ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА
на тему: «РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ ІОТ»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро БІНСЬКИЙ___
(підпис)

Виконав: здобувач вищої освіти гр. ІСДМ-61
Дмитро Бінський

Керівник:
науковий ступінь, вчене звання
Каміла СТОРЧАК
Доктор технічних наук, професор

Рецензент:
науковий ступінь, вчене звання
Ольга ЗІНЧЕНКО
Доктор технічних наук, доцент

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти - магістр

Спеціальність 126 Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС
_____ Каміла СТОРЧАК
«____» _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бінський Дмитро Сергійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка системи автоматизації та візуалізації даних на основі технології iot

керівник кваліфікаційної роботи Каміла СТОРЧАК, д.т.н., проф.,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. № 145

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи:
3.1 Вимоги до кваліфікаційної роботи магістра з актуальних завдань спеціальності.
3.2 Існуючі інструменти автоматизації і візуалізації даних.
3.3 Технічні вимоги..

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
4.1 Аналіз засобів автоматизації і візуалізації.
4.2 Створення системи автоматизації та візуалізації даних на основі технологій IoT.
4.3 Результати дослідження.
4.4 Висновки.

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.10.23 – 05.11.23	
2	Вивчення матеріалів для аналізу розвитку технології IoT	06.11.23 – 12.11.23	
3	Дослідження сучасних рішень пов’язаних з технологією IoT	13.11.23 - 19.11.23	
4	Визначення технічного завдання	20.11.23 - 22.11.23	
5	Реалізація технічного завдання	23.11.23 - 12.12.23	
6	Вступ, висновки, реферат	13.12.23 - 15.12.23	
7	Розробка обов‘язкових демонстраційних матеріалів	16.12.23 - 20.12.23	
8	Попередній захист роботи	21.12.23 - 26.12.23	

Здобувач вищої освіти _____ Дмитро БІНСЬКИЙ
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина магістерської роботи: 79 сторінок, 22 рисунка, 4 джерела.

Об'єкт дослідження – процес автоматизації та візуалізації даних на основі технологій IoT.

Предмет дослідження – технологія автоматизації та візуалізації даних на основі технологій IoT.

Мета роботи – розробити систему автоматизації і візуалізації даних.

Методи дослідження – теорія IoT, теорія створення WEB додатків.

В роботі досліджено архітектуру WEB додатку, технології IoT та серверів, проаналізовано існуючі засоби відображення даних за допомогою додатків.

Також проведено порівняння існуючих засобів автоматизації і відображення даних за допомогою технології IoT. Створено WEB додаток для взаємодії з даними, розгорнута серверна частина та мікросервіси для роботи з даними.

Галузь використання – компанії, котрим важливо отримувати своєчасні дані з IoT пристроїв та реагування на них.

АВТОМАТИЗАЦІЯ, ВІЗУАЛІЗАЦІЯ, WEB ДОДАТОК, ТЕХНОЛОГІЯ IoT.

ABSTRACT

Text part of the master's qualification work: 79 pages, 22 pictures, 0 table, 4 sources.

Research object - data automation and visualization systems based on IoT technologies.

Research subject - data automation and visualization technology based on IoT technologies.

The goal of the work - to develop a data automation and visualization system.

Research methods - IoT theory, web application development theory.

The research explores the architecture of a web application, IoT technologies, and servers. Existing tools for data display through applications are analyzed.

A comparison of existing automation and data display tools using IoT technology is also conducted. A web application for interacting with data is created, and the server-side and microservices for data processing are deployed.

Application area - companies that need timely data from IoT devices and the ability to respond to them.

AUTOMATION, VISUALIZATION, WEB APPLICATION, IoT TECHNOLOGY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ ПОЗНАЧЕНЬ

MQTT	MQ Telemetry Transport
IoT	Internet of Things
TTI	The Things Industries
TTS	The Things Stack
SPA	Single Page Application

ВСТУП

Відтоді, як у світі з'явилися IoT-пристрої, пройшло значно більше часу, і тепер велика частина нашого життя насичена зручними технологіями у сфері розумних будинків та автоматизації виробничих процесів. На сьогоднішній день важко уявити існування без широкого асортименту IoT-гаджетів, які вносять значний внесок у різні аспекти нашого повсякденного життя, такі як медицина, виробництво продуктів, аграрний сектор і побутове господарювання. У сфері IT автоматизація грає важливу роль, забезпечуючи швидкість реакції на різні умови.

Основною метою цього дипломного проекту є проведення дослідження та практична реалізація системи для зберігання та відображення даних. Серед задач проекту визначено:

Задачами проекту є:

- Дослідити наявні засоби передачі даних з IoT-пристроїв;
- Вивчити існуючі технології зберігання отриманих даних від датчиків;
- Розглянути наявні технічні рішення для реалізації проекту;
- Обґрунтувати вибір конкретного засобу для реалізації проекту;
- Створити систему з використанням обраного засобу;
- На основі проведених досліджень сформулювати остаточні висновки.

Впровадження такої системи значно спростить роботу компаній у сферах продуктового виробництва та медицини, а також покращить швидкість реакції працівників на непередбачені умови зберігання різних продуктів.

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ АВТОМАТИЗОВАНОГО ТА ВІЗУАЛІЗАЦІЇ ДАНИ

1.1 Огляд існуючих технологій отримання даних з пристроїв ioT

З метою здобуття більшої гнучкості та використання досягнень цифровізації, компанії активно впроваджують автоматизацію IT-процесів, яка стала необхідною нормою. Тепер цей вид автоматизації, який раніше розглядався як простий набір інструментів або сценаріїв для допомоги бізнесу, визнається як стратегічна ініціатива та ключова складова довгострокової IT-стратегії.

Згідно з опитуванням Harvard Business Review Analytic Services, проведеним серед 338 керівників компаній у всьому світі, автоматизація IT визнається надзвичайно важливою для успіху організації: 80% респондентів визнали впровадження IT-автоматизації як "надзвичайно важливе" або "дуже важливе".

Серед IT-процесів, які найчастіше піддають автоматизації, визначаються прості рутинні IT-процеси.

Рисунок 1.1 - Поширення автоматизації IT-процесів

1.1.1 WiFi connection

Wi-Fi відігравав ключову роль у розвитку інновацій в галузі IoT, забезпечуючи всюдисуще підключення для зв'язку різноманітних "речей" між собою, до Інтернету та до 19,5 мільярдів пристроїв Wi-Fi, що використовуються по всьому світу. Економічний потенціал Інтернету речей є необмеженим, і Wi-Fi відкриває широкий спектр можливостей у різних секторах, таких як розумний дім, розумне місто, автомобільна галузь, охорона здоров'я, підприємства, уряд та промислові середовища Інтернету речей.

Wi-Fi дозволяє користувачам автоматизувати свої смарт-доми та підключати різноманітні об'єкти домашнього вжитку, відстежувати ланцюги постачання та інші критичні функції в реальному часі в промислових об'єктах, а також розблоковувати бізнес-вартість, підвищуючи продуктивність та ефективність як для підприємств, так і для гібридних робочих сценаріїв. Інтеграція та взаємодія, надана Wi-Fi, дозволять IoT-рішенням безпечно взаємодіяти одне з одним і з мільярдами пристроїв, орієнтованих на користувача, для максимального розкриття потенціалу від застосувань та середовищ Інтернету речей.

Wi-Fi є однією з найбільш широко використовуваних технологій у всьому світі, і інтеграція з існуючими мережами Wi-Fi, широкий спектр технологій і сильна спадщина взаємодії роблять очевидним вибір, щоб отримати найбільшу цінність від IoT. Wi-Fi буде відігравати важливу роль майже в кожному середовищі IoT, окремо або разом із більш спеціалізованими протоколами чи технологіями.

1.1.2 LoRaWAN

LoRaWAN (Long Range Wide Area Network) - це бездротовий протокол передачі даних в інтернеті речей (IoT). Цей протокол спеціально розроблений для використання в широкомасштабних мережах, які підтримують велику кількість пристроїв, розташованих на значній відстані від один одного.

Основні характеристики LoRaWAN:

1. LoRa (Long Range) - це технологія радіомодуляції, яка дозволяє досягати дальність зв'язку на значно великій відстані при невеликому споживанні енергії. Це робить LoRaWAN ефективним для батарейних пристроїв та використання в областях з обмеженим живленням.
2. Характеризується великою дальністю передачі, що дозволяє використовувати LoRaWAN для покриття великих територій, таких як міські або сільські райони.
3. Протокол розроблений з урахуванням потреб батарейних пристроїв, тому він оптимізований для низького споживання енергії. Це особливо важливо для пристроїв, які працюють вдалина від джерел енергії.
4. LoRaWAN може обслуговувати велику кількість пристроїв на одному вузлі базової станції, що робить його ефективним для великомасштабних IoT-мереж.
5. Протокол підтримує двунаправлений обмін даними, що дозволяє відправляти команди та отримувати підтвердження від пристроїв.
6. LoRaWAN використовує класи пристроїв (Class A, Class B, Class C), що дозволяє підлаштовувати мережеві властивості під конкретні вимоги додатку.
7. Розвивається та підтримується Лора Альянс, що забезпечує стандартизацію та сумісність пристроїв різних виробників.

Ці характеристики роблять LoRaWAN популярним вибором для великої кількості застосунків у сфері IoT, таких як віддалене відстеження, сільське господарство, міське управління, та інші, де потрібна дальність передачі даних та ефективне використання енергії.

1.1.3 The Things Industries

Заснована в 2015 році компанія The Things Industries є відомим у галузі експертом із корпоративних рішень LoRaWAN, які забезпечують стабільність, масштабованість і економічну ефективність, необхідні для IoT проектів.

The Things Stack — це мережевий сервер LoRaWAN, який забезпечує підключення, керування та моніторинг пристроїв, шлюзів і додатків кінцевих користувачів. Його метою є забезпечення безпеки, масштабованості та надійності маршрутизації даних у всій мережі.

TTI входить до складу LoRa Alliance.

Основні аспекти "The Things Industries":

1. LoRaWAN Platform: Компанія спеціалізується на розробці та підтримці платформ для роботи з LoRaWAN мережами. Це може включати в себе сервери мережі, системи керування пристроями, та інші інфраструктурні компоненти, необхідні для побудови та управління IoT-мережами.
2. The Things Network (TTN): TTI бере участь у розвитку і підтримці The Things Network - глобальної громадської LoRaWAN-мережі. Ця ініціатива має на меті створення відкритого та доступного інфраструктурного середовища для розвитку IoT-рішень.
3. Промислові Рішення: Крім публічних LoRaWAN-мереж, The Things Industries також надає рішення для промислових використань, таких як управління містом, сільське господарство, логістика та інші.
4. Компоненти та Інструменти: Крім програмного забезпечення, TTI також може надавати апаратні рішення, компоненти та інструменти для розгортання та управління мережами LoRaWAN.

1.1.4 Elsys

ELSYS надає датчики для розумних будівель, міст і промисловості. Датчики використовуються для контролю клімату в приміщенні, енергоефективності, динамічних робочих місць тощо.

Усі датчики ELSYS виробляються в ЄС, щоб зменшити наш вплив на клімат і гарантувати стандарти якості. ELSYS відома в усьому світі своєю якісною продукцією та постійно прагне до екологічно чистих продуктів і практик.

ELSYS було засновано в 2005 році кількома дослідниками та викладачами з факультету прикладної фізики та електроніки Університету Умео. Засновники привнесли свій досвід у сфері підключених пристроїв, і троє з них все ще працюють у компанії сьогодні.

Данна компанія створює такі датчики:

- Датчик температури
- Датчик вологості
- Датчик руху
- Датчик вуглекислого газу
- Датчик присутності
- Датчик рівня звуку
- Датчик світла
- Датчик витоку води
- Датчик прискорення
- Датчик тиску
- Датчик відстані.

Майже всі датчики, які були перелічині вище, будуть інтегровані в систему, відображати показники в реальному часі і зберігати історичні дані отримані з пристроїв.

1.2 Порівняння існуючих технологій отримання даних з пристроїв IoT

Переваги підключання пристроїв через WiFI:

- Базована на стандартах технологія взаємодії
- Всепроникна зв'язність
- Перевірена безпека WPA3

- Просте розгортання
- Зворотна сумісність
- Визначення місця розташування
- Надійне підключення
- Гнучка топологія мережі

Недоліки підключення пристроїв через WiFi:

- Споживання електроенергії
- Обмежений радіус дії
- Питання безпеки
- Обмежена масштабованість
- Високий обсяг використовуваної шириною смуги

На відміну від WiFI, LoRaWAN забезпечує велику дальність передачі даних, що робить його ефективним для додатків, які потребують довгого діапазону. Також протокол спроектований для ефективного використання енергії, що дозволяє пристроям працювати на довгому терміні без зарядки або заміни батарей. LoRaWAN працює на відмінних від технології WiFi частотах, що дозволяє створити стабільне підключення до пристроїв та дозволяє підключати велику кількість пристроїв до одного вузла, що робить його ефективним для розгалужених IoT-мереж.

Серед недоліків треба виділити обмежену швидкість передачі даних, що робить його менш ефективним для застосунків, де потрібна висока швидкість. Якщо велика кількість пристроїв використовує один канал, може виникнути обмеження в пропускній здатності. Для великих міст або об'єктів із великою площею може бути складно забезпечити достатнє покриття.

Враховуючи всі переваги та недоліки технології з’єднання LoRaWAN добре підходить для інфраструктур, орієнтованих на використання пристроїв які працюють від аккумулятора.

Щоб зменшити час на розробку сервера для передачі даних з пристроїв які будуть додані в систему буде використовуватись сервіс під назвою TTI. Він забезпечує нас готовим конфігураційним файлом для налаштування LoRaWAN

з’єднання пристрою, додавання його в мережу та методами управління ними. Такий самий підхід буде реалізований і для маршрутизаторів. Оскільки TTI пропонує безпечні і сучасні рішення в реалізації даних поставлених задач він був вибраний як основний сервіс для передачі даних з пристроїв до системи автоматизації і моніторингу даних.

1.3 Обґрунтування вибору засобу

Зрозуміло, що кожен інструмент має ідеальні сценарії використання, в яких його переваги виявляються особливо чітко. Наприклад, багато людей вважають Wi-Fi простим у встановленні, освоєнні та використанні - його підручники читаються легко і розуміються, дозволяючи досягти результатів за коротший час. Тим не менше, простота цього варіанту може призвести до того, що досвідчені користувачі виражають бажання отримати більше витонченості та розширених можливостей.

Простий/легкий у навчанні

Однією з найвдалих характеристик LoRaWAN є його висока ефективність в навчанні та швидка адаптація користувачів до його використання. Завдяки чіткій та доступній документації можна швидко освоїти робочий процес та логіку операцій у короткий проміжок часу. Крім того, протокол легко інтегрується з сервісом TTI для обробки даних, що значно прискорює створення повноцінної системи. Це дає можливість використовувати вже наявні підходи для майбутніх пристроїв, які можуть бути додані до системи, і забезпечує стабільне та безпечне з'єднання між пристроями та системою.

Враховуючи вище перераховані плюси, для створення системи автоматизації і візуалізації даних було обрано протокол LoRaWAN, сервір доставки даних TTI та пристрої від компанії Elsys.

2 ОГЛЯД ТЕХНІЧНИХ ЗАСОБІВ

2.1 Огляд фреймворку WEB додатку

2.1.1 Angular

Angular - інноваційний фреймворк для створення веб-додатків та SPA від Google, завоювавший широку популярність, реалізує шаблон MVC у розробці клієнтських застосунків. Цей потужний інструмент був випущений у 2016 році, а визначним аспектом Angular є використання мови програмування TypeScript.

Angular надає багатий функціонал, включаючи двосторонню прив'язку даних, що забезпечує зміну даних в одній частині інтерфейсу при модифікації моделі в іншій, а також шаблони, маршрутизацію та інші важливі можливості. Однією з особливостей є використання TypeScript, мови, представленої Microsoft у 2012 році, яка розширює можливості JavaScript.

Фреймворк Angular не лише служить платформою для розробки клієнтських додатків на базі HTML та TypeScript, але також впроваджує базову та додаткову функціональність у вигляді імпортованих бібліотек. Використання цього фреймворку спрощує архітектурне управління кодом, шаблонами проектування та різко зменшує час розробки програмного забезпечення.

