Z-Wave	Домашня автоматизація, низька енергія
LoRa / LoRaWAN	Дальня передача з низьким енергоспоживанням
NB-IoT	Мобільний інтернет речей, на базі LTE
Wi-Fi	Високошвидкісний зв’язок, але енергозатратний
Формати даних	
Формат	Опис
JSON	Найпоширеніший для REST, MQTT, HTTP
JSON-LD	Семантичні дані (для NGSI-LD)
MessagePack	Бінарний, ефективніший за JSON
XML	Застарілий, іноді використовується в SOAP
Protobuf	Компактний формат від Google, використовується в gRPC

Типові приклади використання:

- MQTT + JSON для сенсорів, що надсилають дані в хмару;
- CoAP + CBOR в обмежених пристроях з мінімальним обсягом даних;
- LoRa + MQTT у довготривалих сенсорних мережах на відкритому повітрі.

Метою кваліфікаційної роботи розробка програмного модуля автоматизованого розгортання сервера для інтеграції та обробки даних у системах Інтернету речей. Для досягнення поставленої мети необхідно вирішити такі завдання:

1. проаналізувати методи інтеграції даних у сучасних IoT-системах.
2. вибрати технологій для розгортання та управління сервером інтеграції даних.
3. розробити програмну архітектуру модуля автоматичного розгортання.
4. реалізувати алгоритми автоматизації розгортання та налаштування сервера.
5. провести тестування роботи створеного програмного модуля.

2 АРХІТЕКТУРА ПРОГРАМНОГО МОДУЛЯ РОЗГОРТАННЯ

2.1 Розгортання IoT-додатків

Розгортання IoT-додатків (англ. IoT Application Deployment) — це процес розгортання прикладного програмного забезпечення, яке керує, обробляє або візуалізує дані від пристроїв Інтернету речей, на обрану інфраструктуру (локальний сервер, edge-пристрій, хмара). Це впровадження IoT-додатку в експлуатацію з повним підключенням до датчиків, брокерів повідомлень, баз даних, аналітики й інтерфейсів. В таблиці 2.1 систематизовано основні складові додатків.

Таблиця 2.1 - Основні складові IoT-додатку

Компонент	Опис
IoT-пристрої (сенсори, актуатори)	Фізичні пристрої, які генерують або споживають дані
Протоколи зв’язку	MQTT, CoAP, HTTP, LwM2M — забезпечують передачу даних
Edge/Fog шлюзи	Попередня обробка даних перед відправкою в хмару
IoT-платформа	Наприклад, ThingsBoard, FIWARE, AWS IoT — керує пристроями, даними
Аналітика/обробка	Алгоритми, ML-моделі або правила, що аналізують потоки
Дашборди/інтерфейси	Веб або мобільні інтерфейси для взаємодії з даними

Процес інтеграції даних починається зі збирання інформації з сенсорів та пристроїв. Пристрої генерують потік даних, що часто надходить у сирому вигляді, з різною частотою та точністю. Зі збільшенням кількості пристроїв, зростає і обсяг вхідних даних, що створює навантаження на канали передачі та сервери первинного прийому.

Далі ці дані підлягають нормалізації, що включає приведення їх до єдиного формату, одиниць вимірювання та часових міток. У великих масштабах систем нормалізація ускладнюється через необхідність обробки повідомлень від різнорідних джерел у реальному часі. Зростання кількості повідомлень та показників за секунду вимагає високої пропускної здатності та паралельної обробки.

Після нормалізації дані проходять через етап фільтрації та агрегації. Тут відсікаються надлишкові або помилкові записи, а також обчислюються середні значення, медіани чи інші статистичні показники. За великої кількості точок даних в секунду необхідно застосовувати розподілені обчислення або обробку на рівні edge-пристроїв для уникнення затримок.

Наступним кроком є збагачення даних додатковою інформацією, наприклад, геолокацією, контекстом подій або історичними значеннями. При масштабуванні системи зростає обсяг метаданих, що додається, а також потреба у швидкому доступі до зовнішніх баз знань чи сервісів аналітики.

Після збагачення дані передаються до центрального сховища або аналітичної платформи. За умов високої інтенсивності потоку даних важливим є забезпечення масштабованої інфраструктури з балансуванням навантаження і використанням стрімінгових платформ, таких як Apache Kafka або MQTT-брокерів.

Розглянемо моделі та алгоритми розгортання IoT-серверів.

Фреймворк LEONORE [9] забезпечує еластичне розгортання компонентів додатків на ресурсно-обмежених та гетерогенних edge-пристроях у великомасштабних IoT-системах. Він підтримує як push-, так і pull-методи розгортання, що дозволяє ефективно масштабувати систему та зменшувати мережевий трафік між хмарою та edge-інфраструктурою.

Підхід DIANE [10] дозволяє динамічно генерувати оптимізовані топології розгортання для IoT-додатків у хмарі, враховуючи наявну фізичну інфраструктуру. Він базується на декларативній моделі з обмеженнями, що дозволяє гнучко розміщувати компоненти додатків на edge-пристроях. Метод для зменшення комунікаційних витрат і генерує оптимізовані топології розгортання IoT-додатків у хмарному середовищі, адаптовані до наявної фізичної інфраструктури.

Інтегрована архітектура IFCIoT (fog-cloud IoT) [11] об'єднує обчислювальні ресурси на edge-пристроях та в хмарі для забезпечення високої продуктивності, енергоефективності та низької затримки. Цей підхід особливо корисний для додатків, що вимагають обробки даних у реальному часі. Для оцінки результатів використовуються такі показники: продуктивність, енергоефективність, зменшення

затримки, швидший час відгуку, масштабованість. Також в статті обговорюється різниця між CLOUD, FOG, та EDGE обчисленнями.

Архітектура Chord-based Fog Architecture [12] використовує горизонтальну структуру Fog-обчислень для моделювання довіри в IoT-середовищах. Вона забезпечує масштабованість, швидкий доступ до даних та ефективну внутрішню комунікацію між вузлами.

В Україні проведено дослідження [13] де аналізується ефективність існуючих рішень промислового Інтернету речей [14] (Industrial Internet of Things, IIoT). Також розглядається рівень інтеграції технологій на підприємствах залежно від їх розміру та цілей впровадження.

Дослідження [15] пропонує процес оцінки IoT-платформ з урахуванням їх часової поведінки, що є критично важливим для систем, чутливих до часу, таких як медичні системи моніторингу пацієнтів.

Згідно з дослідженням, представленим у статті [16] оптимізовані топології розгортання для IoT-додатків у хмарному середовищі представляються у вигляді декларативних моделей, заснованих на обмеженнях (constraint-based declarative models). Ці моделі описують бажане розгортання додатків, дозволяючи гнучке розміщення компонентів як у хмарній інфраструктурі, так і на edge-пристроях. У цьому контексті використовується підхід, де кожен компонент додатку розглядається як окрема одиниця, що може бути незалежно розгорнута. Це дозволяє системі адаптувати топологію розгортання в реальному часі у відповідь на зміни в навколишньому середовищі, такі як зміни навантаження або доступності ресурсів.

Для опису таких топологій [17] часто застосовуються стандартизовані мови моделювання, наприклад, Topology and Orchestration Specification for Cloud Applications (TOSCA). TOSCA дозволяє описувати структуру додатку, його компоненти, взаємозв'язки між ними та політики управління, що спрощує автоматизацію розгортання та управління життєвим циклом додатків у хмарних та edge-середовищах. Рисунок демонструє відповідність між концептами шаблонів сервісів TOSCA та відповідними візуальними нотаціями UML або мережевими діаграмами.

Таблиця 2.2 - Візуалізація шаблонів сервісів TOSCA

Концепт TOSCA	Візуальна нотація
Types (Типи) абстрактні типи вузлів, з'єднань, властивостей тощо,	Діаграми класів (class diagrams)
Node/relationship templates конкретні екземпляри типів і їх взаємозв'язки	- Компонентні діаграми (component diagrams) - Діаграми розгортання (deployment diagrams) - Мережеві діаграми (Network diagrams)
Policies (Політики) політики керування сервісом	Діаграми послідовності (sequence diagrams)
Workflows (Робочі процеси) відображають логіку виконання процесів,	Діаграми активності (activity diagrams)

Фізична топологія [14] поверх якої розгортаються IoT додатки зображена на рисунку 2.1.

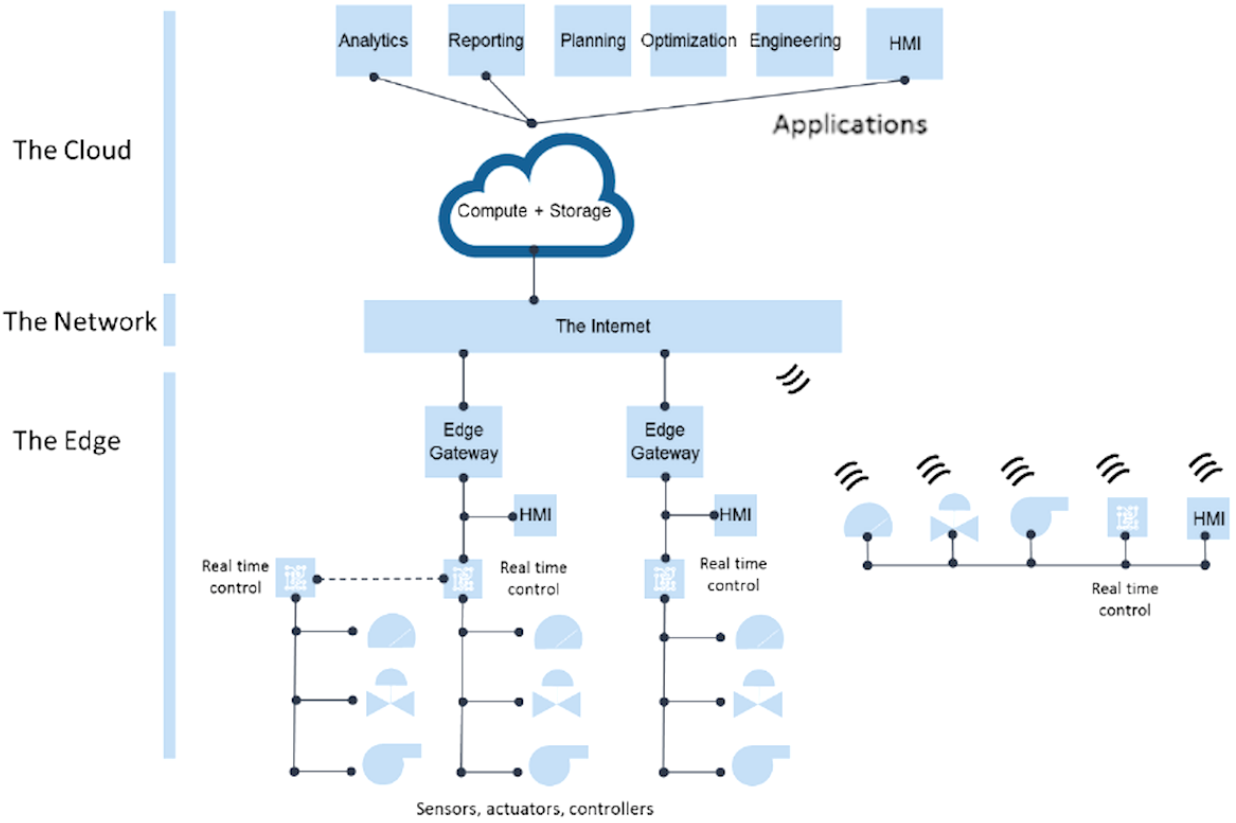


Рисунок 2.1 -Фізична топологія IoT мережі

2.2 Вимоги до модуля автоматизації

Потрібно створити модуль, який дозволяє автоматизовано реєструвати пристрої, підключати брокери даних, налаштовувати правила обробки. Особливості:

- конфігураційний JSON/YAML-файл;
- REST-клієнт для API ThingsBoard;
- генерація шаблонів пристроїв, профілів, правила (Rule Chains);
- вбудована підтримка MQTT/HTTP брокерів або зовнішній конфіг.

Результат роботи: архітектура та прототип модуля.

Для прототипування інтерфейсу можна використати опис модульної архітектури із [14].

Таблиця 2.3 - Рівнева модульні архітектура IIoT

Шар	Опис
Шар контенту	Пристрої взаємодії з користувачем (екрани комп'ютерів, POS-станції, планшети, смарт-окуляри, сенсорні панелі)
Сервісний шар	Програми й ПЗ для аналізу даних та перетворення їх у корисну інформацію
Мережевий шар	Протоколи комунікації: Wi-Fi, Bluetooth, LoRa, стільниковий зв'язок
Шар пристроїв	Апаратні засоби: кіберфізичні системи (CPS), машини, сенсори

Вимоги до інтерфейсу програми сервера інтеграції даних IIoT

Таблиця 2.4 - Вимоги до інтерфейсу програми сервера інтеграції даних IIoT

Функції	Деталізація
Узагальнений REST API	POST /data/ingest — прийом телеметрії з пристроїв (температура, тиск, вібрації тощо) GET /devices — список підключених пристроїв GET /data/query — запит аналітичних даних (фільтр за датами, типами) POST /control/command — керування актуаторами (наприклад, зупинити двигун)
Інтерфейс користувача	Панель візуалізації поточних значень параметрів Графіки за періодами (історія вимірів) Конфігуратор правил та подій (тригери, сповіщення)
Інтеграція з платформами	Підтримка MQTT, LoRa, HTTP-з'єднань API-ключі для SCADA, ERP, ML-платформ

Модуль повинен генерувати TOSCA YAML-файл, який міститиме:

– tosca_definitions_version: версія специфікації TOSCA.

– description: опис розгортання СІД.

– node_templates: опис віртуальної машини/контейнера, на якій розгортається СІД, а також самого СІД як компонента. Це включає:

а) Тип вузла (наприклад, tosca.nodes.Compute для ВМ, tosca.nodes.Container.Application.Docker для Docker-контейнера).

б) Властивості (CPU, RAM, диск, ОС, мережеві інтерфейси).

в) Вимоги (наприклад, до програмного забезпечення).

г) Операції життєвого циклу (інсталяція, конфігурація СІД).

– relationship_templates (якщо є залежності) опис зв'язків між СІД та іншими компонентами (наприклад, базами даних).

– policy_templates (якщо застосовуються політики масштабування/автоматичного відновлення): Опис політик.

– artifacts: Посилання на скрипти встановлення, конфігураційні файли, образи Docker (якщо застосовується).

Спрощена об'єктна модель.

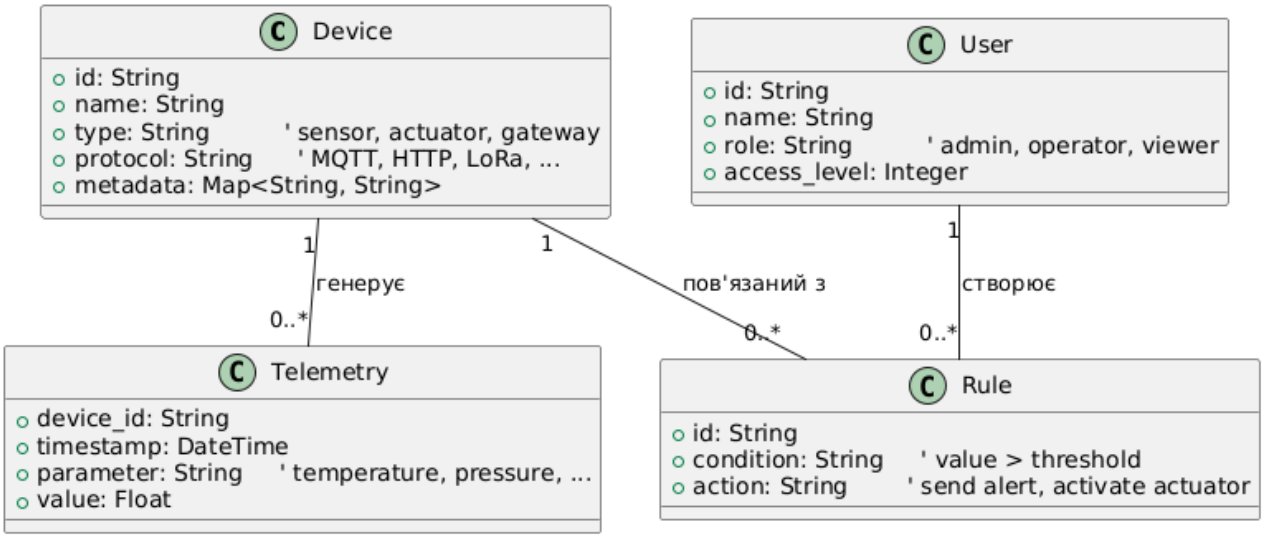
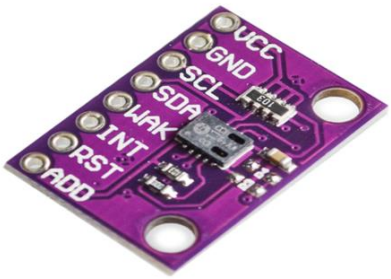


Рисунок 2.2 – Прототип об'єктної моделі сервера інтеграції даних IoT

Систематизуємо параметри IoT-пристроїв для розумного міста для створення довідника. Екологічні сенсори: якості повітря (CO_2, PM1, PM2.5, PM10), вологості