Сучасні програмні рішення переважно базуються на використанні фреймворків і бібліотек, які дозволяють ефективно структурувати код та забезпечують єдиний стиль його написання. Це сприяє легкості обслуговування коду та полегшує спільну роботу розробників над різними його частинами. Бібліотеки, зазвичай, надають функціональність та API, які можуть бути використані розробниками у їхньому коді.

Приклади фреймворків:

- Angular - відомий своєю відкритістю, є інноваційним фреймворком для розробки веб-додатків. Він полегшує створення користувацьких компонентів, які легко вбудовувати в HTML-документи, а також реалізовувати логіку додавання. Його активне використання прив'язки даних, модуль впровадження залежностей, модульність і механізм налаштування маршрутизації робить його потужним інструментом для розробників.
- Ember.js - ще один фреймворк з відкритим вихідним кодом, будується на архітектурі MVC і також служить для створення веб-додатків. Його механізм маршрутизації та підтримка двосторонньої прив'язки даних роблять його ефективним інструментом для розробки, з використанням конвенцій у коді для підвищення продуктивності.
- Jasmine - знову ж таки фреймворк із відкритим вихідним кодом, спрямований на тестування JavaScript-коду. Цей інструмент не потребує DOM-об'єкта і має набір функцій для перевірки правильності виконання частин програми. Його часто використовують разом з Karma, програмою для запуску тестів, яка забезпечує можливість виконання перевірок в різних браузерах.

Прикладами бібліотек для розробки веб додатків можуть бути:

- jQuery - це відома бібліотека для мови програмування JavaScript, яка вирізняється своєю простотою використання та відсутністю потреби в суттєвих змінах у стилі написання коду для веб-програм. Вона дозволяє ефективно знаходити та маніпулювати об'єктами DOM, обробляти події браузера та вирішувати проблеми несумісності між браузерами. Завдяки своїй розширюваності, для jQuery було розроблено тисячі плагінів розробниками з усього світу.
- Bootstrap - бібліотека компонентів для створення користувацького інтерфейсу з відкритим вихідним кодом, розроблена компанією Twitter, визначається своєю адаптивністю веб-дизайну. Це робить її цінною для

веб-додатків, які повинні автоматично адаптувати свій дизайн до розміру екрану користувача.

- Material Design - це бібліотека компонентів від Google, яка може конкурувати з Bootstrap. Оптимізована для використання на різних пристроях, вона постачається з різноманітними елементами інтерфейсу для кінцевого користувача.
- React - розроблений компанією Facebook, представляє собою бібліотеку з відкритим вихідним кодом, спрямовану на побудову користувацьких інтерфейсів. Він використовує шар V у шаблоні MVC та впроваджує власний віртуальний об'єкт DOM, що полегшує доступ до браузерного DOM і підвищує продуктивність. React також використовує формат JSX, розширення синтаксису JavaScript, що нагадує XML.
- RxJS - це набір бібліотек, які використовуються для створення асинхронних та подійних програм з використанням спостережуваних колекцій. Вона надає можливість працювати з асинхронними потоками даних, такими як потік котирувань акцій або події, пов'язані з рухом миші. З RxJS потоки даних моделюються як спостережувані послідовності, і цю бібліотеку можна використовувати як з іншими фреймворками JavaScript, так і самостійно.

Основовою для створення структурних блоків у фреймворку Angular є використання модулів, які визначають контекст компіляції для компонентів. Модулі зберігають залежний код у вигляді функціональних наборів, а програма складається з набору таких модулів. Обов'язковою складовою є кореневий модуль, який відповідає за завантаження додатку, а також може існувати багато функціональних модулів.

Компоненти визначають відображення, яке представляє собою набір елементів екрану, здатних змінюватися та відображатися Angular відповідно до програмної логіки та даних.

Щоб відокремити логіку, не пов'язану з відображенням, компоненти використовують сервіси. Ці сервіси можуть бути підключені до компонентів як

залежності, роблячи код модульним та придатним для повторного використання. Як компоненти, так і сервіси є просто класами, які мають декоратори з метаданими, які вказують фреймворку, як їх використовувати.

Метадані класів компонентів пов'язані з їхнім шаблоном, що визначає представлення. У шаблоні поєднуються звичайний HTML разом з директивами та шаблонами прив'язки даних, що дозволяє Angular змінювати HTML перед його відображенням.

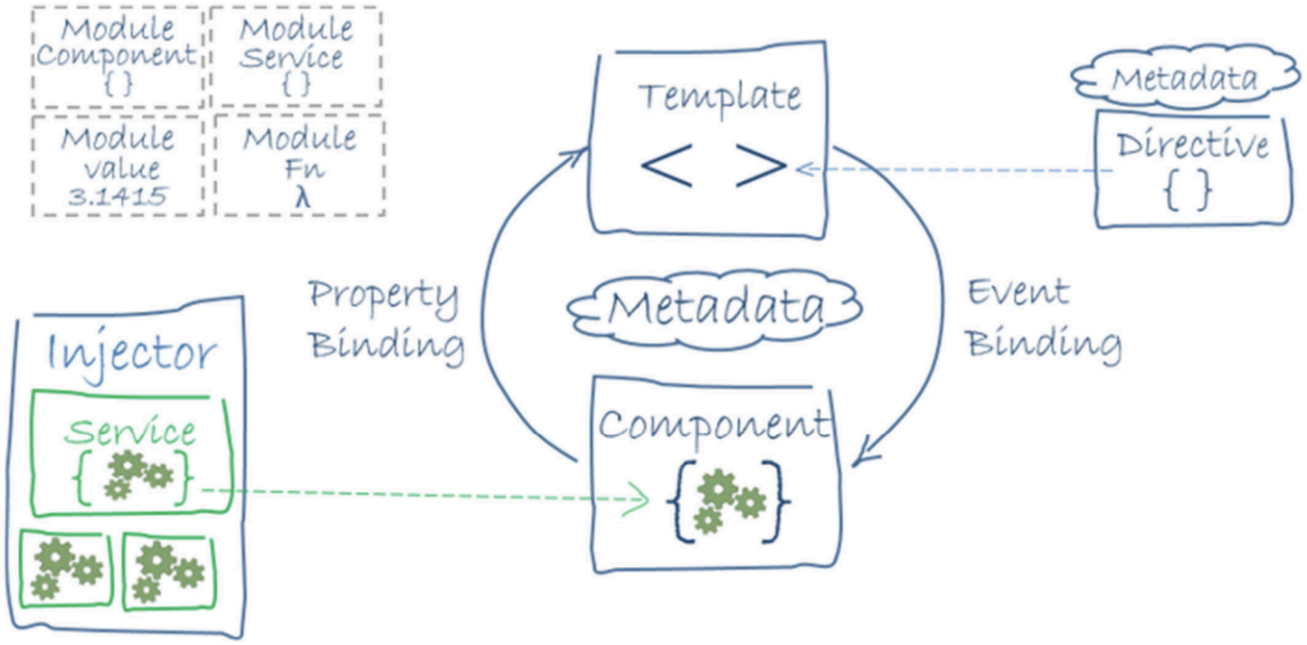

Рисунок 2.1 - Архітектура Angular програми

2.2 Огляд засобів отримання, збереження, передачі та візуалізації даних

На Рис.2.2 наведено загальну архітектуру передачі даних до їх кінцевого користувача

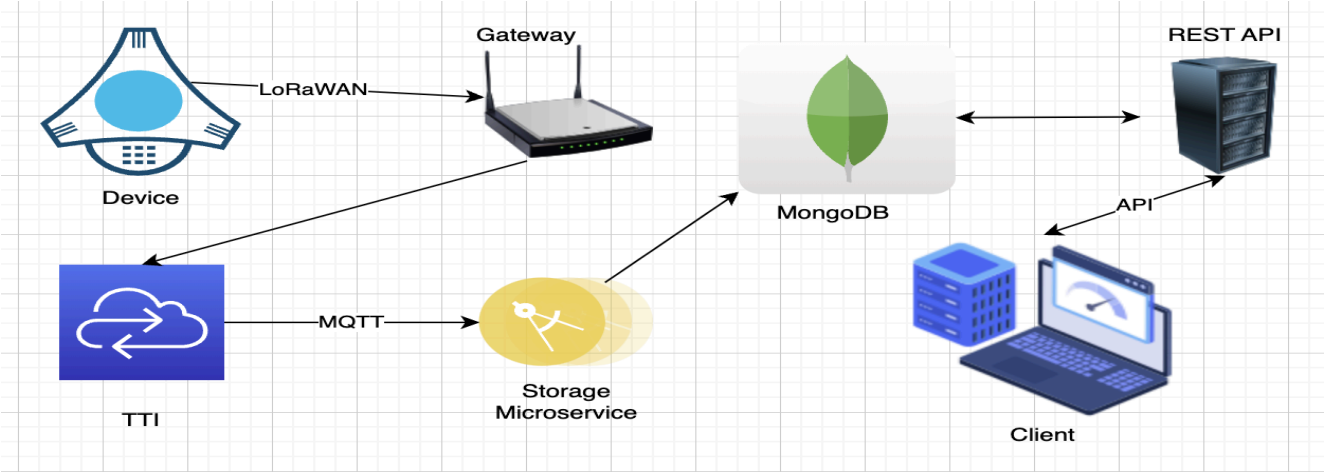

Рисунок 2.2 - Компоненти системи

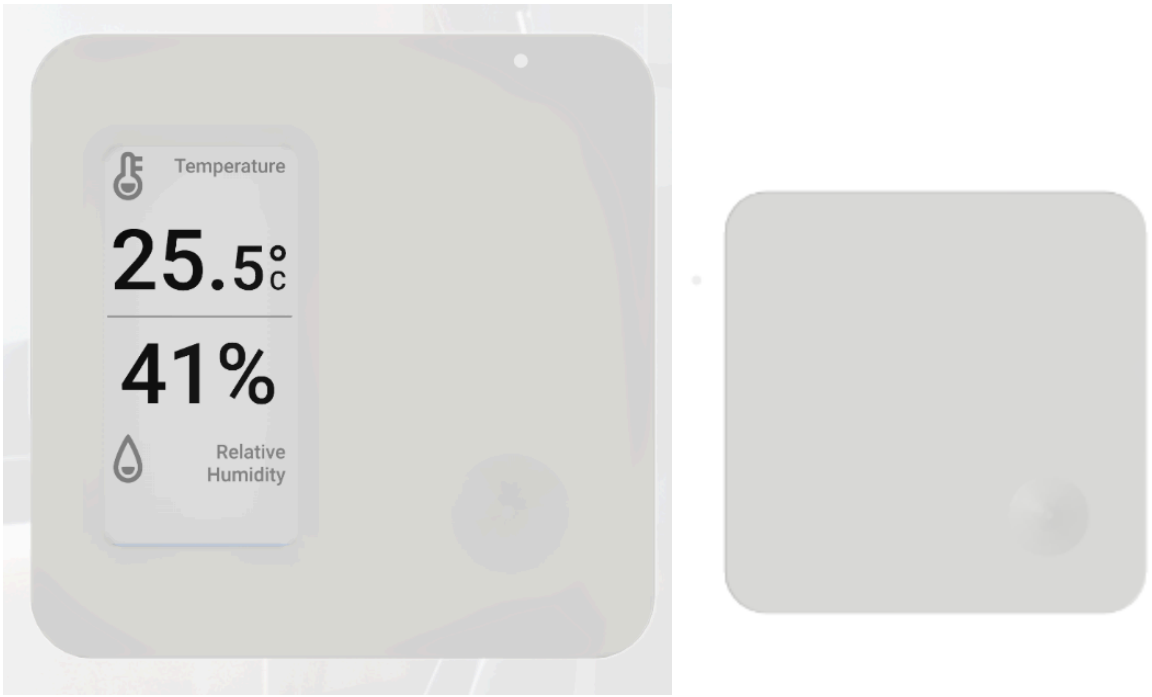

Рисунок 2.3 - Приклад пристрою з екраном і без

Пристрій - гаджет який займаєтесь контролюванням і передачею даних отриманих з датчиків, які він включає. Цей пристрій налаштовується до моменту встановлення його в певні умови, наприклад холодильник. Налаштування відбувається сетодом підключення девайсу до комп'ютера чи планшета на якому предвстановлені додатки, яку дозволяють змінювати конфігурацію пристрою. Найчастіше змінюється налаштування з'єднання з маршрутизатором, інтервал забору даниз з датчиків і інтервал їх відправлення до підключеного маршрутизатора.

Рисунок 2.4 - Маршрутизатор

Маршрутизатор - гаджет який забезпечує комунікацію між підключеними до нього пристроями і серверами обміну і збереження даних. Вибір оптимального місця для маршрутизатора LoRa (LoRaWAN маршрутизатор) важливий для ефективної роботи мережі. Ось деякі фактори, які слід врахувати при виборі місця для маршрутизатора LoRa:

- Висота розташування: Розташування маршрутизатора на високому місці може значно покращити дальність його сигналу та забезпечити кращий охоплення. Якщо можливо, маршрутизатор потрібно встановлювати на високій мачті або будівлі.
- Лінійна видимість: Треба переконатися, щоб між маршрутизатором та підключеними пристроями не було значних перешкод, таких як будівлі, гори або інші перешкоди. Це допоможе уникнути втрат сигналу та покращить зв'язок.
- Покриття території: Маршрутизатори потрібно розташовувати так, щоб їхня зони покриття перекривалися, забезпечуючи стабільний зв'язок на всій території.
- Зони ефективності: Сигнал LoRa може проникати крізь перешкоди, але вони можуть впливати на його якість та дальність. Це потрібно враховувати при розміщенні маршрутизаторів у місцях з підвищеною щільністю будівель чи густотою рослинності.

- Джерела спеціального електромагнітного шуму: Необхідно уникати розташування маршрутизатора поруч із джерелами електромагнітного шуму, такими як високовольтні лінії чи електроприлади, оскільки це може вплинути на роботу мережі.
- Доступ до електропостачання: Вибране місце мусить мати доступ до стабільного джерела електропостачання для живлення маршрутизатора.

Врахування цих факторів допоможе забезпечити оптимальну продуктивність мережі LoRa та забезпечить максимальне охоплення на вибраній території. Також, хочу зазаначити, що пристрої можуть бути підключеними відразу до декількох маршрутизаторів, але передавати сигнал будуть за найстабільнішим зв’язком.

TTI - Інфраструктура яка забезпечує швидку і безпечну передачу данних між маршрутизатором, підключеним до мережі і мікросервісом по збору і обробки даних. Передача данних на мікросервіс здійснюється за допомогою стріму MQTT, на який підписан мікросервіс.

Storage Microservice - мікросервіс, створений для слухання івентів, отриманих за допомогою MQTT, який оброблює вхідні данні котрі були отримані з датчиків пристрою, перетворює їх на читабельні записи і зберігає їх в базі даних.

MongoDB - це документоорієнтована система керування базами даних (NoSQL), яка зберігає дані у форматі BSON (Binary JSON). Вона розроблена компанією MongoDB, Inc. і стала однією з найпопулярніших баз даних NoSQL, особливо серед розробників веб-додатків та інших проектів, де важливо масштабовувати та обробляти великі обсяги даних. Кожен документ - це пара "ключ-значення", подібно до об'єкта JSON. MongoDB розроблена для легкої горизонтальної масштабованості, що дозволяє додавати нові сервери для розподіленого зберігання та обробки даних. Відсутність фіксованої схеми дозволяє додавати поля в документи без необхідності зміни всієї бази даних. MongoDB підтримує мову запитань, подібну до SQL, а також може використовувати мови програмування для взаємодії з базою даних. Індексація та різні методи оптимізації запитів дозволяють отримати швидкий доступ до даних. Можливість використання агрегаційних фреймів для обробки та аналізу даних у

багатоетапних процесах дозволяє побудувати максимально читабельний і оптимізований запит для отримання даних комбінований лише в один масив фреймів. Також MongoDB має велику та активну спільноту розробників, що сприяє обміну знаннями та розвитку технології.

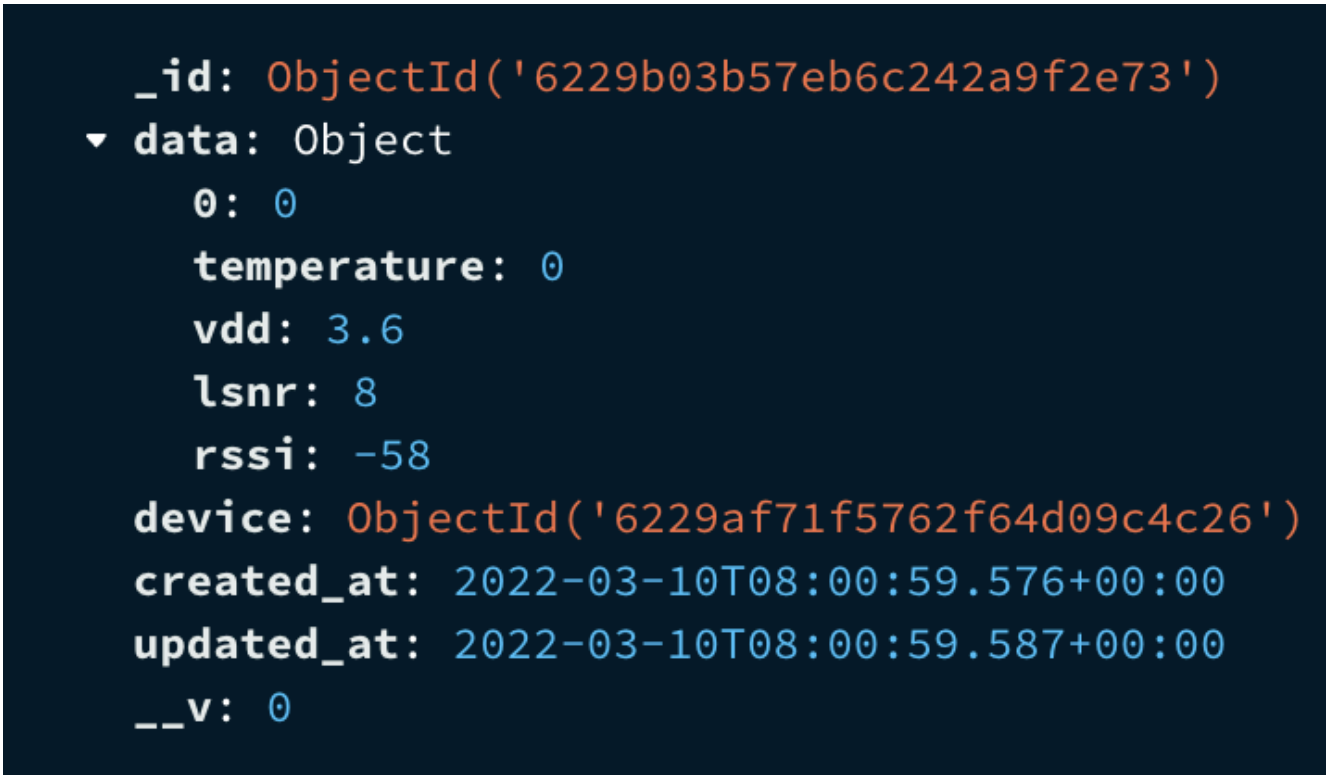

Рисунок 2.5 - Приклад запису в MongoDB

REST API - це стандартний спосіб взаємодії між програмними системами, який базується на принципах архітектури REST. REST є стандартом для розробки веб-служб та взаємодії між компонентами систем, основаними на HTTP. У REST API, дані та функціональність представлені як ресурси, кожен з яких має унікальний ідентифікатор, називаний URI (Uniform Resource Identifier). Ресурси можуть бути конкретними об'єктами або послугами, які можуть бути представлені та оброблені. REST використовує стандартні HTTP методи для виконання різних операцій над ресурсами:

- **GET**: Отримання даних.
- **POST**: Створення нового ресурсу або відправка даних на обробку.
- **PUT**: Оновлення існуючого ресурсу або створення нового, якщо він не існує.

- **DELETE**: Видалення ресурсу.

Дані, які передаються та отримуються через REST API, часто подаються у форматі JSON (JavaScript Object Notation) або XML (eXtensible Markup Language), що робить їх легкими для читання та розуміння. Кожен запит до сервера вважається самостійним, тобто сервер не зберігає інформацію про стан клієнта між запитами. REST API може використовувати заходи безпеки, такі як HTTPS (HTTP Secure) для захисту конфіденційності та цілісності даних. REST API використовує стандартні HTTP коди стану для повідомлення про результат виконання запиту (наприклад, 200 OK, 404 Not Found, 500 Internal Server Error). REST API широко використовується у веб-розробці для реалізації взаємодії між клієнтськими додатками та серверами. Він дозволяє розробникам створювати розділені та ефективні системи, а також легко взаємодіяти з різними сервісами та ресурсами в Інтернеті.

Client - програмного забезпечення якие взаємодіє з веб-серверами і отримує веб-ресурси (такі як HTML-сторінки, зображення, файли CSS, скрипти, тощо) за допомогою протоколів взаємодії в Інтернеті, зокрема HTTP (Hypertext Transfer Protocol) або HTTPS (Hypertext Transfer Protocol Secure). Використовується для відображення інформації отриманих з датчиків користувачам, котрі мають доступ до перегляду того чи іншого контенту.

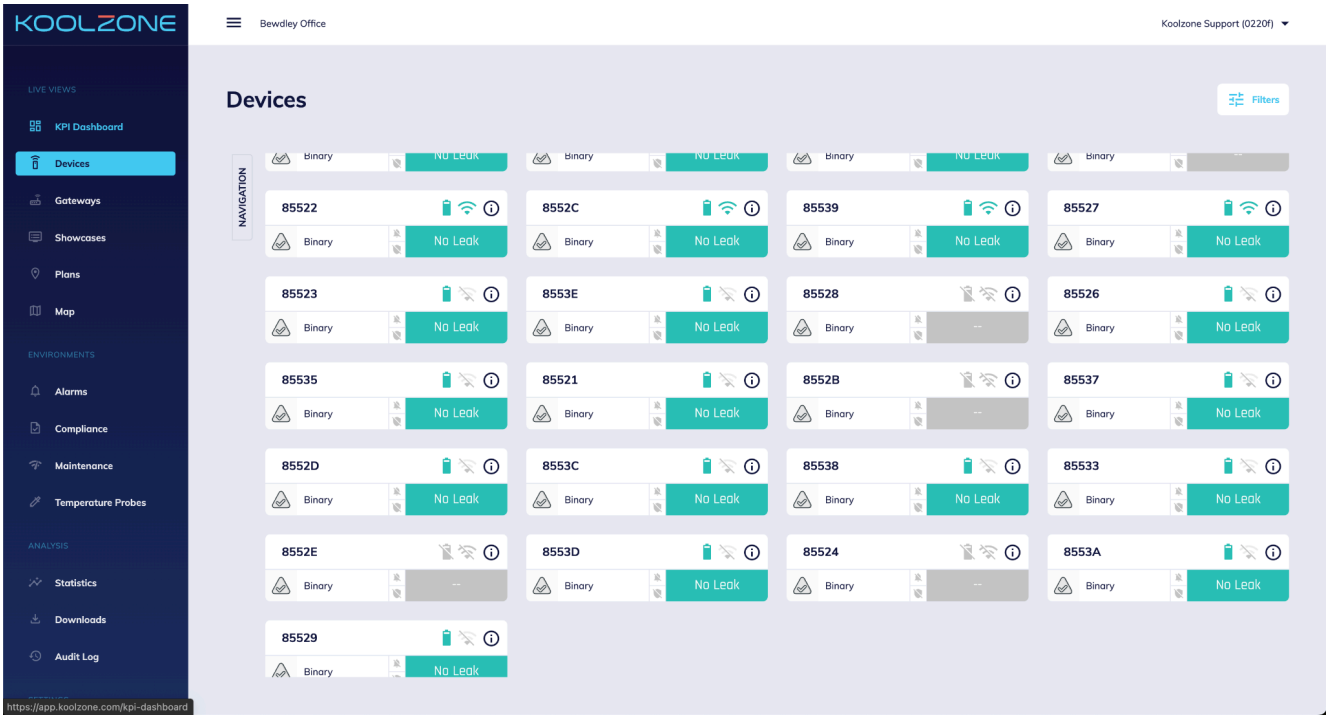

Рисунок 2.6 - Приклад клієнту який виводить стан пристроїв

2.2.1 MQTT

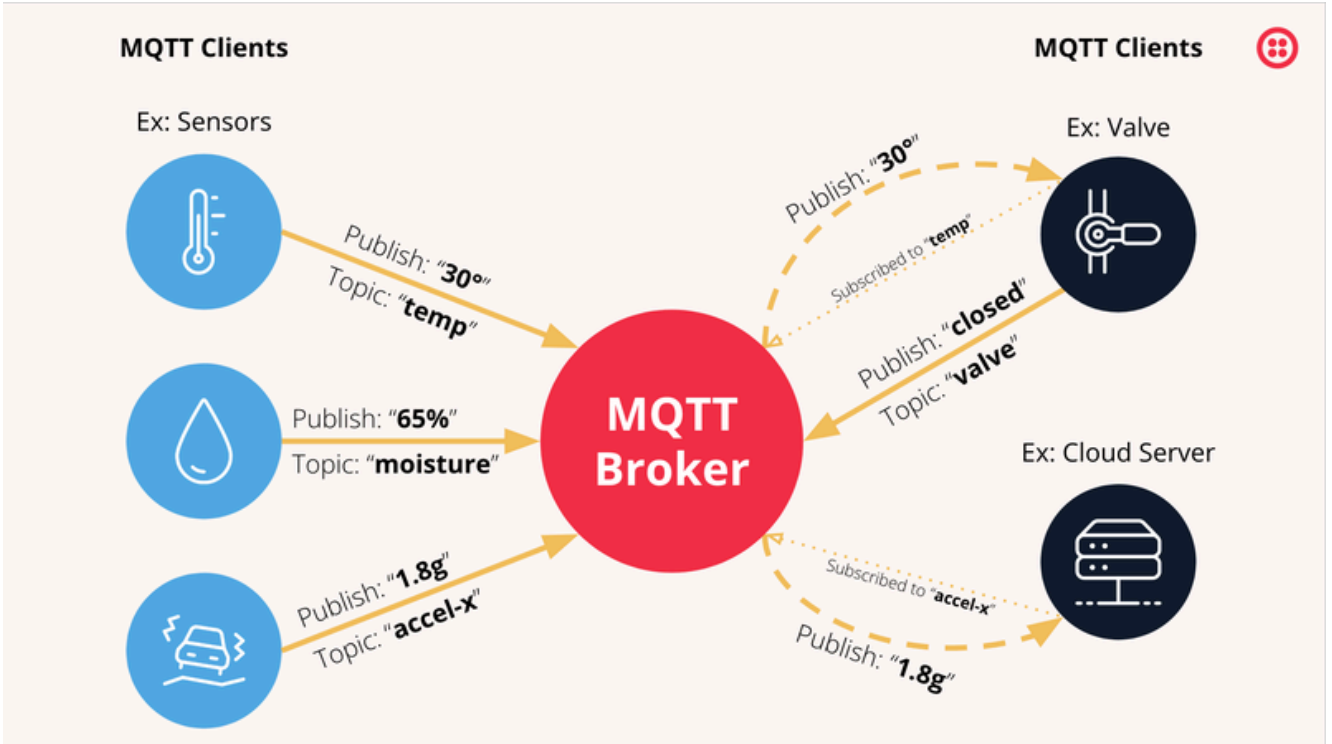

Рисунок 2.7 - Приклад роботи MQTT Broker

MQTT (Message Queuing Telemetry Transport) - це легкий та ефективний протокол передачі повідомлень, який розроблений для взаємодії між пристроями в обміні даними в реальному часі. Вперше стандартизований у 1999 році, MQTT став широко використовуваним в Інтернеті речей (IoT) та в інших контекстах, де важливо забезпечити низьку затримку та ефективність мережі.

Основні особливості та характеристики MQTT:

- Легкий та ефективний: Протокол спроектований для мінімізації навантаження на мережу та забезпечення ефективності обміну повідомленнями.
- Публікація/Підписка (Publish/Subscribe): Взаємодія заснована на моделі "публікація/підписка", де пристрої можуть публікувати повідомлення в "теми" та підписуватися на отримання повідомлень з конкретних тем.
- Теми (Topics): Повідомлення розсилаються за конкретними темами, що дозволяє гнучко організовувати та керувати обміном даними.
- QoS (Quality of Service): MQTT підтримує три рівні QoS, що дозволяє визначити гарантії доставки повідомлень: QoS 0 (по одному разу), QoS 1 (підтвердження доставки) і QoS 2 (гарантована доставка).
- Зберігання останнього повідомлення (Retained messages): Система може зберігати останнє повідомлення для кожної теми, яке буде надіслане новим підписникам після їхнього підписання.
- Сприяє масштабованості та низькій латентності: Через свою легкість та ефективність, MQTT чудово підходить для розгортання великих мереж IoT пристроїв з низькою латентністю.

MQTT Broker - це сервер або програмне забезпечення, яке відповідає за приймання, обробку та маршрутизацію повідомлень в мережі MQTT. Брокер MQTT грає ключову роль у забезпеченні зв'язку між пристроями (клієнтами) в мережі MQTT, дозволяючи їм обмінюватися повідомленнями за принципом публікації/підписки.

2.2.2 Express.js

Express.js, або просто Express, — це базова основа веб-додатків для створення RESTful API за допомогою Node.js, випущена як безкоштовне програмне забезпечення з відкритим вихідним кодом за ліцензією MIT. Він призначений для створення веб-додатків і API. Його називають де-факто стандартною серверною структурою для Node.js.

Автор оригіналу TJ Holowaychuk описав його як сервер, натхненний Сінатрою, що означає, що він відносно мінімальний із багатьма функціями, доступними у вигляді плагінів. Express — це внутрішній компонент популярних стеків розробки, як-от стек MEAN, MERN або MEVN, разом із програмним забезпеченням бази даних MongoDB і зовнішнім фреймворком або бібліотекою JavaScript. Ось деякі аспекти Express.js, які важливо враховувати:

- Маршрутизація (Routing): Express дозволяє визначити різні маршрути для HTTP-запитів, такі як GET, POST, PUT, DELETE тощо. Маршрутизація використовується для визначення, як додаток повинен відповідати на конкретні запити.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.post('/api/users', (req, res) => {
  // Обробка POST-запиту для створення нового користувача
});

// Інші маршрути...

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Рисунок 2.8 - маршрутизація у фреймворку Express.js

- Middleware: Express використовує концепцію middleware для обробки HTTP-запитів. Middleware - це функції, які мають доступ до об'єкта запиту (req), об'єкта відповіді (res), та наступної middleware в потоці. Вони можуть виконувати різні завдання, такі як логування, перевірка автентифікації, обробка помилок та інше.
- Шаблонізатори (Template Engines): Express може використовувати шаблонізатори для генерації HTML на стороні сервера. Наприклад, ejs, pug та інші.
- Статичні файли (Static Files): Express надає можливість служити статичні файли (зображення, CSS, JavaScript) безпосередньо з папки.

```
// Сервірування статичних файлів з папки "public"
app.use(express.static('public'));
```

Рисунок 2.9 - Сервірування статичних файлів

2.2.3 Highcharts

Highcharts - це JavaScript бібліотека для створення інтерактивних та красивих графіків та діаграм на стороні клієнта. Ця бібліотека надає розширений набір можливостей для візуалізації даних у веб-додатках та на веб-сайтах. Highcharts підтримує різні типи графіків, такі як лінійні графіки, стовпчасті графіки, кругові діаграми, area-графіки, графіки розсіювання, сплайн-графіки та багато інших. Це дозволяє вибрати оптимальний тип графіка для конкретного набору даних.

Також дана бібліотека надає можливості для взаємодії з графіками, такі як масштабування, перетягування, відображення деталей при наведенні, вибір точок та багато іншого. Ви маєте повний контроль над виглядом та стилізацією графіків, можливість налаштувати кольори, шрифти, лінії, маркери та інше. Графіки Highcharts можуть бути анімованими, що додає ефективності та привабливості при зміні даних або взаємодії з користувачем. Що не мало вожливо готові діаграми або графіки легко експортувати у різні формати (PNG, JPEG, PDF або SVG), що також надає можливість друку графіків.