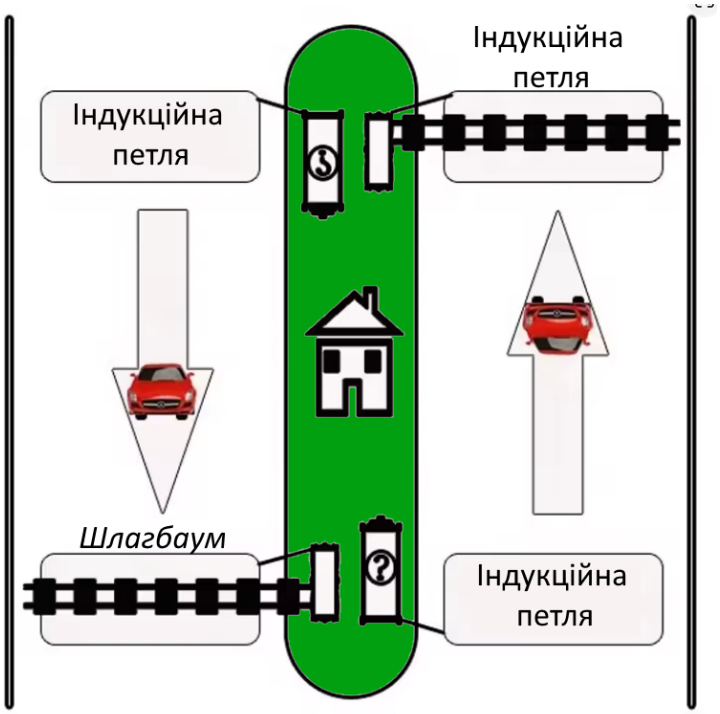
та температури повітря, датчики шуму. Пристрої керування освітленням: Розумні світлодіодні світильники, датчики освітленості, сенсори руху. Сенсори для керування дорожнім рухом включають: індуктивні петлі (детектори транспорту), сенсори заповненості парковок. Пристрої контролю комунальної інфраструктури включають: датчики рівня наповненості сміттєвих контейнерів, лічильники води, лічильники газу, лічильники електрики, сенсори виявлення витоків води. Сенсори безпеки включають: датчики пожежної сигналізації, камери відеоспостереження, системи контролю доступу



CCS811 - Датчик якості повітря, co2, вуглекислого газу для Arduino, 500 грн

Контролер індукційної петлі PD-132 (магнітної) 2000 грн

Детектор виявлення витоків води ER-ZSG-06, 500 грн



Контроль автомобілів на світлофорах, шлагбаумах

Рисунок 2.3 - IoT-пристрої для «розумного» міста

Таблиця 2.5 – Систематизація параметрів пристроїв

Тип пристрою	Приклад пристрою	Телеметричні дані	Атрибути	Приклади тривог (Alarms)
Екологічні сенсори	Датчик якості повітря MQ-135	CO_2, PM2.5, PM10, температура, вологість	Порогові значення концентрації, модель, місце розташування	Перевищення порогів CO_2 чи PM2.5
Керування освітленням	Smart LED лампа Philips	Статус (увімкнено/вимкнено), освітленість, споживання електрики	Поріг освітленості, режим роботи, модель пристрою	Несправність світильника, високе енергоспоживання
Контроль дорожнього руху	Паркувальний сенсор Bosch	Статус паркомісця (зайняте/вільне), час зайнятості	ID місця, GPS-координати, виробник	Тривале зайняття місця
Комунальна інфраструктура	Сенсор рівня сміття Sensoneo	Рівень наповненості контейнера (%)	Тип контейнера, місце розташування, графік обслуговування	Наповненість понад 90%
Безпека	Камера відеоспостереження Hikvision	Відеопотік, статус пристрою (онлайн/офлайн)	IP-адреса, локація, версія ПЗ	Втрата зв’язку, виявлення руху

Висновки щодо порівняння:

– екологічні сенсори генерують дані з низькою частотою, але критичні для моніторингу якості життя мешканців;

– пристрої керування освітленням мають високу інтеграцію з актуаторами та важливі для енергоефективності;

– пристрої для дорожнього руху генерують великий потік даних, які потребують миттєвої обробки;

– пристрої контролю комунальної інфраструктури характеризуються чіткими критичними значеннями для планування технічного обслуговування;

– сенсори безпеки мають найвищу важливість щодо безпеки та швидкого реагування.

На сучасних виробничих підприємствах впровадження технологій Інтернету

речей забезпечує суттєве покращення ефективності управління та контролю за виробничими процесами. Серед найпоширеніших пристроїв, що використовуються в умовах виробництва, є різноманітні сенсори, що дозволяють безперервно контролювати технологічні параметри, такі як температура обладнання, тиск у системах, вологість приміщень, рівень шуму та вібрацій. Також активно застосовуються сенсори контролю якості повітря, детектори диму та витоків газу, що дозволяє своєчасно виявляти й попереджати аварійні ситуації.

Крім сенсорів, на виробництві широко використовуються актуатори—пристрої, що забезпечують фізичну дію на обладнання. До них належать електромеханічні реле, розумні клапани регулювання витрати газів чи рідин, автоматизовані перемикачі й мотори, що здійснюють контроль швидкості та руху виробничих конвеєрів або інших механізмів. Використання актуаторів у поєднанні із сенсорами дозволяє будувати автоматизовані системи керування технологічними процесами, які здатні оперативно реагувати на зміни умов виробництва та забезпечувати оптимальну продуктивність, безпеку й економію ресурсів.

Актуатори передають інформацію щодо стану (увімкнено/вимкнено), а також інформацію щодо режиму роботи або виникнення помилок. В системах ThingsBoard кожен пристрій має свої атрибути—загальні, клієнтські та серверні, що визначають параметри експлуатації та налаштування. Важливою складовою є налаштування сигналів тривог, що активуються у випадку виходу контрольованих параметрів за межі нормативних значень або несправності обладнання.

2.3 Алгоритм генерації топології розгортання

Сформулюємо модель, що передбачає вибір пристроїв із довідників, визначення параметрів на рівні протоколів, та розрахунок параметрів для розгортання серверів ThingsBoard та інших необхідних серверів.

Таблиця 2.6 - Порівняння IoT-пристроїв виробничого підприємства

Категорія пристроїв	Приклад пристрою	Телеметричні дані	Атрибути	Приклади тривог (Alarms)
Сенсори температури обладнання	DS18B20 (датчик температури)	Температура (°C) обладнання	Граничні значення, модель, розташування	Перегрів обладнання
Сенсори вібрації	Вібраційний сенсор SKF CMSS	Рівень вібрації (Hz), амплітуда вібрації (g)	Порогові значення вібрації, місце встановлення, тип обладнання	Підвищений рівень вібрації
Сенсори вологості повітря	DHT22 (датчик вологості і температури)	Вологість (%), температура (°C) приміщення	Модель сенсора, місце встановлення, нормативна вологість	Підвищена вологість
Сенсори газу та диму	MQ-2 (датчик диму і газу)	Рівень концентрації газів (ppm), дим (наявність/відсутність)	Тип газу, допустимі концентрації, місце встановлення	Виявлення диму або витоку газу
Актуатори (клапани, реле)	Електричний клапан Belimo	Статус клапана (відкрито/закрито), режим роботи	Робочий тиск, модель, тип контрольованого середовища	Відмова клапана, несанкціоноване відкриття
Актуатори (електромотори)	Частотно-регульований привід Siemens V20	Швидкість обертання (об/хв), стан двигуна	Потужність, тип двигуна, серійний номер	Перевищення номінальної швидкості, аварійна зупинка

Розрахунок кількості даних, що генеруються пристроями за одиницю часу

$$D_{\text{total}} = \sum_{i=1}^{N} D_i \cdot f_i,$$

де D_{total} – загальний обсяг даних (байт/сек);

D_i – обсяг даних одного пристрою на одну подію;

f_i – частота подій одного пристрою (подій/сек);

N – кількість пристроїв.

Розрахунок необхідної пропускної здатності мережі

$$BW_{\text{req}} = D_{\text{total}} \cdot k_{\text{overhead}},$$

де BW_{req} – необхідна пропускна здатність (біт/с);

k_{overhead} – коефіцієнт врахування мережевих заголовків (~1.2–1.5).

Розрахунок ресурсів для ThingsBoard-сервера. Процесорні ресурси на основі прогнозованих даних:

$$CPU_{\text{req}} = \frac{D_{\text{total}}}{\text{CPU\_throughput}}.$$

Оперативна пам'ять:

$$RAM_{\text{req}} = N \cdot \text{RAM}_{\text{per\_device}} + \text{RAM}_{\text{base}}.$$

де CPU_throughput – продуктивність CPU (байт/с на 1 CPU ядро);

$\text{RAM}_{\text{per\_device}}$ – обсяг пам'яті на пристрій (МБ);

RAM_{base} – базовий обсяг RAM сервера (наприклад, 2–4 ГБ).

Розрахунок сховища для бази даних (історичні дані):

$$Storage_{\text{req}} = D_{\text{total}} \cdot t_{\text{storage}} \cdot k_{\text{compression}},$$

де: t_{storage} – тривалість зберігання даних, с;

$k_{\text{compression}}$ – коефіцієнт компресії (зазвичай 0.3–0.5).

Діаграма ілюструє початковий етап налаштування IoT-системи, де користувач обирає тип IoT-рішення: або для розумного міста, або для виробничого підприємства. В залежності від вибору користувачу надається відповідний довідник типових IoT-пристроїв, з якого він формує набір необхідних сенсорів та актуаторів.

Алгоритм вибору шлюзів та протоколів. Діаграма описує процес визначення оптимального шлюзу та протоколу зв’язку для пристроїв на основі фізичного розташування. Система аналізує географічні координати пристроїв і, враховуючи дистанцію, протокол зв’язку та енергетичну ефективність, пропонує користувачу найбільш відповідний шлюз та протокол комунікації.