Графіки, створені за допомогою Highcharts, добре працюють на різних пристроях, включаючи мобільні та планшетні. Highcharts може бути легко інтегрований з різними веб-технологіями, такими як HTML, JavaScript, Angular, React, та інші.

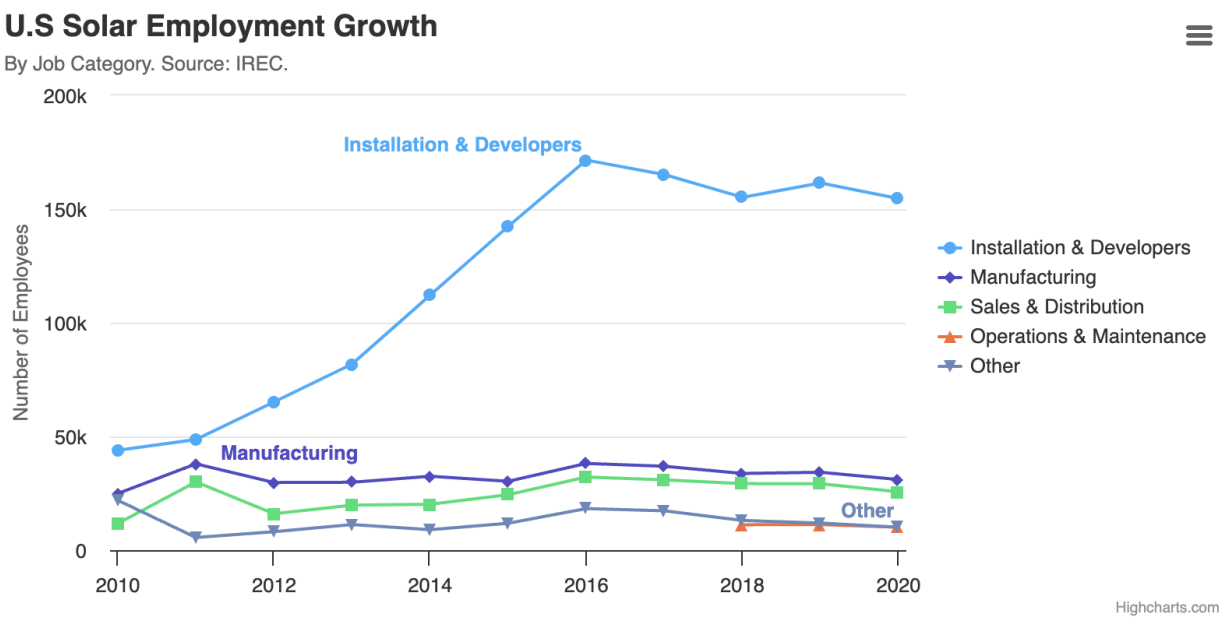

Рисунок 2.10 - Лінійний графік

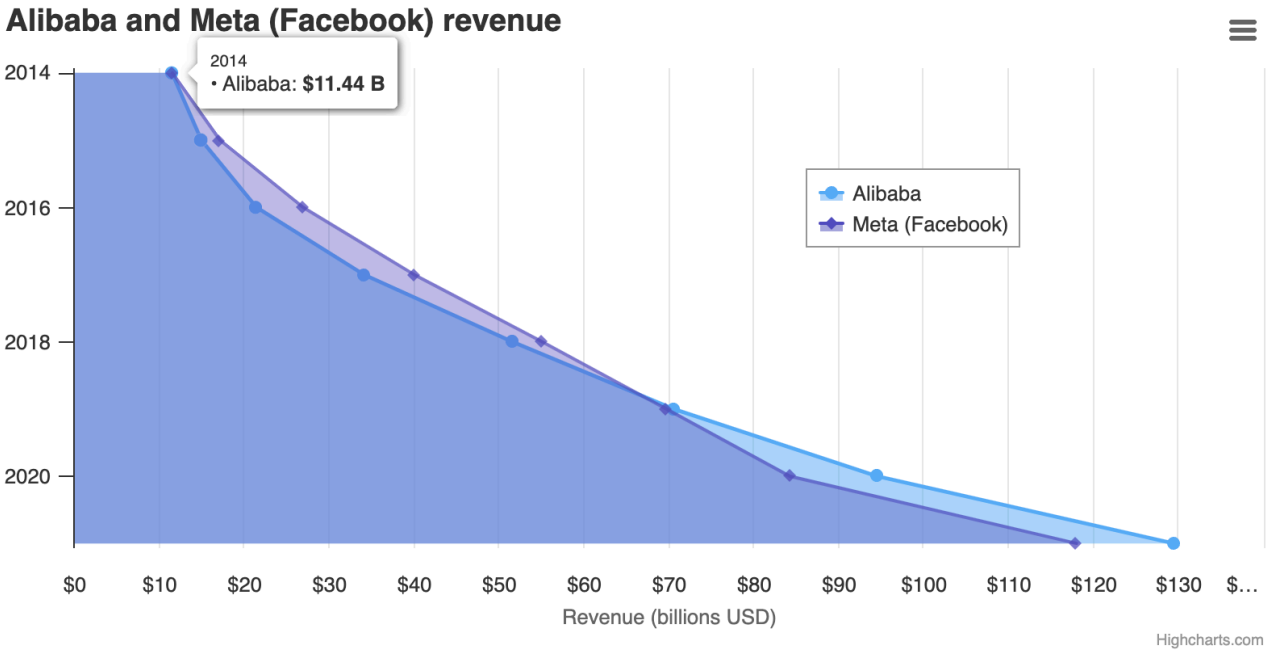

Рисунок 2.11 - Зональний графік

3 РЕАЛІЗАЦІЯ ОТРИМАННЯ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ

Як вже зазначалося я буду використовувати пристрої фірми Elsys котрі вже були налаштовані і додані до сервісу TTI для їх подальшого використання в системі.

Наступним етапом буде створення бази даних MongoDB і опис моделей записів, для їх подальшого використання в системі. Для цього першим чином ми зайдемо на MongoAtlas - сервіс для створення баз даних MongoDB (Рис. 3.1). Даний сервіс дозволяє нам створити безкоштовну версію бази даних з обмеженнями, але цього вистачить, аби продемонструвати потенціал і можливості данної системи. Для своєї БД я використав AWS в якості хмарного провайдера і вибрав Франкфур як регіон розташування вже вибраного провайдера. Наступним кроком був вибір типу кластера. Мій вибір впав на безкоштовний варінт з об’ємом пам’яті 512MB. Далі я вибрав 6ту версію MongoDB, так як вона була найостаннішою на момент створення хмарного сховища. Новостворену базу даних вже можна використовувати для зберігання записів. Для взаємодії програми з MongoDB існує два основних методи: MongoDB CLient та Mongoose. Оскільки Mongoose є більш легким у розумінні і має розширений функціонал, то в данному проекті біду використовуватись саме він. Для цього нам потрібно створити репозиторій для зберігання та оновлення моделей Mongoose та для більш легкого імпорту їх в різні частини данної системи. Для створення репозиторію я використовую BitBucket (Рис. 3.2). Він дозволяє легко керувати файлами, комітами і мерджити гілки. Після створення репозиторію, якій міститиме моделі ми відкриваємо термінал і вставляємо скопійований URL репозиторію для того, щоб підтягнути його на наш ПК. Після скачування я відкриваю даний проект і за допомогою “*npm init*” ініціалізую проект на мові Node.js. Наступним моїм кроком буде встановлення залежності “mongoose” за допомогою команди “npm і mongoose” котра буде використовуватися для опису моделей БД, їх залежностей і взаємодії (Рис. 3.3).

Рисунок 3.1 - Створення нової бази даних

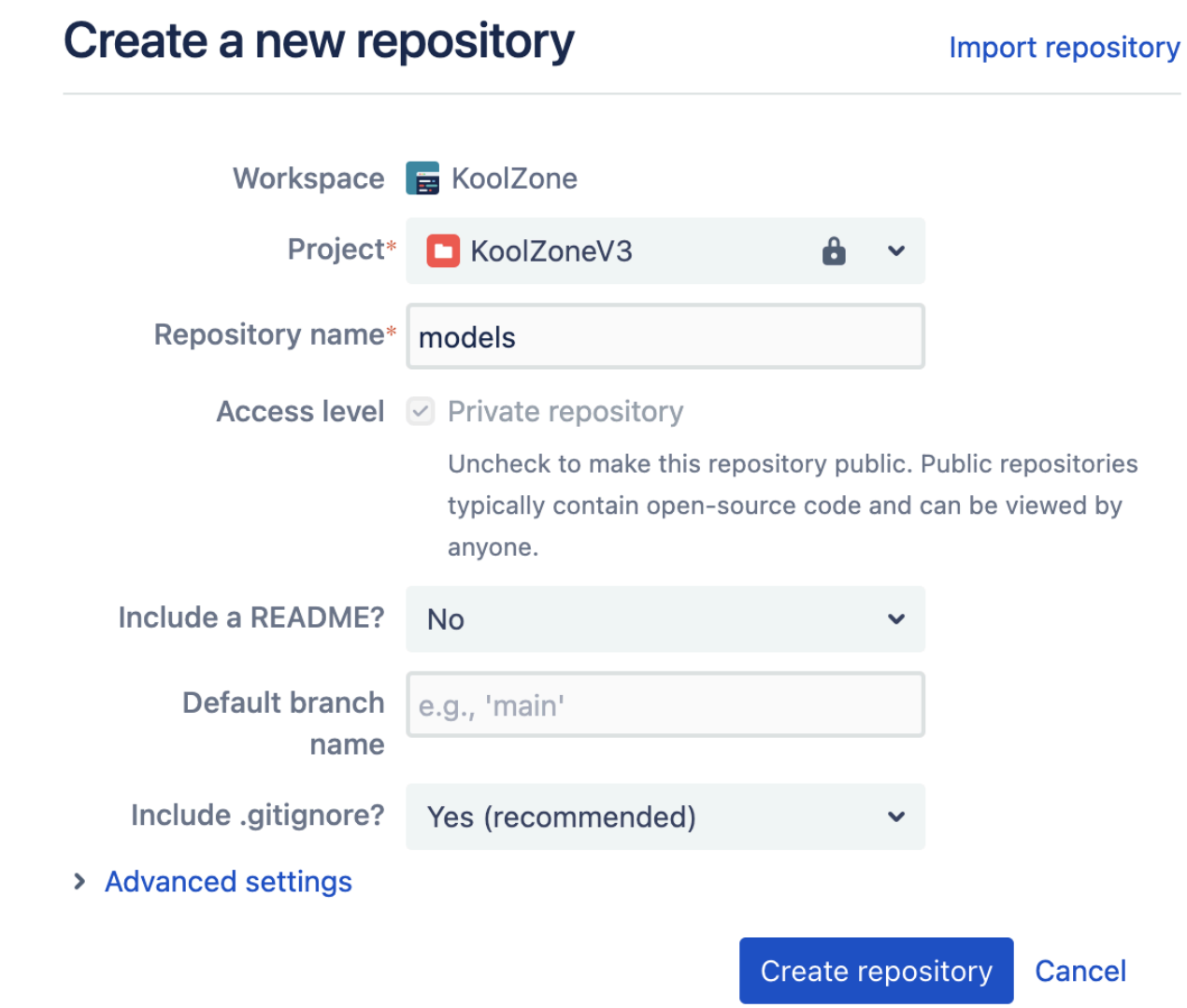

Рисунок 3.2 - Створення репозиторію в BitBucket

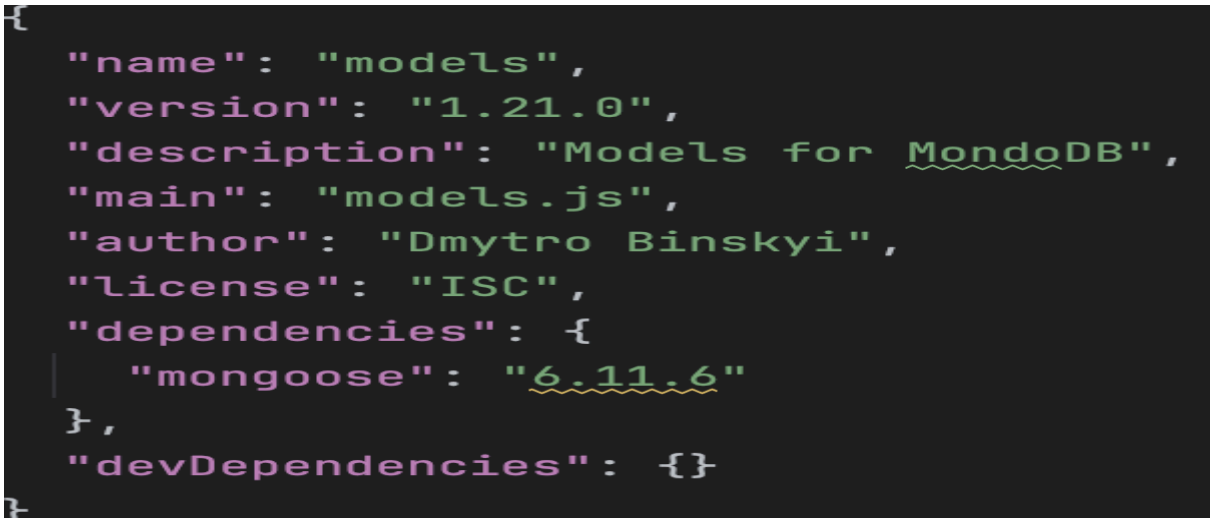

Рисунок 3.3 - Список залежностей в репозиторію models

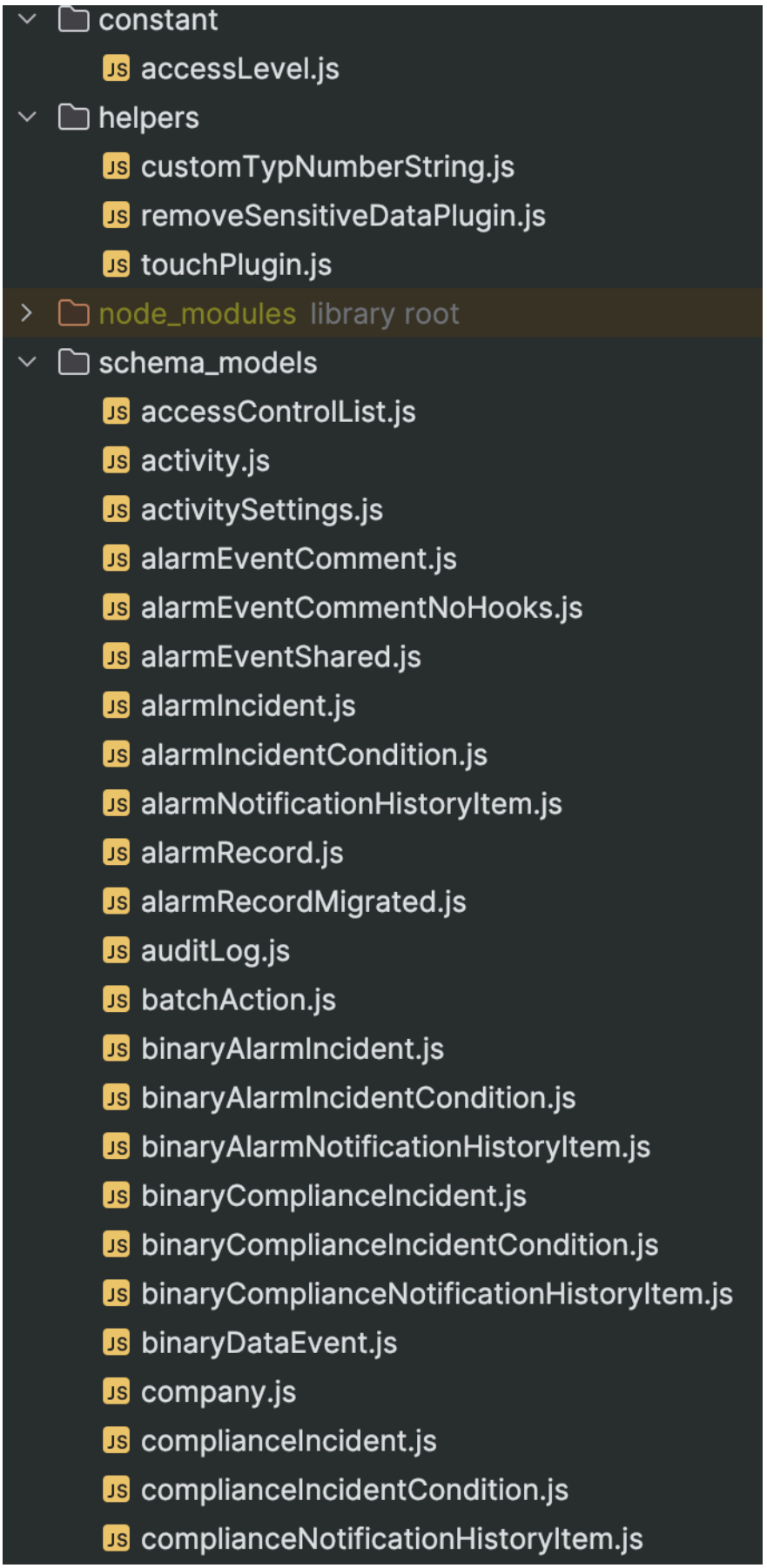

Рисунок 3.4 - Ієрархія файлів репозиторію models

На Рис. 3.4 відображено ієрархію файлів та їх моделей створених для їх підключення до БД з метою автоматичної конфігурації, проставлення індексів та калькульованих параметрів. Наступним кроком буде підключення і ініціалізація всіх моделей в основний файл “main.js” (Рис. 3.3). Даний файл містить такий код, що дозволяє нам експортувати моделі в інші мікросервіси нашої системи:

```
let mongoose = require('mongoose');
const MongoClient = require('mongodb').MongoClient;
mongoose.Promise = global.Promise;

require('./helpers/customTypNumberString');

const DataSchema = require('./schema_models/data');
const DataPerCollectionSchema = require('./schema_models/dataPerCollection');
const SensorSchema = require('./schema_models/sensor');
const GatewaySchema = require('./schema_models/gateway');
const UserSchema = require('./schema_models/user');
const LocationSchema = require('./schema_models/location');
const SiteSchema = require('./schema_models/site');
const CompanySchema = require('./schema_models/company');
const NotificationSettingsSchema = require('./schema_models/notificationSetting');
const RemarkSchema = require('./schema_models/remark');
const HashedUrlSchema = require('./schema_models/hashedUrl');
const AuditLogSchema = require('./schema_models/auditLog');
const UserAuditLogSchema = require('./schema_models/userAuditLog');
const PreferencesSchema = require('./schema_models/preferences');
const SensorStatsSchema = require('./schema_models/sensorStats');
const SensorStatsArchiveSchema = require('./schema_models/sensorStatsArchive');
const ACLSchema = require('./schema_models/accessControlList');
```

```
    const AlarmEventCommentSchema =
require('./schema_models/alarmEventComment');
    const AlarmEventCommentSchemaNoHooks =
require('./schema_models/alarmEventCommentNoHooks');
    const AlarmEventSharedSchema =
require('./schema_models/alarmEventShared');
    const SensorReadingSchema = require('./schema_models/sensorReading');
    const AlarmRecordSchema = require('./schema_models/alarmRecord');
    const AlarmRecordMigratedSchema =
require('./schema_models/alarmRecordMigrated');
    const PlanSchema = require('./schema_models/plan');
    const ShowcaseSchema = require('./schema_models/showcase');
    const ActivitySettingsSchema = require('./schema_models/activitySettings');
    const ActivitySchema = require('./schema_models/activity');
    const ShowcaseImageSchema = require('./schema_models/showcaseImage');
    const SensorTProbeSchema = require('./schema_models/sensorTProbe');
    const SensorTProbeDataSchema = require('./schema_models/sensorTProbeData');
    const DataAnalysisResultSchema =
require('./schema_models/dataAnalysisResult');
    const LanguageSchema = require('./schema_models/language');
    const TranslationSchema = require('./schema_models/translation');
    const TranslationPageSchema = require('./schema_models/translationPage');
    const GatewayWithDeviceSchema =
require('./schema_models/gatewayWithDevice');
    const PresetTextSchema = require('./schema_models/presetText');
    const ScheduledReportSettingsSchema =
require('./schema_models/scheduledReportSettings');
    const SensorItemSchema = require('./schema_models/sensorItem');
    const DeviceSchema = require('./schema_models/device');
    const DeviceToGroupSchema = require('./schema_models/deviceToGroup');
```

```
const DeviceEventsSchema = require('./schema_models/deviceEvents');
const DataPerCollectionKzHistorySchema = require('./schema_models/dataPerCollectionKzHistory');
const ProfileSchema = require('./schema_models/profiles');
const TemperatureProbeItemSchema = require('./schema_models/temperatureProbeItem');
const TemperatureProbeMeasurementSchema = require('./schema_models/temperatureProbeMeasurement');
const BinaryDataEventSchema = require('./schema_models/binaryDataEvent');

const NewsSchema = require('./schema_models/news');
const NewsContentSchema = require('./schema_models/newsContent');
const ReleaseNoteSchema = require('./schema_models/releaseNote');
const ReleaseNoteContentSchema = require('./schema_models/releaseNoteContent');
const ViewedNewsSchema = require('./schema_models/viewedNews');
const ComplianceIncidentConditionSchema = require('./schema_models/complianceIncidentCondition');
const ComplianceIncidentSchema = require('./schema_models/complianceIncident');
const BinaryComplianceIncidentSchema = require('./schema_models/binaryComplianceIncident');
const BinaryComplianceIncidentConditionSchema = require('./schema_models/binaryComplianceIncidentCondition');
const ReportResultsSchema = require('./schema_models/reportResults');
const ReportActionsSchema = require('./schema_models/reportActions');
const BatchActionSchema = require('./schema_models/batchAction');
const NodataIncidentSchema = require('./schema_models/nodataIncident');
const GatewayIncidentSchema = require('./schema_models/gatewayIncident');
```

```
    const AlarmIncidentConditionSchema = require('./schema_models/alarmIncidentCondition');
    const AlarmIncidentSchema = require('./schema_models/alarmIncident');
    const BinaryAlarmIncidentSchema = require('./schema_models/binaryAlarmIncident');
    const BinaryAlarmIncidentConditionSchema = require('./schema_models/binaryAlarmIncidentCondition');
    const NotificationCountItemSchema = require('./schema_models/notificationCountItem');
    const NotificationItemSchema = require('./schema_models/notificationItem');

    const AlarmNotificationHistoryItemSchema = require('./schema_models/alarmNotificationHistoryItem');
    const BinaryAlarmNotificationHistoryItemSchema = require('./schema_models/binaryAlarmNotificationHistoryItem');
    const ComplianceNotificationHistoryItemSchema = require('./schema_models/complianceNotificationHistoryItem');
    const BinaryComplianceNotificationHistoryItemSchema = require('./schema_models/binaryComplianceNotificationHistoryItem');
    const GatewayNotificationHistoryItemSchema = require('./schema_models/gatewayNotificationHistoryItem');
    const NodataNotificationHistoryItemSchema = require('./schema_models/nodataNotificationHistoryItem');

    // mongoose.set('debug', true);

    let initialized = false;

    const defaultConnectionOptions = {
      serverSelectionTimeoutMS: 10000, // default 30 seconds was 5000
```

```
  socketTimeoutMS: 25000, // 0
  maxPoolSize: 10,
  // bufferMaxEntries: 0,
  // bufferCommands: false
  // poolSize: 5 /default
};

mongoose.set('strictQuery', false); // by default will be true
const models = {
  mongoose,
  MongoClient,
  GatewayWithDeviceSchema,
  //Schemas
  DataSchema,
  DeviceToGroupSchema,
  SensorSchema,
  GatewaySchema,
  UserSchema,
  LocationSchema,
  SiteSchema,
  CompanySchema,
  NotificationSettingsSchema,
  RemarkSchema,
  HashedUrlSchema,
  AuditLogSchema,
  UserAuditLogSchema,
  PreferencesSchema,
  SensorStatsSchema,
  SensorStatsArchiveSchema,
  AlarmEventCommentSchema,
```

AlarmEventSharedSchema,

ACLSchema,

SensorReadingSchema,

AlarmRecordSchema,

AlarmRecordMigratedSchema,

PlanSchema,

ShowcaseSchema,

ActivitySchema,

ActivitySettingsSchema,

ShowcaseImageSchema,

SensorTProbeSchema,

SensorTProbeDataSchema,

DataAnalysisResultSchema,

LanguageSchema,

TranslationSchema,

TranslationPageSchema,

PresetTextSchema,

ScheduledReportSettingsSchema,

AlarmEventCommentSchemaNoHooks,

ProfileSchema,

NewsSchema,

NewsContentSchema,

ReleaseNoteSchema,

ReleaseNoteContentSchema,

ViewedNewsSchema,

ComplianceIncidentConditionSchema,

ComplianceIncidentSchema,

BinaryComplianceIncidentSchema,

BinaryComplianceIncidentConditionSchema,

ReportResultsSchema,

```
ReportActionsSchema,
BatchActionSchema,
BinaryDataEventSchema,
SensorItemSchema,
DeviceSchema,
DeviceEventsSchema,
NodataIncidentSchema,
GatewayIncidentSchema,
AlarmIncidentSchema,
AlarmIncidentConditionSchema,
BinaryAlarmIncidentSchema,
BinaryAlarmIncidentConditionSchema,
NotificationCountItemSchema,
NotificationItemSchema,

AlarmNotificationHistoryItemSchema,
BinaryAlarmNotificationHistoryItemSchema,
ComplianceNotificationHistoryItemSchema,
BinaryComplianceNotificationHistoryItemSchema,
GatewayNotificationHistoryItemSchema,
NodataNotificationHistoryItemSchema,

getModels() {
  if (initialized) {
    return this;
  }
  throw new Error('Models not initialized run init method first');
},
// MODELS
/**
```

```
     *
     * @param mongoUrl
     * @param options
     * @param dbUrlForSensor
     * @returns {*}
     */
    initiateMongooseModels(mongoUrl, options = {

                                        }, dbUrlForSensor =
process.env.KLZN_SERVER_MONGODB_SENSOR_DATA) {
     if (initialized) {
      return this;
     }
     options = options || {};
     let deviceDataDbName = options.deviceDataDbName;
     delete options.deviceDataDbName;
     let mainDbName = options.mainDbName;
     delete options.mainDbName;
     let deviceDataDbNameObj = {};
     let mainDbNameObj = {};
     if (deviceDataDbName) {
        deviceDataDbNameObj.dbName = deviceDataDbName;
     }
     if (mainDbName) {
        mainDbNameObj.dbName = mainDbName;
     }
     const NODE_ENV = (process.env.NODE_ENV || '').toLowerCase();
     if (NODE_ENV === 'production' || NODE_ENV === 'prod') {
      deviceDataDbNameObj = {};
      mainDbNameObj = {};
```

```
}
if (NODE_ENV === 'uat') {
  deviceDataDbNameObj.dbName = 'koolzone-devices-data';
  mainDbNameObj.dbName = 'koolzone-db';
}

options = {...defaultConnectionOptions, ...options};

                const sensorDataConnection = dbUrlForSensor && mongoose.createConnection(dbUrlForSensor, {...options, ...deviceDataDbNameObj});

const DataModel = mongoose.model('Data', DataSchema);
const SensorModel = mongoose.model('Sensor', SensorSchema);
const GatewayModel = mongoose.model('Gateway', GatewaySchema);
const UserModel = mongoose.model('User', UserSchema);
const LocationModel = mongoose.model('Location', LocationSchema);
const SiteModel = mongoose.model('Site', SiteSchema);
const CompanyModel = mongoose.model('Company', CompanySchema);
   const NotificationSettingsModel = mongoose.model('NotificationSettings', NotificationSettingsSchema);
const RemarkModel = mongoose.model('Remark', RemarkSchema);
const HashedUrlModel = mongoose.model('HashedUrl', HashedUrlSchema);
const AuditLogModel = mongoose.model('AuditLog', AuditLogSchema);
        const UserAuditLogModel = mongoose.model('UserAuditLog', UserAuditLogSchema);
const PreferencesModel = mongoose.model('Preferences', PreferencesSchema);
const SensorStatsModel = mongoose.model('SensorStats', SensorStatsSchema);
   const SensorStatsArchiveModel = mongoose.model('SensorStatsArchive', SensorStatsArchiveSchema);
```

```
const AlarmEventCommentModel = mongoose.model('AlarmEventComment', AlarmEventCommentSchema);
const AlarmEventCommentNoHooksModel = mongoose.model('AlarmEventCommentNoHooks', AlarmEventCommentSchemaNoHooks);
const AlarmEventSharedModel = mongoose.model('AlarmEventShared', AlarmEventSharedSchema);
const ACLModel = mongoose.model('ACL', ACLSchema);
const SensorReadingModel = mongoose.model('SensorReading', SensorReadingSchema);
const AlarmRecordModel = mongoose.model('AlarmRecord', AlarmRecordSchema);
const AlarmRecordMigratedModel = mongoose.model('AlarmRecordMigrated', AlarmRecordMigratedSchema);
const PlanModel = mongoose.model('Plan', PlanSchema);
const ShowcaseModel = mongoose.model('Showcase', ShowcaseSchema);
const ActivitySettingsModel = mongoose.model('ActivitySettings', ActivitySettingsSchema);
const ActivityModel = mongoose.model('Activity', ActivitySchema);
const ShowcaseImageModel = mongoose.model('ShowcaseImage', ShowcaseImageSchema);
const SensorTProbeModel = mongoose.model('SensorTProbe', SensorTProbeSchema);
const SensorTProbeDataModel = mongoose.model('SensorTProbeData', SensorTProbeDataSchema);
const DataAnalysisResultModel = mongoose.model('DataAnalysisResult', DataAnalysisResultSchema);
const LanguageModel = mongoose.model('Language', LanguageSchema);
const TranslationModel = mongoose.model('Translation', TranslationSchema);
```

```
const TranslationPageModel = mongoose.model('TranslationPage', TranslationPageSchema);
const GatewayWithDeviceModel = mongoose.model('GatewayWithDevice', GatewayWithDeviceSchema);
const PresetTextModel = mongoose.model('PresetText', PresetTextSchema);
const ScheduledReportSettingsModel = mongoose.model('ScheduledReportSettings', ScheduledReportSettingsSchema);
const SensorItemModel = mongoose.model('SensorItem', SensorItemSchema);
const DeviceModel = mongoose.model('Device', DeviceSchema);
const DeviceToGroup = mongoose.model('DeviceToGroup', DeviceToGroupSchema);
const DeviceEvents = mongoose.model('DeviceEvent', DeviceEventsSchema);
const ProfileModel = mongoose.model('Profile', ProfileSchema);
const TemperatureProbeItemModel = mongoose.model('TemperatureProbeItem', TemperatureProbeItemSchema);
const TemperatureProbeMeasurementModel = mongoose.model('TemperatureProbeMeasurement', TemperatureProbeMeasurementSchema);
const BinaryDataEventModel = mongoose.model('BinaryDataEvent', BinaryDataEventSchema);

const NewsModel = mongoose.model('News', NewsSchema);
const NewsContentModel = mongoose.model('NewsContent', NewsContentSchema);
const ReleaseNoteModel = mongoose.model('ReleaseNote', ReleaseNoteSchema);
const ReleaseNoteContentModel = mongoose.model('ReleaseNoteContent', ReleaseNoteContentSchema);
const ViewedNewsModel = mongoose.model('ViewedNews', ViewedNewsSchema);
```

```
const ComplianceIncidentConditionModel = mongoose.model('ComplianceIncidentCondition', ComplianceIncidentConditionSchema);
const ComplianceIncidentModel = mongoose.model('ComplianceIncident', ComplianceIncidentSchema);
const BinaryComplianceIncidentModel = mongoose.model('BinaryComplianceIncident', BinaryComplianceIncidentSchema);
const BinaryComplianceIncidentConditionModel = mongoose.model('BinaryComplianceIncidentCondition', BinaryComplianceIncidentConditionSchema);

const ReportResultsModel = mongoose.model('ReportResult', ReportResultsSchema);
const ReportActionsModel = mongoose.model('ReportAction', ReportActionsSchema);
const BatchActionModel = mongoose.model('BatchAction', BatchActionSchema);
const NodataIncidentModel = mongoose.model('NodataIncident', NodataIncidentSchema);
const GatewayIncidentModel = mongoose.model('GatewayIncident', GatewayIncidentSchema);
const AlarmIncidentConditionModel = mongoose.model('AlarmIncidentCondition', AlarmIncidentConditionSchema);
const AlarmIncidentModel = mongoose.model('AlarmIncident', AlarmIncidentSchema);
const BinaryAlarmIncidentModel = mongoose.model('BinaryAlarmIncident', BinaryAlarmIncidentSchema);
const BinaryAlarmIncidentConditionModel = mongoose.model('BinaryAlarmIncidentCondition', BinaryAlarmIncidentConditionSchema);
```

```
const NotificationCountItemModel = mongoose.model('NotificationCountItem', NotificationCountItemSchema);
const NotificationItemModel = mongoose.model('NotificationItem', NotificationItemSchema);

const AlarmNotificationHistoryItemModel = mongoose.model('AlarmNotificationHistoryItem', AlarmNotificationHistoryItemSchema);
const ComplianceNotificationHistoryItemModel = mongoose.model('ComplianceNotificationHistoryItem', ComplianceNotificationHistoryItemSchema);
const BinaryAlarmNotificationHistoryItemModel = mongoose.model('BinaryAlarmNotificationHistoryItem', BinaryAlarmNotificationHistoryItemSchema);
const BinaryComplianceNotificationHistoryItemModel = mongoose.model('BinaryComplianceNotificationHistoryItem', BinaryComplianceNotificationHistoryItemSchema);
const GatewayNotificationHistoryItemModel = mongoose.model('GatewayNotificationHistoryItem', GatewayNotificationHistoryItemSchema);
const NodataNotificationHistoryItemModel = mongoose.model('NodataNotificationHistoryItem', NodataNotificationHistoryItemSchema);

    // AlarmRecordModel.on('index', (...args) => {
    //   console.log(args);
    //   console.log('index createdin alarm record');
    // });
```

```
  mongoose.connect(mongoUrl, {...options, ...mainDbNameObj}).then(succ =>
{
  console.log('Connected' );
  process.send && process.send('ready'); // wait ready option pm2
  // indexRemoval(LocationModel)(['name_1_company_1']);
 });

 mongoose.connection.once('close', function() {
  console.log('EVENT --CLOSE-- closing connection to DB');
 });

 mongoose.connection.once('error', function (error) {
  console.error('MONGOERROR!', error);
  console.log('CLOSING CONNECTION');
  mongoose.connection.close();
  sensorDataConnection && sensorDataConnection.close();
  setTimeout(() => {
   console.log('PROCESS EXIT in 3 sec');
   process.exit(1);
  }, 3000);
 });

 Object.assign(this,
  {
   mongoose,
   //Models
   GatewayWithDeviceModel,
   DataModel,
   SensorModel,
   GatewayModel,
```