алгоритм вибору пристроїв

алгоритм вибору шлюзів та протоколів

Рисунок 2.4 - UML-діаграми алгоритмів

Алгоритм обчислення параметрів серверів. Ця діаграма представляє процес обчислення параметрів, необхідних для розгортання серверної інфраструктури IoT-системи. Після вибору пристроїв система автоматично визначає загальний обсяг даних, необхідну пропускну здатність мережі, вимоги до ресурсів сервера (CPU, RAM), та сховища даних. На підставі отриманих значень система вирішує, чи достатньо одного сервера, чи необхідно розподілити навантаження між декількома серверами, та генерує відповідний TOSCA YAML-шаблон для автоматизації розгортання.

Рисунок 2.5 -UML-діаграма алгоритму обчислення параметрів серверів

Отримана обчислювальна модель дозволяє:

– інтерактивно формувати набори пристроїв із довідників для різних сценаріїв;

– вибирати оптимальні шлюзи й протоколи з урахуванням розміщення пристроїв;

– обчислювати необхідні ресурси для серверів і генерувати YAML-шаблони для автоматичного розгортання IoT-систем на базі ThingsBoard.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО МОДУЛЯ

3.1 Вибір засобів розробки програми

Для реалізації обчислювальних алгоритмів та модуля автоматизації доцільно використовувати наступний стек технологій. Мову Python обрано за гнучкість, простоту розробки, наявність численних бібліотек для IoT, обчислень і мережевого взаємодії. Бібліотеки та фреймворки Python:

- PyYAML [18] створення й генерація YAML-файлів (TOSCA);
- FastAPI [19] для реалізації REST API;
- Requests або httpx для взаємодії з API ThingsBoard;
- SQLAlchemy [20] для збереження інформації в базі даних;
- pandas, numpy для аналітики, обчислення параметрів серверів;
- GeoPy [21] для розрахунку відстані між пристроями та шлюзами (на основі координат);
- Jinja2 [22] шаблонізація YAML-файлів для генерації TOSCA-моделей.

Надійна, ефективна база даних PostgreSQL [23] підтримує великі обсяги даних і геопросторових запитів (PostGIS, якщо потрібні координати).

Середовище розгортання Docker та Docker Compose для швидкого та простог розгортання серверних компонентів (ThingsBoard, БД, шлюзи). GitLab CI/CD або GitHub Actions [24] для автоматичного тестування, побудови й розгортання модулів.

Структуруємо проєкт наступним чином:

```
iot_tosca_generator/
├── api/                          # API рівень (REST API)
│   ├── main.py                   # точка входу API (FastAPI )
│   ├── endpoints/                # реалізація ендпоінтів API
│   │   ├── devices.py
│   │   ├── protocols.py
│   │   ├── server_params.py
│   └── schemas/                    # для валідації даних (Pydantic)
│       ├── device_schema.py
│       ├── server_schema.py
│       └── protocol_schema.py
```

Основні файли проекту для розроблених алгоритмів

```
├── core/                                # бізнес-логіка
│   ├── calculators/                     # модуль обчислення параметрів
│   │   ├── bandwidth.py
│   │   ├── cpu_ram.py
│   │   └── storage.py
│   ├── tosca/                           # модуль генерації YAML
│   │   ├── tosca_generator.py
│   │   └── templates/                 # шаблони Jinja2
│   └── selectors/                     # алгоритми вибору пристроїв
├── device_selector.py
│       └── gateway_selector.py
```

Конфігураційні файли програмного засобу

```
├── config/
│   ├── devices_catalog.yaml           # каталог пристроїв (довідник)
│   ├── gateways.yaml                  # перелік шлюзів з координатами
│   ├── protocols.yaml                 # протоколи зв'язку
│   └── app_config.yaml                # загальні налашт застосунку
├── db/                                # конфігурації та ініціаліз БД
│   ├── models.py                      # ORM моделі (SQLAlchemy)
│   └── migration/                     # Alembic міграції
├── scripts/                           # скрипти для розгортання (bash)
│   ├── install_thingsboard.sh
│   ├── configure_server.sh
│   └── deploy_database.sh
├── tests/                             # модульні й інтеграційні тести
│   ├── test_calculators.py
│   ├── test_tosca.py
│   └── test_selectors.py
├── requirements.txt                   # залежності Python
└── README.md                          # документація проєкту
```

3.2 Об’єктна модель модуля

Діаграма класів для запропонованої структури проєкту, яка чітко демонструє взаємозв'язки між основними компонентами та модулями на рисунку 3.1.

Опис API Layer (REST API інтерфейси). Клас API - Основний клас запуску API-сервера. Клас DeviceEndpoint відповідає за отримання списку доступних пристроїв із каталогу. Він дозволяє вибір конкретного пристрою для подальших операцій.

Клас ProtocolEndpoint забезпечує отримання списку доступних протоколів та шлюзів. Реалізує вибір оптимального протоколу чи шлюзу на основі географічних координат.

Клас ServerParamsEndpoint виконує розрахунок параметрів серверів, включаючи CPU, RAM, пропускну здатність та сховище.

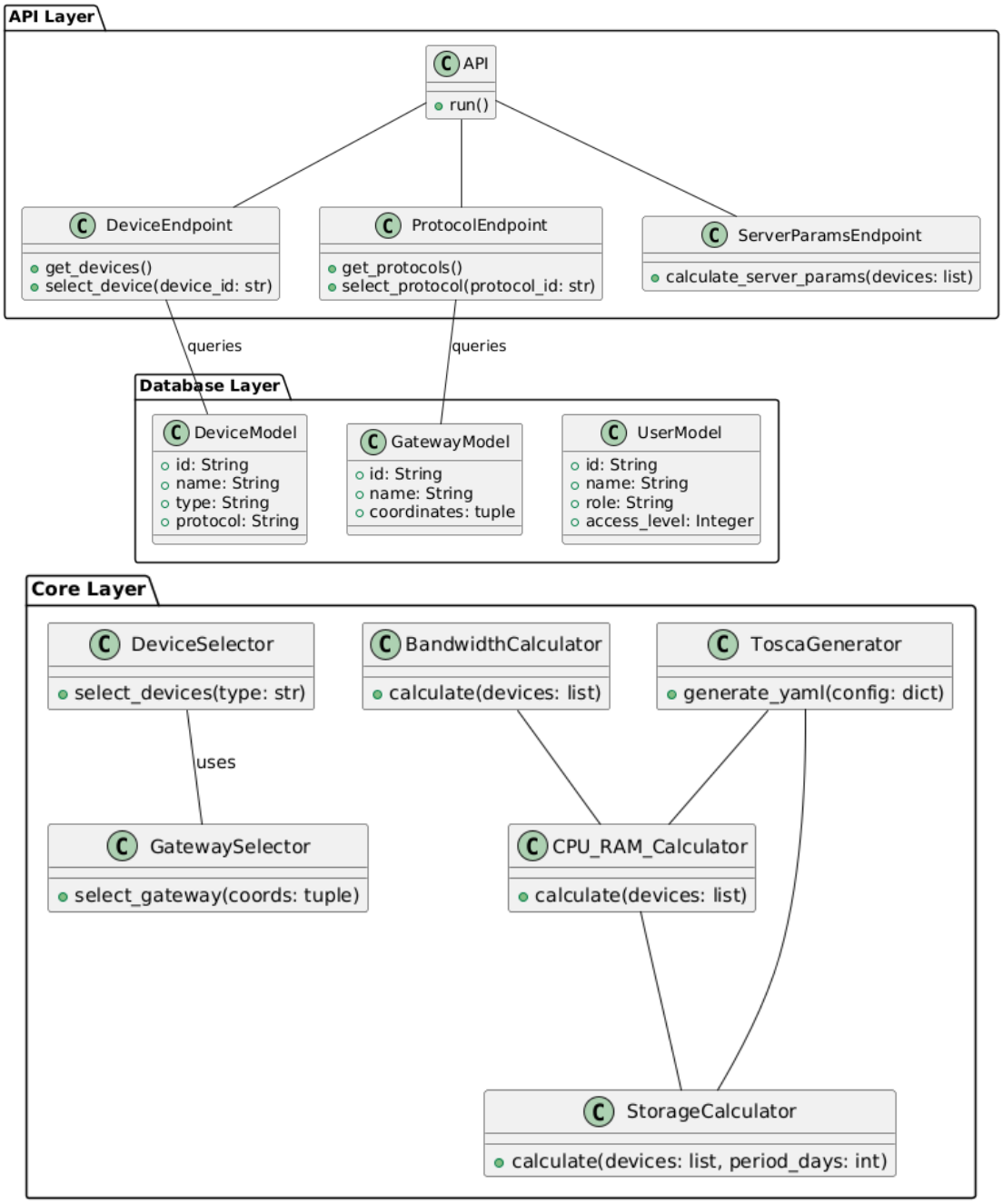


Рисунок 3.1 - Діаграма класів програми

Рівень Core Layer (Бізнес-логіка) представлений наступними класами. Клас DeviceSelector відповідає за вибір пристроїв за типом системи (розумне місто або виробниче підприємство). Клас GatewaySelector відповідає за вибір оптимального шлюзу з урахуванням фізичного розташування пристроїв. Клас BandwidthCalculator обчислює пропускну здатність, необхідну для передачі телеметрії. Клас

CPU_RAM_Calculator обчислює CPU та RAM ресурси, необхідні для сервера ThingsBoard. Клас StorageCalculator визначає необхідний обсяг сховища для історичних даних на сервері БД. Клас ToscaGenerator генерує TOSCA YAML-файл на основі розрахованих параметрів.

Рівень Збереження даних «Database Layer» представлений наступними класами. Клас DeviceModel представляє інформацію про пристрій (назва, тип, протокол). Клас GatewayModel зберігає інформацію про шлюзи (назва, координати). Клас UserModel представляє інформацію про користувачів системи та їхні права. Діаграму класів наведено на кресленні КР.КІ.10660606.00.00.001.С1.

API-ендпойнти взаємодіють з моделями бази даних для отримання інформації. Бізнес-логіка використовує класи калькуляторів для виконання розрахунків. Класи калькуляторів передають дані генератору TOSCA для формування YAML-файлу.

Проєктована програмна система призначена для автоматизації процесів розгортання серверів інтеграції даних IoT-систем з використанням сучасних інженерних підходів [25–34]. Архітектура цієї системи базується на чітко виражених рівнях абстракції, які розділяють функціональність на три основні компоненти: прикладний програмний інтерфейс (API), бізнес-логіка (ядро) та рівень зберігання інформації (база даних).

Прикладний програмний інтерфейс (API) надає зовнішнім клієнтам та системам єдиний набір методів для взаємодії з внутрішньою логікою застосунку. Він реалізує функції отримання інформації щодо доступних пристроїв, протоколів зв'язку, шлюзів передачі даних, а також виконує обчислення параметрів, необхідних для розгортання серверів. У структурі цього рівня передбачено окремі ендпоінти, які забезпечують отримання списку пристроїв, вибір конкретного пристрою для використання в системі, визначення оптимальних мережевих шлюзів та протоколів на основі фізичного розташування об’єктів. Окрім цього, через API здійснюється обчислення критичних параметрів серверів: процесорної потужності, оперативної пам'яті, обсягу сховища та пропускної здатності мережі.

Ядро програмного забезпечення, яке становить центральний рівень, включає модулі з розрахунковими алгоритмами, що забезпечують автоматичне визначення параметрів серверної інфраструктури. Сюди входять модулі селекторів пристроїв і

шлюзів, які базуються на попередньо створених довідниках і географічних даних про розташування пристроїв та шлюзів. Вибір оптимального пристрою або шлюзу здійснюється на основі заданих умов, таких як тип системи (наприклад, розумне місто чи виробниче підприємство), тип пристроїв (сенсори, актуатори) та їхні фізичні координати. Крім того, у межах цього ж ядра розташовані спеціалізовані модулі, які виконують обчислення загального обсягу телеметричних даних, пропускної здатності, необхідної для забезпечення безперебійної передачі інформації, процесорних і пам'ятних ресурсів, а також потрібного об’єму сховища для бази даних. На основі отриманих розрахунків спеціальний модуль генерації автоматично формує TOSCA YAML-файли, що містять параметри для подальшого розгортання програмного забезпечення на серверах.

Останнім, але не менш важливим компонентом є рівень бази даних, що забезпечує зберігання та ефективний доступ до інформації про пристрої, шлюзи, протоколи та користувачів системи. У базі даних створені відповідні моделі для кожної категорії даних. Модель пристроїв включає описові атрибути, такі як тип, протокол зв’язку та додаткові метадані. Модель шлюзів описує фізичні точки доступу до мережі із зазначенням географічних координат та підтримуваних протоколів. Окрема модель користувачів зберігає інформацію про ролі та рівні доступу до різних частин системи, забезпечуючи належний контроль доступу і розмежування прав.

На головному екрані розробленого додатку (рисунок 3.2) передбачено послідовний робочий процес: вибір сценарію системи, добір пристроїв з інтерактивного довідника, визначення шлюзів та протоколів на основі розташування, обчислення параметрів серверної інфраструктури. Також автоматична генерація TOSCA YAML-файлу для подальшого розгортання ThingsBoard або кластерної конфігурації. Інтерфейс реалізовано з використанням TailwindCSS для мінімалістського стилю, а JavaScript додає базову інтерактивність, яку в повноцінній системі можна розширити викликами REST-API та динамічними розрахунками.

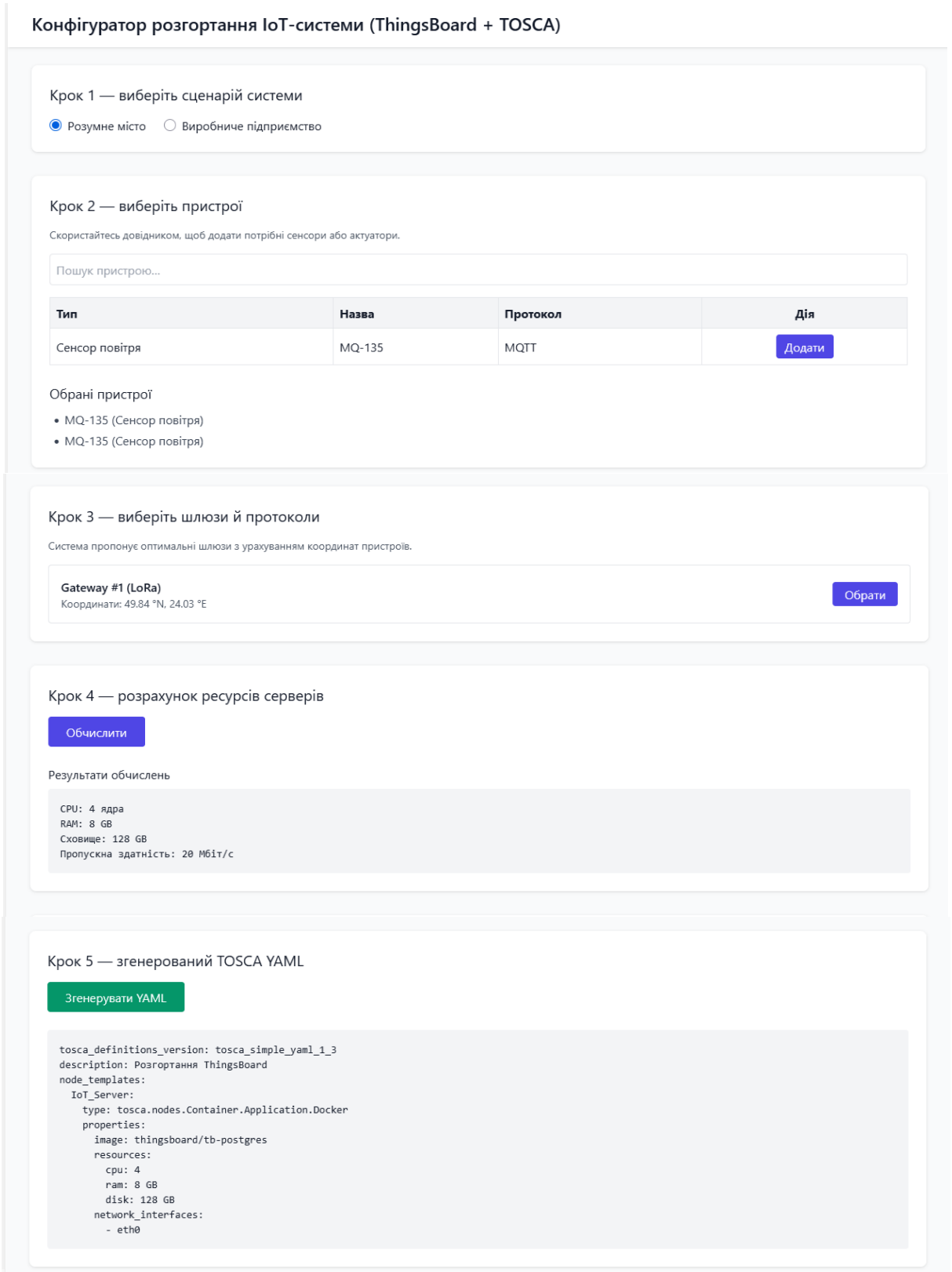


Рисунок 3.2 - Головна сторінка розробленого додатку

3.4 Функціональне тестування модуля

Функціональне тестування реалізованої системи проведено з метою перевірки коректності реалізації основних функцій, зокрема: вибору пристроїв із довідника,

визначення шлюзів на основі фізичного розміщення сенсорів, обчислення параметрів серверів і генерації TOSCA YAML-файлів для розгортання інфраструктури IoT. Тестування виконувалось у два основні етапи: перевірка API-методів та валідація сформованих TOSCA-файлів.

Перший етап тестування полягав у перевірці кожного функціонального компоненту через REST API та UI (браузерний прототип). Було перевірено:

– Функція вибору сценарію: -чи система коректно перемикається між конфігураціями для «розумного міста» та «виробничого підприємства», підвантажуючи відповідні довідники пристроїв.