UserModel,

BinaryDataEventModel,

LocationModel,

AlarmRecordModel,

AlarmRecordMigratedModel,

SiteModel,

CompanyModel,

NotificationSettingsModel,

RemarkModel,

HashedUrlModel,

AuditLogModel,

UserAuditLogModel,

AlarmEventCommentModel,

AlarmEventCommentNoHooksModel,

AlarmEventSharedModel,

ACLModel,

PreferencesModel,

SensorStatsModel,

SensorStatsArchiveModel,

SensorReadingModel,

PlanModel,

ShowcaseModel,

ActivityModel,

ActivitySettingsModel,

ShowcaseImageModel,

SensorTProbeModel,

SensorTProbeDataModel,

DataAnalysisResultModel,

LanguageModel,

TranslationModel,

TranslationPageModel,

PresetTextModel,

ScheduledReportSettingsModel,

SensorItemModel,

DeviceModel,

DeviceToGroup,

DeviceEvents,

ProfileModel,

TemperatureProbeItemModel,

TemperatureProbeMeasurementModel,

NewsModel,

NewsContentModel,

ReleaseNoteModel,

ReleaseNoteContentModel,

ViewedNewsModel,

ComplianceIncidentConditionModel,

ComplianceIncidentModel,

BinaryComplianceIncidentModel,

BinaryComplianceIncidentConditionModel,

ReportResultsModel,

ReportActionsModel,

BatchActionModel,

NodataIncidentModel,

GatewayIncidentModel,

AlarmIncidentModel,

AlarmIncidentConditionModel,

BinaryAlarmIncidentConditionModel,

BinaryAlarmIncidentModel,

NotificationCountItemModel,

NotificationItemModel,

```
        AlarmNotificationHistoryItemModel,
        ComplianceNotificationHistoryItemModel,
        BinaryAlarmNotificationHistoryItemModel,
        BinaryComplianceNotificationHistoryItemModel,
        GatewayNotificationHistoryItemModel,
        NodataNotificationHistoryItemModel,

        DataModelFor: (deveui) => {
          return sensorDataConnection && deveui
                  ? sensorDataConnection.model('Data', DataPerCollectionSchema,
`data_${deveui}`)
            : DataModel;
        },
        DataModelForKzHistory: (deveui) => {
          return sensorDataConnection && deveui
                                  ? sensorDataConnection.model('DataKzHistory',
DataPerCollectionKzHistorySchema, `data_${deveui}`)
            : DataModel;
        },
        dataModelForConnection: sensorDataConnection,
        mainConnection: mongoose.connection,
        dbs: {
          close(cb = () => {
          }) {
            var cb1, cb2;
            if (!sensorDataConnection) {
              cb1 = true;
            } else {
              sensorDataConnection.close((err) => {
```

```
            cb1 = true;
            if (cb1 && cb2) {
              cb(err);
            }
          });
        }
        mongoose.connection.close((err) => {
          cb2 = true;
          if (cb1 && cb2) {
            cb(err);
          }
        });
      }
    }
  });
  initialized = true;
  return this;

 }
};

module.exports = models;
```

На даний конфігураційний файл посилається “package.json”, що створює можливість використовувати його при посиланні на даний репозиторій і отримувати увесь список моделей, які будуть використовуватись в подальшому в таких компонентах системи як: storage service, REST API. Даним підходом ми унеможливлюємо несумісність використання БД в частинах системи, отже всі компоненти системи мають використовувати ці моделі для з’єднання з базою даних і ніякі інші способи, аби притримуватись правильній структурі документу.

Наступним єтапом буде створення аналогічного репозиторію вже для підписки на MQTT Broker, обробки та збереження даних. Для майбутнього хостингу даного сервісу ми будемо використовувати PM2 як безкоштовний і легкий менеджер процесів, тож законфігорувати ці файли на даному єтапі не буде зайвим. Цей файл має даний вигляд:

```
require('dotenv').config();

const ttn2Stream = 'ttn2Stream';
const ttn3Stream = 'ttn3Stream';
const gpsStream = 'gpsStream';

module.exports = {
  /**
   * Application configuration section
   * http://pm2.keymetrics.io/docs/usage/application-declaration/
   */
  apps : [

    // DATA SAVE SERVICE application
    {
      name      : 'data-save-service_1',
      script    : 'index.js',
      kill_timeout: 3000,
      shutdown_with_message : true,
      wait_ready: true,
    max_restarts: 50,
      max_memory_restart: '500M',
    restart_delay: 7000,
      env: {
        stream: ttn2Stream,
```

```
    COMMON_VARIABLE: 'true',
    DEVICE_GROUP: 1,
   },
   env_production : {
    NODE_ENV: 'production'
   },
   env_uat : {
    NODE_ENV: 'uat'
   }
  },
  {
   name      : 'data-save-service_2',
   script    : 'index.js',
   kill_timeout: 3000,
   shutdown_with_message : true,
   wait_ready: true,
   max_restarts: 50,
   max_memory_restart: '500M',
   restart_delay: 7000,
   env: {
    stream: ttn2Stream,
    COMMON_VARIABLE: 'true',
    DEVICE_GROUP: 2

   },
   env_production : {
    NODE_ENV: 'production'
   },
   env_uat : {
    NODE_ENV: 'uat'
```

```
  }
},
{
 name      : 'data-save-service_3',
 script    : 'index.js',
 kill_timeout: 3000,
 shutdown_with_message : true,
 wait_ready: true,
 max_restarts: 50,
 max_memory_restart: '500M',
 restart_delay: 7000,
 env: {
  stream: ttn2Stream,
  COMMON_VARIABLE: 'true',
  DEVICE_GROUP: 3

 },
 env_production : {
  NODE_ENV: 'production'
 },
 env_uat : {
  NODE_ENV: 'uat'
 }
},
{
 name      : 'data-save-service_default',
 script    : 'index.js',
 kill_timeout: 3000,
 shutdown_with_message : true,
 wait_ready: true,
```

```
      max_restarts: 50,
      max_memory_restart: '500M',
      restart_delay: 7000,
      env: {
        stream: ttn2Stream,
        COMMON_VARIABLE: 'true',
        DEVICE_GROUP: undefined

      },
      env_production : {
        NODE_ENV: 'production'
      },
      env_uat : {
        NODE_ENV: 'uat'
      }
    },
  ],


};
```

В даний спосіб ми забезпечили себе швидким стартом нашого сервісу, логуванням помилок, які можуть трапитись під час роботи та рестартом процесу у їх випадку. Головним завданням данного сервісу є підписка на MQTT потік, перетворення даних і їх зберігання в БД для подальшого використання. Для легкого використання підписки на потік ми будемо використовувати реактивні технології з пакету RxJS. А за для безпечного використання приватних даних ми будемо використовувати “.env” файл який ми включемо в “.gitignore”, аби не зберігати чутливі ключу в репозиторію вв інтернеті. Отже на виході ми маємо такий приклад стріму і його обробники:

```
require('dotenv').config();
const { DataClient } = require('ttn');
const { Observable } = require('rxjs');
const { share } = require('rxjs/operators');
const baseTTNServerHandler = require('./data-handlers/base-ttn-server-handler');
const utcNotificationHandler = require('./data-handlers/utc-notification-handler');

const appID = process.env.KLZN_PRIVATE_NET_SERVER_APP_ID;
const privateAccessKey = process.env.KLZN_PRIVATE_NET_SERVER_APP_KEY;
const mqttAddress = process.env.KLZN_PRIVATE_NET_SERVER_ADDRESS;

/**
 *
 * @type {Observable<DataProcessingTool>}
 */
module.exports = (new Observable(observer => {
  const client = new DataClient(appID, privateAccessKey, mqttAddress);
  baseTTNServerHandler(client, observer);
  utcNotificationHandler(client, observer);
})).pipe(share());
```

Всі стріми, які були створені будуть об’єднані в одному файлі, який буде потім використовуватись при запуску екосистеми pm2. Приклад файлу, який включає в себе всі потоки MQTT нашої системи:

```
require('dotenv').config();
require('./lib/logging-tool');
const serviceMessagingClient = require('./lib/service-messaging-client');

console.log('NODE_ENV', process.env.NODE_ENV);
```

```
console.log('IS NEW FIRMWARE', process.env.new_firmware);
const { dbs, AlarmRecordModel } =

require('models').initiateMongooseModels(process.env.KLZN_SERVER_MONGODB,
undefined,
   process.env.KLZN_SERVER_MONGODB_SENSOR_DATA);

process.on('message', function(msg) {
  if (msg === 'shutdown') {
    dbs.close((err) => {
      if (err) console.error('db stopped', err);
      console.log('!!shutdown exit with code!! ', !!err);
      process.exit(err ? 1 : 0);
    });
    console.log('Closing all connections...SHUTDOWN');
  }
});

process.on('SIGINT', function() {
  dbs.close(function(err) {
    if (err) console.error('db stopped', err);
    console.log('Mongoose disconnected on app termination');
    process.exit(err ? 1 : 0);
  });
});

process.on('uncaughtException', function(e) {
  console.error('unCaughtException', e);
  dbs.close();
  process.exit(1);
```

```
});


const { merge, of } = require('rxjs');
const { mergeMap } = require('rxjs/operators');
const privateNetWorkServerData$ = require('./data-streams/private-network-server-data');
const falconNetWorkServerData$ = require('./data-streams/falcon-network-server-data');
const gpsServerData$ = require('./data-streams/gps-server-data');
const ttn3Stream$ = require('./data-streams/ttn-v3-stream');
const ttn3DemoStream$ = require('./data-streams/ttn-v3-demo-stream');
const ttn3c30stream$ = require('./data-streams/ttn-v3-c30-stream');
const ttn2KoolzoneHistoryStream$ = require('./data-streams/ttn2-koolzone-history-stream');
const ttn3KoolzoneHistoryStream$ = require('./data-streams/ttn3-koolzone-history-stream');
const ttn3KoolzoneHistoryLegacyStream$ = require('./data-streams/ttn3-koolzone-history-legacy-stream');
const ttn3KoolzoneCalibrationStream$ = require('./data-streams/ttn3-koolzone-calibration');

let streams$ = [
  privateNetWorkServerData$,
  gpsServerData$,
  falconNetWorkServerData$,
  ttn3DemoStream$,
  ttn2KoolzoneHistoryStream$,
];
let streamMap = { // fake streams are running of undefined Device Group
  gpsStream: [gpsServerData$],
```

```
  ttn2Stream: [privateNetWorkServerData$, falconNetWorkServerData$],
  ttn3Stream: [ttn3Stream$],
  ttn3DemoStream: [ttn3DemoStream$],
  ttn3c30Stream: [ttn3c30stream$],
  ttn2KoolzoneHistoryStream: [ttn2KoolzoneHistoryStream$],
  ttn3KoolzoneHistoryStream: [ttn3KoolzoneHistoryStream$],
  ttn3KoolzoneHistoryLegacyStream: [ttn3KoolzoneHistoryLegacyStream$],
  ttn3KoolzoneCalibration: [ttn3KoolzoneCalibrationStream$],
};
if (process.env.hasOwnProperty('stream')) {
  const stream = process.env.stream;
  let matchedStreams$ = streamMap[stream] || [];
  streams$ = [...matchedStreams$];
}

merge(
  ...streams$,
)
  .pipe(
    mergeMap(sensorTypesHandlerObject => {
                                        const deviceTypeHandlerService =
sensorTypesHandlerObject.deviceTypeHandlerService;
      const processedData = deviceTypeHandlerService.getProcessedData();
      const sensorDoc = sensorTypesHandlerObject.getSensorDoc();
      const deviceDoc = sensorTypesHandlerObject.deviceDoc;
      const device = deviceDoc.toObject();
      const sensor = sensorDoc.toObject();
      let sensorObject = { ...sensor, ...device };
      sensorObject.sensor_item_name = sensor.name;
      sensorObject._id = sensor._id;
```

```
      sensorObject.device = device._id;
      const sensorDocPromise = sensorDoc.save().catch(error => console.error(error));
                  const wasConnected = processedData[`${ sensorDoc.sensor_type }_was_connected`];
      AlarmRecordModel.saveStatsNoData('end', sensorObject, wasConnected).then(v => {
      }).catch(e => {
        console.error(e);
      });

      return of({
        sensor: { _id: sensorDoc._id },
        was_connected: wasConnected,
        sensorDocPromise,
      });
    }),
  )
  .subscribe(
    (obj) => {
      var messageToAlarmService = JSON.stringify({
        sensor: { _id: obj.sensor._id },
        was_connected: obj.was_connected,
      });
      var promise = obj.sensorDocPromise;
      promise.then(sensorDoc => {
                                  serviceMessagingClient.publish('koolzone/alarm-service', messageToAlarmService);
      });
    },
    (error) => {
```

```
      console.error(error);
      dbs.close();
      process.exit(1);
    },
  );

try {
  var execMainPath = require.resolve('pm2-service-monitoring');
} catch (e) {} finally {
  if (!execMainPath) {
    return;
  }
  const runHealthCheck = require('pm2-service-monitoring');
  runHealthCheck();
}
```

Також так виглядає файл який використовує моделі, створені нами раніше:

```
const DataProcessingTool = require('../../sensor-types/configuration-based-handlers/data-processing-tool');
const DeviceTypesToHandlers = require('../../sensor-types/configuration-based-handlers/device-types-to-handlers-service');
const {
  DataModelFor,
  BinaryDataEventModel,
  NodataIncidentModel,
} = require('models');
const { ALL_CONFIGURATION_TYPES } = require('common-module');
const { DF702NL } = ALL_CONFIGURATION_TYPES;
const _ = require('lodash');
const koolzoneHistory = require('../../lib/koolzone-history');
```

```
const draginoHistory = require('../../lib/dragino-history');
const getBatteryLevel = require('../../lib/airtime-to-battery-level');
const setChannelsFromType = require('../../lib/channels/set-channels-from-type');
const outgoingApiClient = require('../../lib/outgoing-api-client');
console.log('NODE_ENV', process.env.NODE_ENV);

const serviceMessagingClient = require('../../lib/service-messaging-client');

function extractReadingsAsObject(typeHandlersArrayExecuted, deveui) {
  let outgoingApiData = {
    deveui,
    sensors: [],
  };
  const readingsObject = typeHandlersArrayExecuted.reduce((aggregated, current) => {
    const sensorDoc = current.getSensorDoc();

    if (sensorDoc.is_enabled) {
      outgoingApiData.sensors.push({
        sensor_type: sensorDoc.sensor_type,
        reading: sensorDoc.last_reading,
        channel: sensorDoc.channel,
      });
      aggregated[sensorDoc.sensor_type_old] = sensorDoc.last_reading;

    }
    return aggregated;
  }, {});

  return { readingsObject, outgoingApiData };
}
```

```
function getBinaryStatesCounter(act_st_v, last_reading, sts) {
  if ((act_st_v === 1 && last_reading === 1) || (act_st_v === 0 && last_reading === 0)) {
    sts.act_stc = (sts.act_stc || 0) + 1;
  }
  if ((act_st_v === 1 && last_reading === 0) || act_st_v === 0 && last_reading === 1) {
    sts.rest_stc = (sts.rest_stc || 0) + 1;
  }

  return sts;
}

module.exports = async function(observer, incomingData, client, applicationId) {
  try {
    const deveui = incomingData.hardware_serial;
    const devid = incomingData.dev_id;
    incomingData.deveui = deveui;


    const devicePort = incomingData.port;
      const dataProcessingTool = await DataProcessingTool.load(incomingData, client, deveui, devicePort);
    const frames_count = incomingData.counter;
    // @TODO just put check if gps device
    const latest_airtime = _.get(incomingData, 'metadata.airtime', null);

    const deviceTypeHandlerObj = DeviceTypesToHandlers.create(dataProcessingTool);
    if (!deviceTypeHandlerObj) {
```

```
    return;
  }
  logger.deviceData({
    raw: (incomingData.payload_raw || '').toString('hex'),
    'deveui-raw': deveui,
  });

  const result = deviceTypeHandlerObj.exec();
  const deviceDoc = deviceTypeHandlerObj.getDeviceDoc();
  const processedData = deviceTypeHandlerObj.getProcessedData();
  if (!result) { // 0000 no parsed data, duplicates
    // console.log('No result');
                                if (_.isArray(deviceTypeHandlerObj.historyData) &&
deviceTypeHandlerObj.historyData.length) {
      await koolzoneHistory.checkHistoryOnReading(deviceTypeHandlerObj);
      /*END KOOLZONE HISTORY*/

      await draginoHistory.checkHistoryOnReading(deviceTypeHandlerObj);
    }
    return;
  }
  /*KOOLZONE HISTORY*/
  await koolzoneHistory.checkHistoryOnReading(deviceTypeHandlerObj);
  /*END KOOLZONE HISTORY*/

  await draginoHistory.checkHistoryOnReading(deviceTypeHandlerObj);
  deviceDoc.application_id = applicationId;
  const isBinDevice = deviceDoc.sensor_configuration_type === DF702NL;
   let airtime_agg = _.isNil(latest_airtime) ? null : (deviceDoc.airtime_aggregated || 0) +
latest_airtime;
```

```
  deviceDoc.frames_count = frames_count;
  deviceDoc.latest_airtime = latest_airtime;
  if (!_.isNil(airtime_agg)) {
    deviceDoc.airtime_aggregated = airtime_agg + (deviceDoc.airtime_sum || 0);
  }
        const batteryLevel = getBatteryLevel(deviceDoc.airtime_aggregated, deviceDoc.sensor_configuration_type, deviceDoc.last_vdd);
  deviceDoc.battery_level = batteryLevel;

  if (_.isNumber(processedData.dbmc)) {
    let dbmc = processedData.dbmc;
    deviceDoc.signal_strength_bars = null;

    if (dbmc >= -189) {
      deviceDoc.signal_strength_bars = 3;
    } else if (dbmc >= -240 && dbmc <= -189.1) {
      deviceDoc.signal_strength_bars = 2;
    } else {
      deviceDoc.signal_strength_bars = 1;
    }
    deviceDoc.signal_strength = dbmc;
  }
        // console.log('!!!!', processedData.rssi, processedData.lsnr, dbmc, deviceDoc.signal_strength_bars);

  const gateways = _.get(incomingData, 'metadata.gateways', []);
  const last_seen_gateways = gateways.map(g => {
    const eui = _.last((g.gtw_id || '').split('-'));
    return {
      dbmc: g.dbmc,
```

```
      rssi: g.rssi,
      snr: g.snr,
      eui,
     };
   });
   deviceDoc.last_seen_gateways = last_seen_gateways;
   deviceDoc.last_seen_at = new Date;
   if (deviceDoc.opened_nodataincident) {
        NodataIncidentModel.updateOne({ _id: deviceDoc.opened_nodataincident }, { ended_at: new Date }).catch(e => console.error(e));
     deviceDoc.opened_nodataincident = null;
   }

   if (last_seen_gateways.length) {
     deviceDoc.last_seen_gateways_at = new Date;
   }

   if (_.isNumber(deviceDoc.last_vdd)) {
     let vddWarningSet = false;
     const sensorConfigTypesForVdd = [
      'kzpt1000v2', 'kz01sub80', 'kz01plus300', 'kz01sub200',
      'klspt1000v2', 'klssub80', 'klsplus300', 'klssub200',
     ];
     const manufacturer = deviceDoc.manufacturer || '';
     if (manufacturer.toLowerCase() === 'dragino') {
      if (deviceDoc.last_vdd === 0 || deviceDoc.last_vdd === 1) {
       deviceDoc.vdd_warning = 2;
      }
      if (deviceDoc.last_vdd === 2) {
       deviceDoc.vdd_warning = 1;
```

```
  }
  if (deviceDoc.last_vdd > 2) {
    deviceDoc.vdd_warning = 0;
  }
  vddWarningSet = true;
}
if (manufacturer.toLowerCase() === 'laird') {
  if (deviceDoc.last_vdd === 0) {
    deviceDoc.vdd_warning = 2;
  }
  if (deviceDoc.last_vdd === 1) {
    deviceDoc.vdd_warning = 1;
  }
  if (deviceDoc.last_vdd > 1) {
    deviceDoc.vdd_warning = 0;
  }
  vddWarningSet = true;
}

if (sensorConfigTypesForVdd.includes(deviceDoc.sensor_configuration_type)) {
  if (deviceDoc.last_vdd < 3.39 && deviceDoc.last_vdd > 3.11) {
    deviceDoc.vdd_warning = 1;
  }
  if (deviceDoc.last_vdd < 3.10) {
    deviceDoc.vdd_warning = 2;
  }
  if (deviceDoc.last_vdd >= 3.40) {
    deviceDoc.vdd_warning = 0;
  }
  vddWarningSet = true;
```

```
    }
    if (!vddWarningSet) {
      if (deviceDoc.last_vdd < 3.39 && deviceDoc.last_vdd > 3.11) {
        deviceDoc.vdd_warning = 1;
      }
      if (deviceDoc.last_vdd < 2.9) {
        deviceDoc.vdd_warning = 2;
      }
      if (deviceDoc.last_vdd >= 3.40) {
        deviceDoc.vdd_warning = 0;
      }
    }
  }
  deviceDoc.save().then(
    () => {
      // console.log('saved device doc')
    },
  ).catch(error => console.error(error));
                                    const sensorTypesHandlerObjects =
deviceTypeHandlerObj.sensorTypesHandlerObjects;

  let {
    readingsObject,
    outgoingApiData,
  } = extractReadingsAsObject(sensorTypesHandlerObjects, deveui);
  const gatewaysInMeta = _.get(incomingData.metadata, 'gateways', [])
    .map(obj => {
      if ((obj.gateway_ids || {}).hasOwnProperty('forwarded')) {
        obj.forwarded = obj.gateway_ids.forwarded;
      }
```

```
    return _.omit(obj, ['gateway_ids', 'uplink_token']);
  });
if (incomingData.metadata) {
  incomingData.metadata.gateways = gatewaysInMeta;
}
logger.deviceData({
  port: incomingData.port,
  deveui,
  counter: incomingData.counter,
  meta: incomingData.metadata,
  parsed: {
    ...readingsObject,
    sl: deviceDoc.signal_strength_bars,
    bl: deviceDoc.battery_level,
  },
});


sensorTypesHandlerObjects.map(dItem => {
  observer.next(dItem);
});

const sensorDataCreatedAt = processedData.created_at || new Date;

if (!sensorTypesHandlerObjects.length) {
  console.log('POSSIBLE DUPLICATES', deveui);
  return;
}

if (isBinDevice) {
```

```
    const usdist_metadata = { ...processedData.usdist_metadata };
    readingsObject = { usdist: processedData.usdist, usdist_metadata };
    processedData.usdist_metadata.usdist = processedData.usdist;

  }
  const dataModel = {
    data: processedData,
    device: deviceDoc._id,
    created_at: sensorDataCreatedAt,
    updated_at: new Date,
  };
  setChannelsFromType(deviceDoc.sensor_configuration_type, dataModel);
  const possibleTemperatureNaN = _.get(dataModel, 'data.temperature');
  if (possibleTemperatureNaN === 'NaN' || _.isNaN(possibleTemperatureNaN)) {
    console.log('TEMPERATURE_NAN setting null for device', deviceDoc.deveui);
    dataModel.data.temperature = null;
  }

                DataModelFor(deviceDoc.deveui).create(dataModel).catch(error =>
console.error(error));
  for (let sensorTypeObj of sensorTypesHandlerObjects) {
    const sensorDoc = sensorTypeObj.getSensorDoc();
    if (!sensorDoc.is_enabled || sensorDoc.sensor_type !== 'binary') {
      continue;
    }
    const binaryDataDoc = await BinaryDataEventModel.findOne({
      deveui,
      ch: sensorDoc.channel,
      ended_at: { $exists: false },
    });
```

```
const act_st_v = sensorDoc.act_st_v;

if (_.isNil(act_st_v)) {
  continue;
}
const last_reading = sensorDoc.last_reading;
if (_.isNil(last_reading)) {
  continue;
}
let stsFromDoc = binaryDataDoc ? {
  act_stc: binaryDataDoc.act_stc,
  rest_stc: binaryDataDoc.rest_stc,
} : {};
let sts = getBinaryStatesCounter(act_st_v, last_reading, stsFromDoc);
if (!binaryDataDoc) {
  BinaryDataEventModel.create({
    sensor: sensorDoc._id,
    act_st_v,
    deveui,
    ch: sensorDoc.channel,
    started_at: new Date,
    last_st_at: new Date,
    last_st: last_reading,
    ...sts,
  }).catch(e => console.error(e));
} else {
     if ((act_st_v === 1 && last_reading === 0 && binaryDataDoc.last_st === 1) ||
(act_st_v === 0 && last_reading === 1 && binaryDataDoc.last_st === 0)) {
    // close event create new data event model
    binaryDataDoc.ended_at = new Date; // ended at time when rest state recieved
```

```
        await binaryDataDoc.save(); // need to wait as we have changeEventListerer in binary Incident service
        BinaryDataEventModel.create({
          sensor: sensorDoc._id,
          act_st_v,
          deveui,
          ch: sensorDoc.channel,
          started_at: new Date,
          last_st_at: new Date,
          last_st: last_reading,
          rest_stc: 1,
        }).catch(e => console.error(e));
      } else {
                  if ((binaryDataDoc.act_st_v === 1 && last_reading === 1 && !binaryDataDoc.st_change_at && binaryDataDoc.rest_stc)
                    || (binaryDataDoc.act_st_v === 0 && last_reading === 0 && !binaryDataDoc.st_change_at && binaryDataDoc.rest_stc)) {
          binaryDataDoc.st_change_at = new Date;
        }
        binaryDataDoc.act_stc = sts.act_stc;
        binaryDataDoc.rest_stc = sts.rest_stc;
        binaryDataDoc.last_st = last_reading;
        binaryDataDoc.last_st_at = new Date;
        binaryDataDoc.save().catch(e => console.error(e));
      }
    }
  }

  deviceDoc.company && await deviceDoc.populate('company', '_id integration_status group_name name');
```

```
  outgoingApiData.vdd = deviceDoc.last_vdd;

  if (!process.env.LIVE_FEED) {
    return;
  }
        if (deviceDoc.company && deviceDoc.company._id.toString() ===
'5e9858e53ab6da400dbaaefc') {
          outgoingApiClient.publish(`koolzone/v1/anttelecom/data/${ deveui }`,
JSON.stringify(outgoingApiData));
  }

        if (deviceDoc.company && deviceDoc.company._id.toString() ===
'5b2a0fee07c69075aa4d1df5') {
        outgoingApiClient.publish(`koolzone/v1/drschnellgmbh/data/${ deveui }`,
JSON.stringify(outgoingApiData));
  }

        if (deviceDoc.company && deviceDoc.company._id.toString() ===
'636d5eaedd02940a6bfdfd3d') {
          outgoingApiClient.publish(`koolzone/v1/ecobat/data/${ deveui }`,
JSON.stringify(outgoingApiData));
  }

  if (deviceDoc.company && deviceDoc.company.integration_status === 'enabled' &&
deviceDoc.company.group_name) {
    const groupName = deviceDoc.company.group_name;
    let readingObj = {
      name: deviceDoc.name,
      deveui: deviceDoc.deveui,
```

```
      ...readingsObject,
      rssi: deviceDoc.last_rssi,
      snr: deviceDoc.last_lsnr,
      vdd: deviceDoc.last_vdd,
      timestamp: (new Date(sensorDataCreatedAt)).valueOf(),
    };

    if (groupName.toLowerCase() === 'alegria') {
      console.log('alegria', readingObj);
      if (readingObj.hasOwnProperty('temperature_0')) {
        readingObj['temperature'] = readingObj['temperature_0'];
        delete readingObj['temperature_0'];
        delete readingObj['temperature_1'];
      }
    }
    const data = JSON.stringify({
      reading: readingObj,
      topic: `sensor-readings/${ groupName.toLowerCase() }`,
    });
    serviceMessagingClient.publish('koolonze/resellers-messages', data);

  }
 } catch (e) {
  console.error(e);
  observer.error(e);
 }
};
```

За аналогічним сценарієм ми створюємо репозиторій для REST API, підключаємо до нього наші моделі і створюємо з’єднання з нашою БД. Як було згадано раніше для серверної частини додатку буде використовуватись фреймвор

Express.js який працює на мові програмування Node.js, яка в свою чергу є однопотоковою мовою програмування. Тому для збільшення пропускної можливості нашого серверу за допомогою екосистеми PM2 ми створемо кластер процесів, що полегшить користування клієнтів. Для аутентифікації і авторизації в системі ми будемо використовувати метод JWT. Токен міститеме в собі час, до якого він дійсний, ім’я та прізвище користувача та його рівні доступу до частин системи.

Таким самим способом ми створемо інфраструктуру для WEB додатку на основі фреймворку Angular. В цій роботі було використано підхід SPA, PWA та Lazy Loading для більш комфортного використання додатку користувачем. SPA в поєднанні з Lazy Loading дозволяє нам зменшити навантаження на мережі інтернет користувача відображаючи лише потрібні модулі додатку. Наприклад, якщо користувач вперший раз заходить в додаток на сторінку пристроїв, він отримає лише код, який потрібен для відображення лише цієї сторінки, отже код для сторінки маршрутизаторів не буде завантажений до моменту як користувач відвідає цю сторінку. PWA дозволить нам користуватися Service Workers котрі дозволяють більш точніше кєшувати завантажену інформацію, вибирати стратегію запитів і дає можливість користуватись додатком під час слабкого або повністю відсутнього інтернету. Враховуючи всі ці аспекти додаток був створений з урахуванням відображення статусу пристроїв та маршрутизаторів. Можливістю відображення і порівняння даних за допомогою бубліотеки Highcharts.