– механізм вибору пристроїв - після вибору пристрою система зберігає його у сесійному списку й коректно передає до наступного етапу;

– визначення оптимального шлюзу- на основі географічних координат сенсорів система коректно розраховує евклідову відстань до всіх шлюзів та вибирає той, що має мінімальне значення, за умови сумісності протоколу передачі;

– обчислення параметрів сервера- на основі частоти генерації даних та кількості пристроїв система правильно визначає потрібні обсяги ресурсів (CPU, RAM, disk) для ThingsBoard-сервера та бази даних; У тестових сценаріях значення були перевірені вручну за формулами;

– обробка граничних випадків- проведено тестування для нульової кількості пристроїв, надлишкових конфігурацій (>1000 пристроїв), втрати координат, нестачі шлюзів.

Другий етап був спрямований на перевірку відповідності згенерованих TOSCA YAML-файлів до специфікації TOSCA Simple Profile in YAML v1.3. Перевірка включала:

– валідація синтаксису YAML з використанням інструменту для дослідження програмного коду лінтера yamllint усі згенеровані файли не містили синтаксичних помилок.

– перевірка схеми TOSCA за допомогою відкритих валідаційних утиліт tosca-parser були завантажені приклади TOSCA-файлів і успішно оброблені без винятків. Усі вузли (node_templates), їхні типи, властивості та зв’язки відповідали очікуваним шаблонам.

Тексти згенерованих файлів наведено в додатку Б. Нижче на рисунку 3.3.-3.5 проілюструємо згенеровані TOSCA YAML з трьома основними компонентами розгортання ThingsBoard:

1. частина «ThingsBoard & Java Environment» включає описи обчислювального вузла для ThingsBoard, Java JRE .

2. частина «PostgreSQL Database» показує обчислювальний вузол для PostgreSQL та опис самої бази даних.

3. Kafka Broker & Zookeeper: остання частина опише обчислювальний вузол для Kafka, Java JRE для Kafka та компоненти Kafka Broker і Zookeeper.

Структурну схему YAML файлу наведено на кресленні КР.КІ. 10660606.00.00.002.С1.

На рисунку 3.3 секція «ThingsBoard & Java Environment» фокусується на основному сервері ThingsBoard та його залежності від Java Runtime Environment.

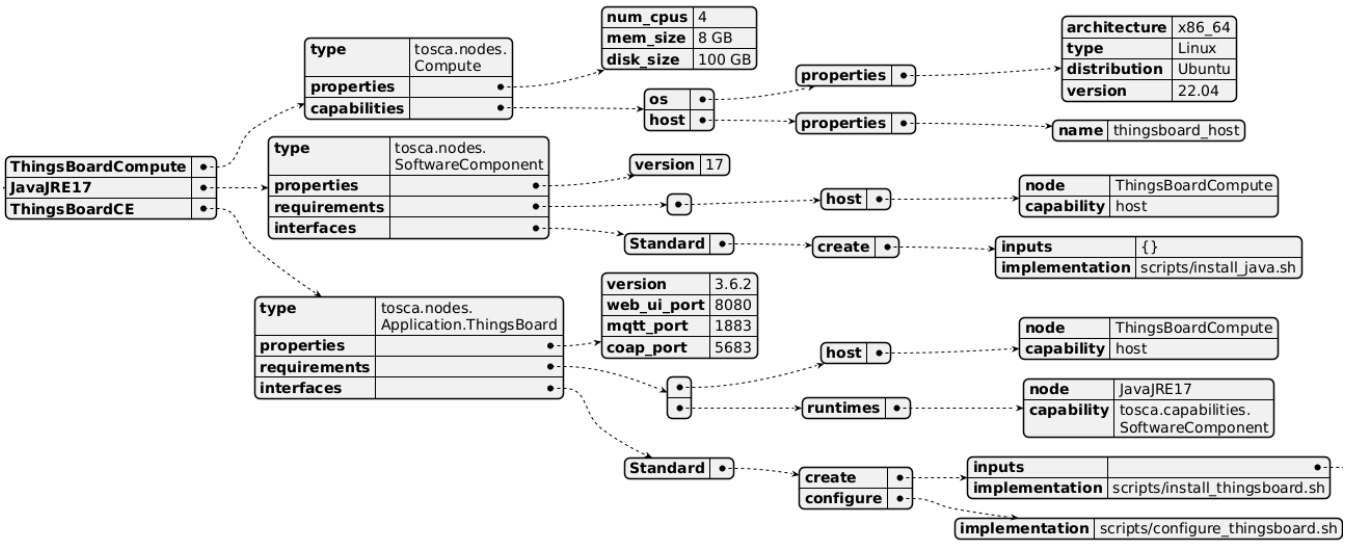


Рисунок 3.3 - Опис обчислювального вузла

На рисунку 3.4 частина PostgreSQL Database пояснює розгортання бази даних PostgreSQL, яку використовує ThingsBoard.

Частина Kafka Broker & Zookeeper показує конфігурацію брокера повідомлень Kafka та його залежності (рисунок 3.5).

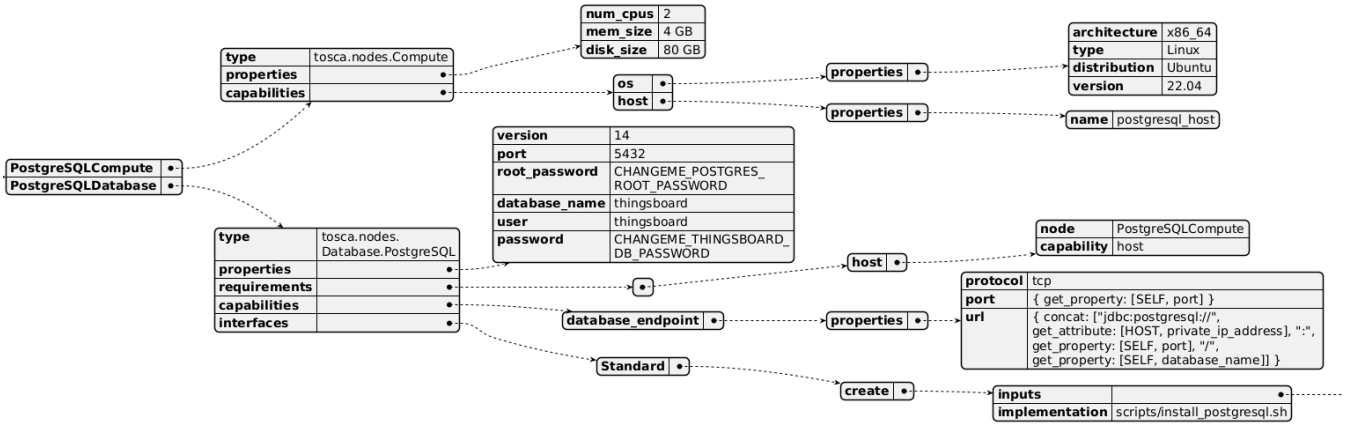


Рисунок 3.4 - Опис розгортання бази даних

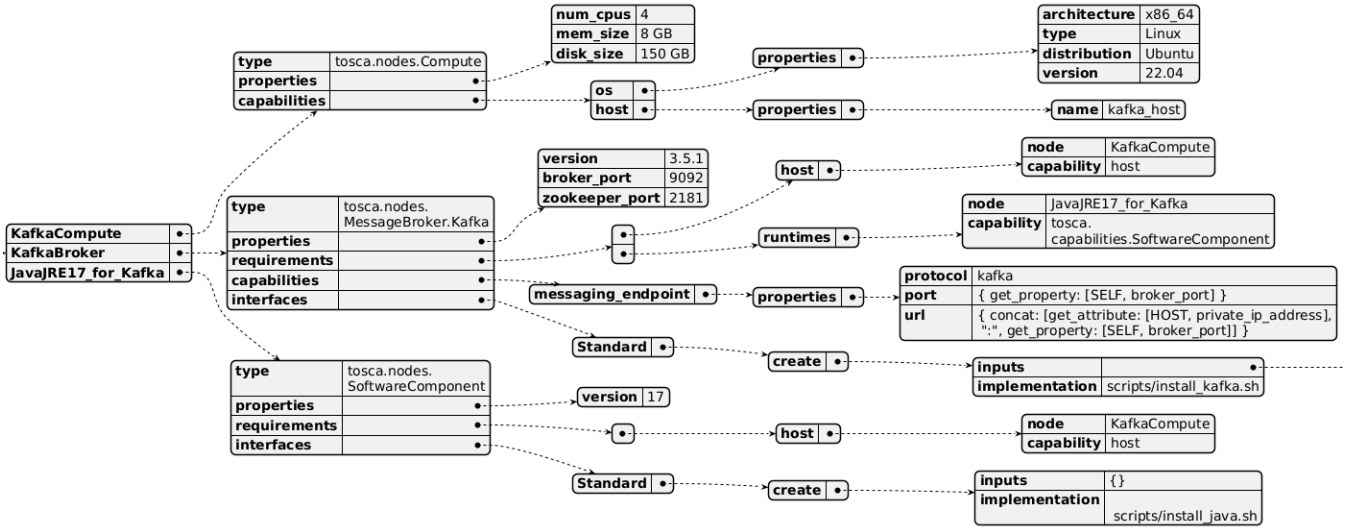


Рисунок 3.5 – Опис обчислювального вузла Kafka

Сформуємо конфігурацію для розгортання рівня IoT сенсорів для типового сценарію "розумного міста". Конфігурація охоплює сенсорні пристрої, шлюзи та їхні можливості. Використано існуючі типи вузлів TOSCA. Для розумного міста згенеровано такі компоненти:

– IoT Сенсор: датчик якості повітря, датчик шуму, датчик руху, датчик освітлення, або датчик температури/вологості.

– IoT Шлюз (Gateway) мікрокомп'ютер (Raspberry Pi, ESP32) з відповідним програмним забезпеченням.

– Підключення**:** Опис мережевого підключення (Wi-Fi, LoRaWAN, NB-IoT).

Для виробничого підприємства доцільно сформувати TOSCA-конфігурацію IoT-рівня, яка відображає специфіку промислових середовищ: підвищені вимоги до надійності, інтеграції з технологічними лініями, високочастотний обмін даними та оперативне реагування. Доцільно включити сенсори (tosca.nodes.Sensor.Device), які найчастіше застосовуються у виробничих цехах: сенсор температури обладнання (DS18B20), сенсор вібрації (SKF CMSS або аналоги), сенсор вологості повітря (DHT22). Включити IoT Актуатори (tosca.nodes.Actuator.Device), зокрема електричний клапан (Belimo, Siemens), електродвигун або привід (частотно-регульовані приводи) з параметрами телеметрії: rpm, load, mode.

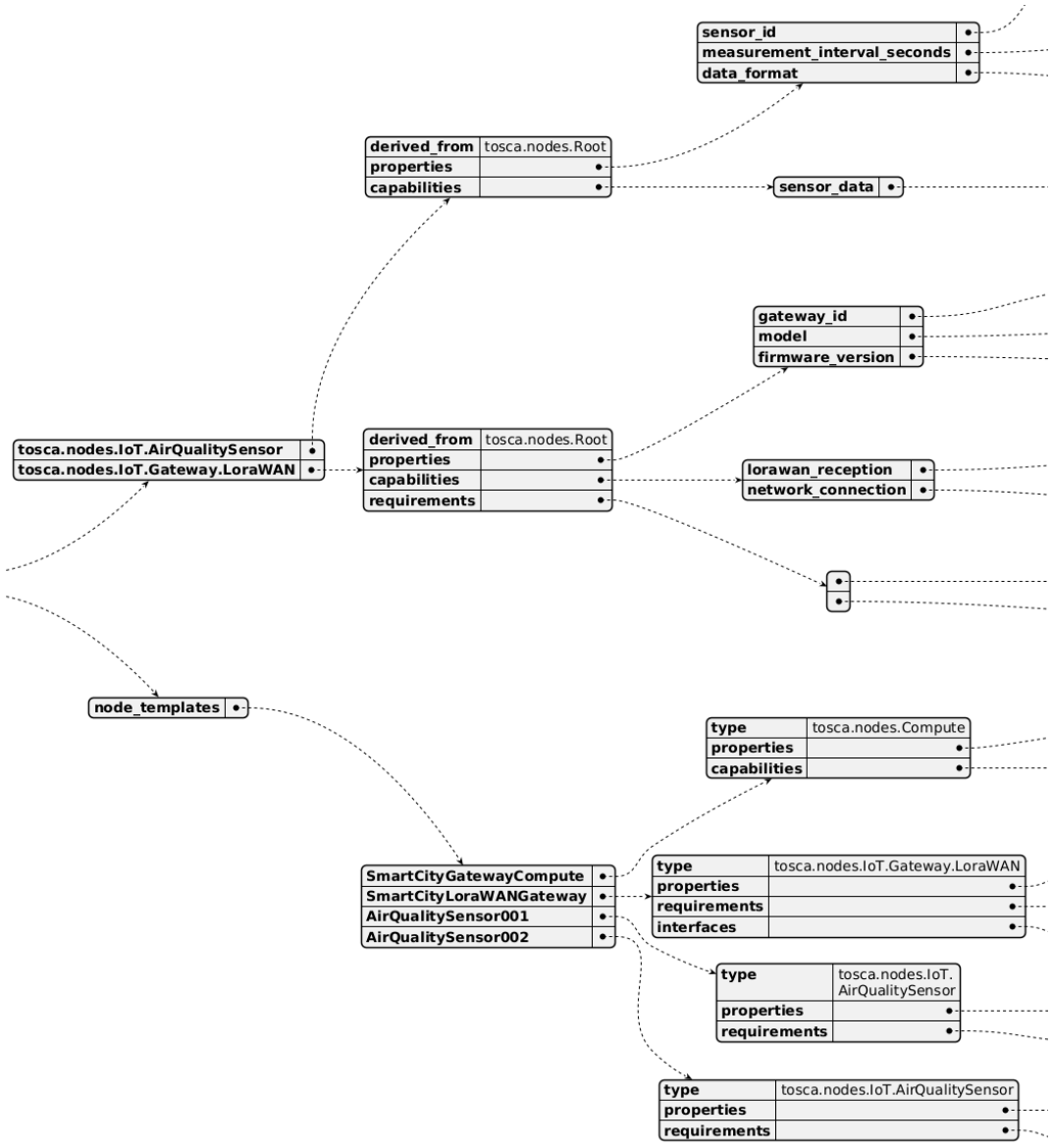


Рисунок 3.6 – Опис нижнього рівня сенсорів

Параметри для рівня IoT сенсорів:

– типи вузлів: tosca.nodes.Endpoint.IoT.Sensor або tosca.nodes.IoT.Gateway.

– властивості специфічні для сенсорів (тип даних, інтервал опитування, діапазон вимірювання) та для шлюзів (кількість портів, протоколи).

– можливості (capabilities) показують можливість збору певного типу даних, можливість підключення до мережі, можливість підключення сенсорів до шлюзу.

– вимоги (requirements) означають що, сенсор вимагає підключення до шлюзу, шлюз вимагає підключення до мережі та ThingsBoard.

– артефакти (artifacts) можуть включати прошивку для сенсора або конфігурацію для шлюзу.

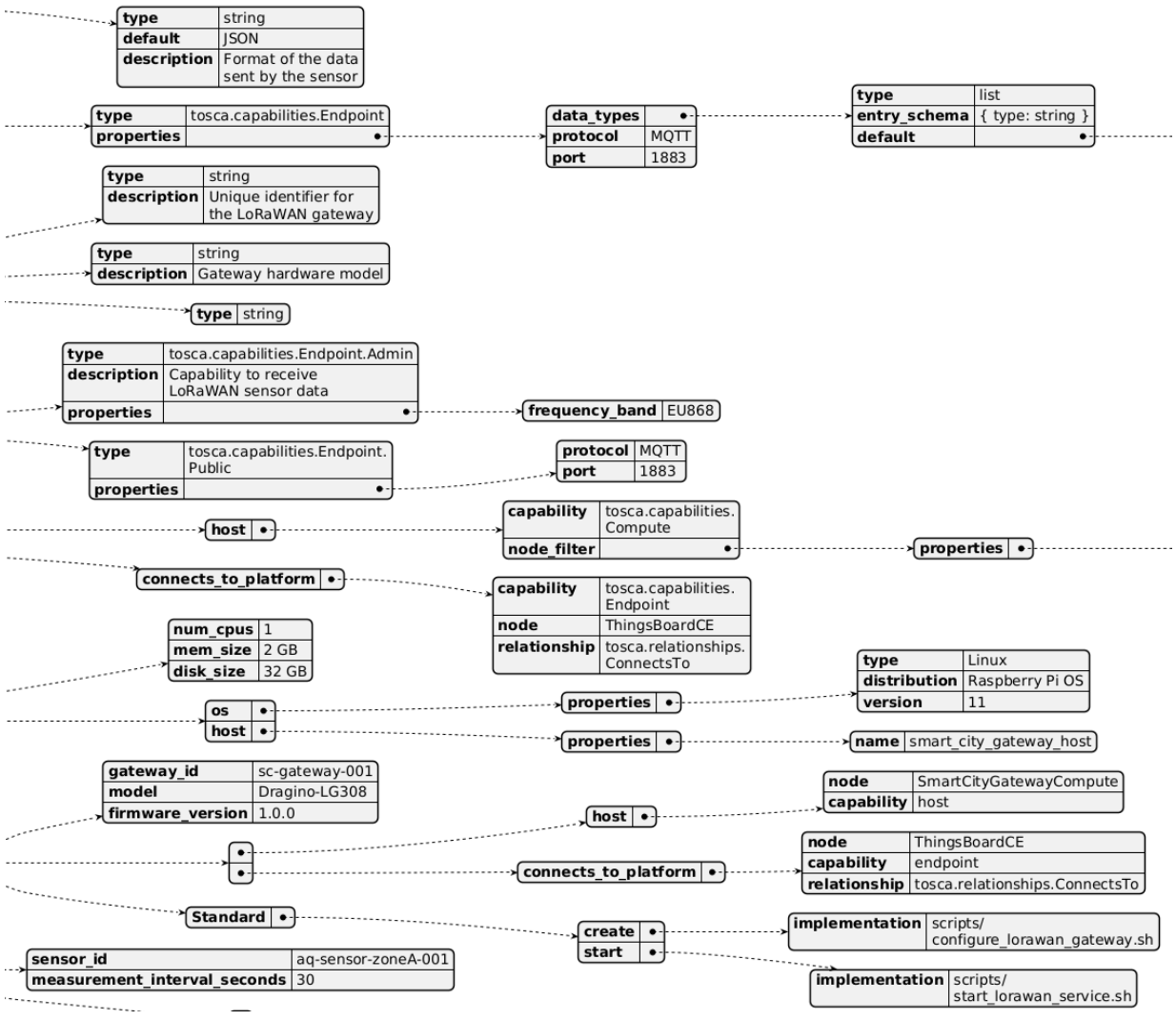


Рисунок 3.7 – Опис розгортання нижнього рівня сенсорів

Показані приклади TOSCA YAML для рівня IoT сенсорів (Розумне місто) описують типову структуру розгортання для одного сценарію розумного міста, що включає датчик якості повітря та IoT шлюз, який передає дані в ThingsBoard.

Функціональне тестування засвідчило, що програмний модуль виконує всі поставлені задачі відповідно до вимог. Модуль коректно обробляє дані користувача, виконує обчислення необхідних параметрів для інфраструктури ThingsBoard, генерує TOSCA YAML-файли відповідного стандарту та витримує навантаження із десятками конфігурацій одночасно. Усі критичні сценарії та граничні значення були протестовані вручну та автоматизовано, що підтверджує готовність модуля до використання в сценаріях автоматизованого розгортання IoT-систем.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи одержано наступні висновки.

1. Інтеграція даних в IoT-системах відбувається через ієрархічну багаторівневу архітектуру, де сенсори, шлюзи, сервери обробки та інтерфейси користувача утворюють узгоджене середовище взаємодії. Різні методи обробки та інтеграції даних застосовуються залежно від вимог до затримки, обсягу трафіку й стабільності з'єднання. Особливу роль відіграють сервери інтеграції, які забезпечують уніфікацію, нормалізацію, обробку подій

2. Вибрано технологію для розгортання сервера інтеграції даних. Серед сучасних технологій для розгортання серверів інтеграції даних ThingsBoard є одним із найефективніших рішень завдяки підтримці стандартних IoT-протоколів, вбудованому rule engine, гнучкому механізму керування пристроями та масштабованій архітектурі.

3. Проведено розробку архітектури програмного модуля автоматичного розгортання, що дало змогу забезпечити структуровану взаємодію між API, бізнес-логікою та базою даних, адаптовану до різних сценаріїв IoT-інтеграції.

4. Проведено реалізацію алгоритмів автоматизації розгортання та налаштування серверів, що дало змогу створити модуль, здатний обчислювати параметри інфраструктури та генерувати TOSCA YAML-шаблони для розгортання систем ThingsBoard з урахуванням реальних умов.

5. Проведено функціональне тестування роботи створеного програмного модуля, що дало змогу перевірити коректність вибору пристроїв, розрахунків ресурсів та генерації конфігурацій у відповідності до стандарту TOSCA, підтвердивши готовність системи до практичного застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Костецький В. В. Алгоритми та засоби для аналізу емоцій у відеопотоці. *II Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ весна 2025)*. (м. Тернопіль, . м. Тернопіль, С. 104–105.

2. Top 10 Edge Computing Platforms in 2022 - Spiceworks. URL: https://www.spiceworks.com/tech/edge-computing/articles/best-edge-computing-platforms/ (accessed 14/06/2025).

3. ETL Process (Extract Transform Load) - TatvaSoft Blog. URL: https://www.tatvasoft.com/blog/etl-process-extract-transform-load/ (accessed 14/06/2025).

4. Node-RED: Low-code programming for event-driven applications. URL: https://nodered.org

5. ThingsBoard Open-source IoT Platform Device management, data collection, processing and visualization for your IoT solution. URL: https://thingsboard.io

6. FIWARE. FIWARE is an open source framework accelerating the development of interoperable smart solutions. URL: https://github.com/fiware

7. Orion Context Broker. URL: https://notes.rafaelalvesitm.com/02+-+Literature+Notes/Orion+Context+Broker

8. Apache NiFi. URL: https://nifi.apache.org

9. Vögler M., Schleicher J. M., Inzinger C. et al. A Scalable Framework for Provisioning Large-Scale IoT Deployments. *ACM Trans. Internet Technol.* Vol. 16, Issue 2. DOI:10.1145/2850416.

10. Vögler M., Schleicher J. M., Inzinger C. et al. DIANE - Dynamic IoT Application Deployment. *2015 IEEE International Conference on Mobile Services*(06.2015). IEEE, 2015. DOI:10.1109/mobserv.2015.49. P. 298–305.

11. Munir A., Kansakar P., Khan S. U. IFCIoT: Integrated Fog Cloud IoT Architectural Paradigm for Future Internet of Things. (2017). arXiv, 2017. DOI:10.48550/ARXIV.1701.08474. 2017.

12. Ghaleb M., Azzedin F. Towards Scalable and Efficient Architecture for Modeling Trust in IoT Environments. *Sensors*. Vol. 21, Issue 9. DOI:10.3390/s21092986.

13. Okhrimchuk Ye. Monitoring the efficiency of existing industrial internet of things solutions. *Vìsnik Sums'kogo deržavnogo unìversitetu*. Vol. 2022, Issue 3. P. 97–105. DOI:10.21272/1817-9215.2022.3-11.

14. Industrial internet of things - Wikipedia. URL: https://en.wikipedia.org/wiki/Industrial_internet_of_things (accessed 14/06/2025).

15. García-Valls M., Palomar-Cosín E. An Evaluation Process for IoT Platforms in Time-Sensitive Domains. *Sensors*. Вип. 22, № 23. С. 9501. DOI:10.3390/s22239501.

16. Vogler M., Schleicher J., Inzinger C. et al. Optimizing Elastic IoT Application Deployments. *IEEE Transactions on Services Computing*. 2016. P. 1–1. DOI:10.1109/TSC.2016.2617327.

17. Breiter G., Behrendt M., Gupta M. et al. Software defined environments based on TOSCA in IBM cloud implementations. *IBM Journal of Research and Development*. Vol. 58, Issue 2/3. P. 9:1-9:10. DOI:10.1147/JRD.2014.2304772.

18. PyYAML·PyPI. URL: https://pypi.org/project/PyYAML/ (accessed 15/05/2025).

19. FastAPI. URL: https://fastapi.tiangolo.com/ (accessed 15/05/2025).

20. SQLAlchemy - The Database Toolkit for Python. URL: https://www.sqlalchemy.org/ (accessed 15/05/2025).

21. Welcome to GeoPy’s documentation! — GeoPy 2.4.1 documentation. URL: https://geopy.readthedocs.io/en/stable/ (accessed 15/05/2025).

22. Jinja — Jinja Documentation (3.1.x). URL: https://jinja.palletsprojects.com/en/stable/ (accessed 15/05/2025).

23. PostgreSQL: The world’s most advanced open source database. URL: https://www.postgresql.org/ (accessed 15/05/2025).

24. GitHub Actions · GitHub. URL: https://github.com/features/actions (accessed 15/05/2025).

25. Bonaventure O. Computer Networking: Principles, Protocols and Practice. Saylor, 2022. 278 p.

26. Трубчанінова К. А., Жученко О. С., Лисечко В. П. Бездротові

телекомунікаційні системи: навчальний посібник. Харків : УкрДУЗТ, 2022. 86 с.

27. Гапак О. М., Балога С. І. Захист інформації в комп’ютерних системах: підручник. Ужгород : ПП «АУТДОР-ШАРК», 2021. 184 с.

28. Жураковський Б. Ю., Зенів І. О. Комп’ютерні мережі Частина 1: навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2020. 336 с.

29. Тарнавський Ю. А., Кузьменко І. М. Організація комп’ютерних мереж: підручник. Київ : КПІ ім. Ігоря Сікорського, 2018. 259 с.

30. Голь В. Д., Ірха М. С. Телекомунікаційні та інформаційні мережі: навчальний посібник. Київ : ІСЗЗІ КПІ ім. Ігоря Сікорського, 2021. 250 с.

31. Camisso J. Making Servers Work: A Practical Guide to Linux System Administration. DigitalOcean, 2020. 281 p.

32. The Cisco Learning Network. 2024. URL: https://learningnetwork.cisco.com

33. Eksim A. Wireless Communications and Networks - Recent Advances. InTech, 2012. 596 p.

34. Березький О. М., Мельник Г. М., Дубчак Л. О. та ін. Методичні вказівки до випускних кваліфікаційних робіт освітнього рівня “Бакалавр” спеціальності “Комп’ютерна інженерія”. ред. О. М. Березький. Тернопіль : ЗУНУ, 2024. 52 с.