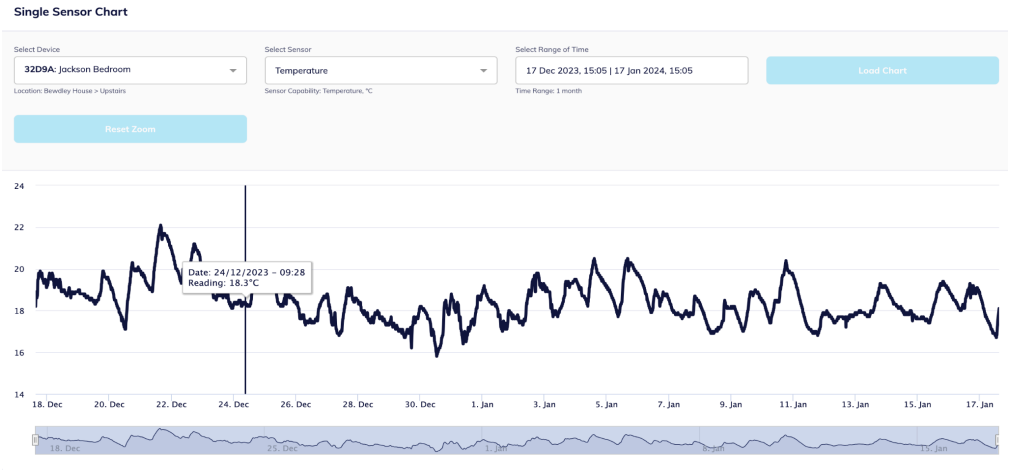

Рисунок 3.5 - Лінійний графік датчику температури за місяць

Рисунок 3.6 - Лінійний графік порівняння двох температурних датчиків

Рисунок 3.7 - Синхронізовані графіки пристрою котрий містить датчики температури, вологості і вуглекислого газу

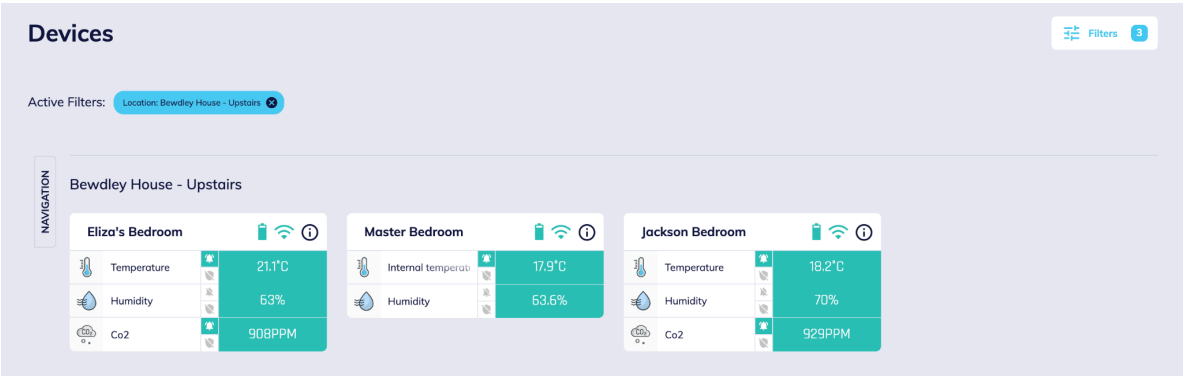

Рисунок 3.8 - Сторінка статусу пристроїв і їх датчиків

ВИСНОВОК

У ході дослідження для дипломної роботи були отримані наступні висновки:

1. Виявлено, що автоматизація IT-процесів, яка недавно розглядалася як простий інструмент або сценарій для підтримки бізнесу, тепер розглядається як стратегічна ініціатива та довгострокова IT-стратегія.
2. IT-фахівці можуть вибирати з широкого спектру засобів для автоматизації як простих рутинних, так і складних процесів.
3. Найбільш поширеними серед них, завдяки доступним кодам, наявності навчальних матеріалів та простоті використання, є інструменти такі як MQTT, TTI, LoRaWAN.
4. Результатом виконання цього дослідження є створена система автоматизації та візуалізації даних на основі технології IoT.

   Отримані результати відповідають сучасним викликам, пов'язаним із необхідністю інтеграції та впровадженням моніторингу пристроїв IoT

Список використаної літератури

1. *Портал "https://www.thethingsindustries.com/docs/"*
2. *Портал "https://www.wikipedia.org/"*
3. *Портал "https://www.wi-fi.org/discover-wi-fi/internet-things"*
4. *Портал "https://www.highcharts.com/"*

Державний університет інформаційно-комунікаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
“РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ IOT”

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр. ІСДМ-61
Дмитро БІНСЬКИЙ
Керівник: д.т.н., завідувач кафедри ІПЗАС
Каміла СТОРЧАК

Київ - 2024

- ***Актуальність теми:*** необхідність вирішення інженерно-технічних питання автоматизації і візуалізації даних отриманих з девайсів на основі технології IoT . Тому тема роботи є актуальною.
- ***Об`єкт дослідження:*** система моніторингу IoT.
- ***Предмет дослідження:*** система моніторингу і візуалізації даних отриманих з IoT девайсів.
- ***Мета дослідження:*** побудова системи автоматизації і візуалізації отриманих даних за допомогою технології IoT.
- ***Завдання дослідження:***
  - здійснити аналіз перспективних технологій для моніторингу;
  - дослідити способи обробки і візуалізації даних;
  - дослідити можливість онлайн моніторингу девайсів основаних на технології IoT;
  - обґрунтувати підходи реалізації поставлених задач.

Рис. 3.1 - Девайс для отримання температури і вологи в приміщенні

Рис. 3.2 - Девайс відкриття та закриття дверей

Рис. 3.3 - Девайс витоку води та міні вербіа девайсу відкриття та закриття дверей

Рис. 4.1 - Гейтвей для передачі даних

Рис. 3.4 - Девайс моніторингу спожитої енергії та віддаленого вимикання

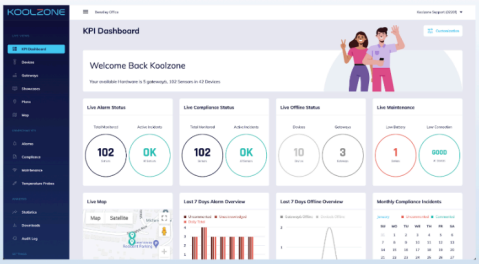

Рис. 5.1 - Глобальне відображення статистики

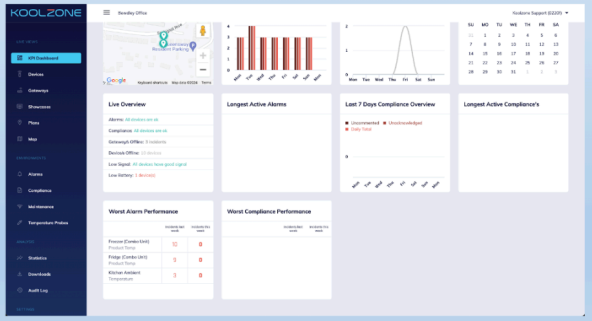

Рис. 5.2 - Глобальне відображення статистики 2

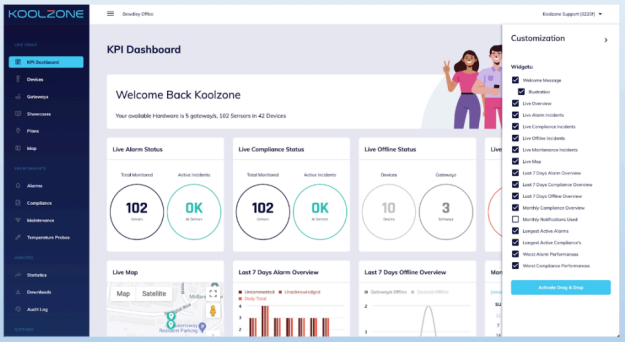

Рис. 5.3 - Конфігурація відображення статистики

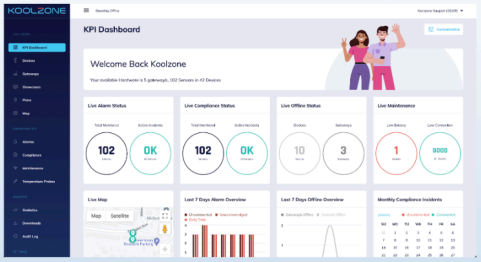

Рис. 6.1 - Глобальне відображення статистики

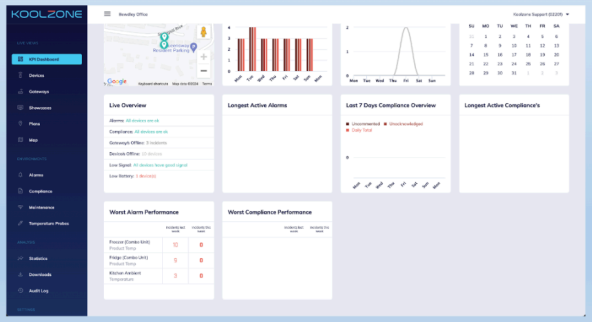

Рис. 6.2 - Глобальне відображення статистики 2

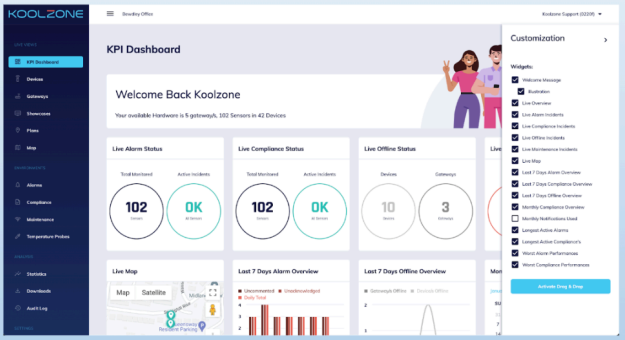

Рис. 6.3 - Конфігурація відображення статистики

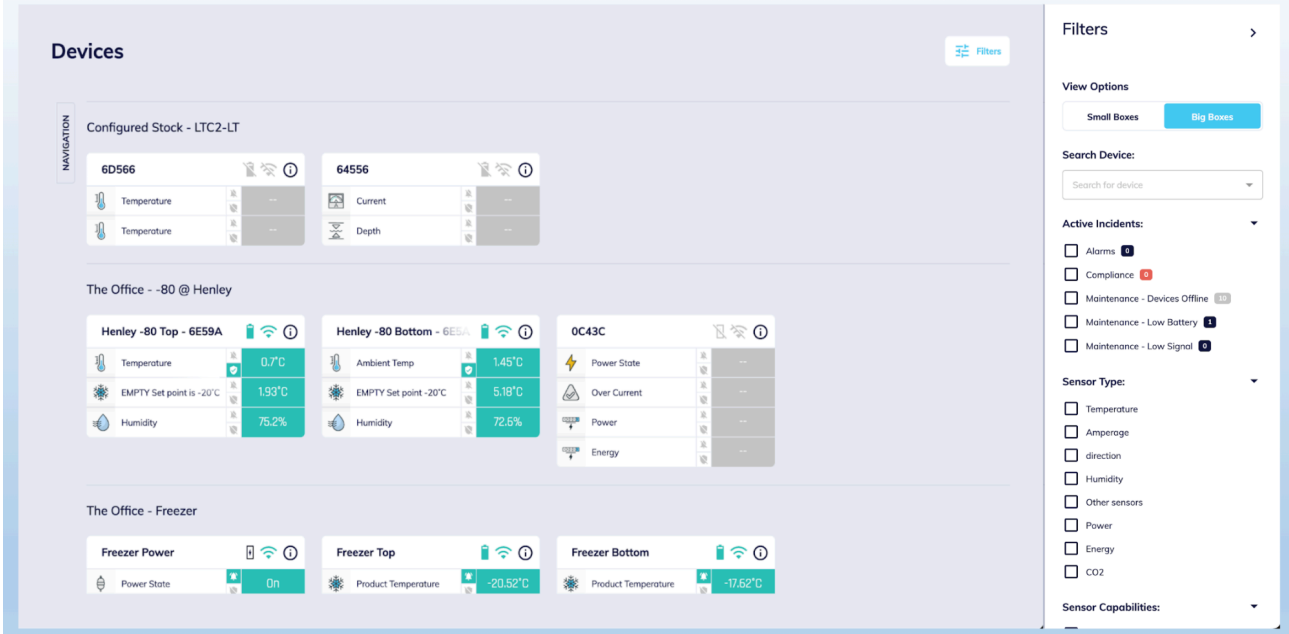

Рис. 7.1 – Список девайсів та фільтрація

7

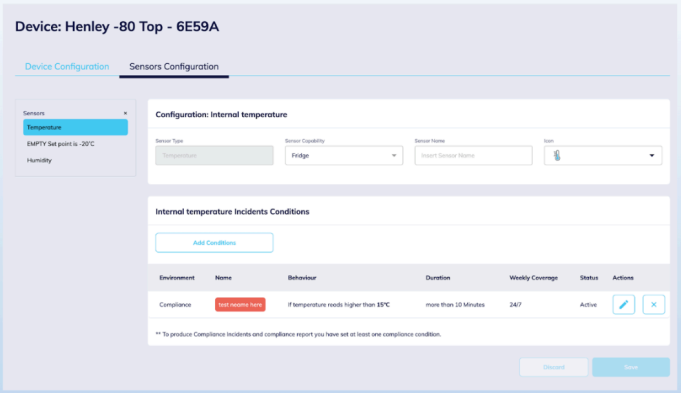

Рис. 8.1 – Конфігурація сенсорів

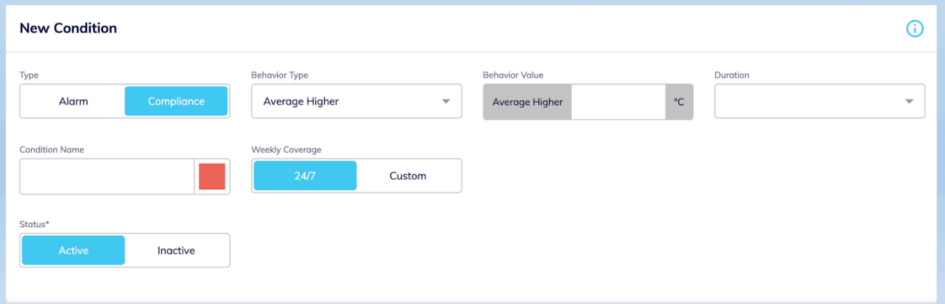

Рис. 8.1 – Створення умови для створення інцинденту сенсора

8

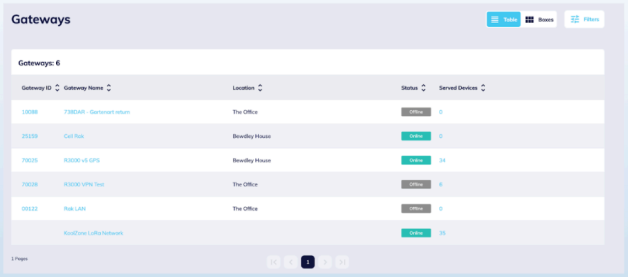

Рис.9.1 - Список Гейтвеїв

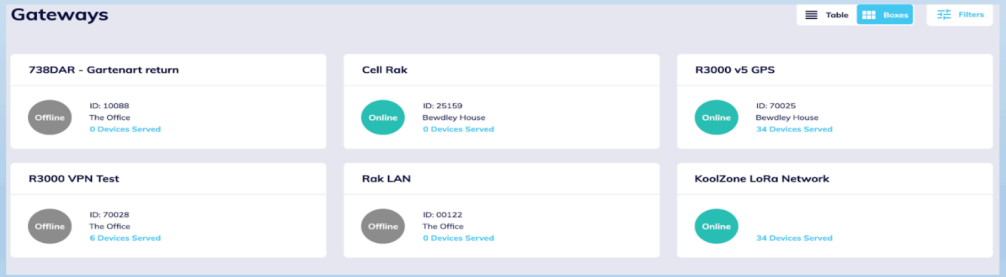

Рис. 9.2 - Табличний список Гейтвеїв

9

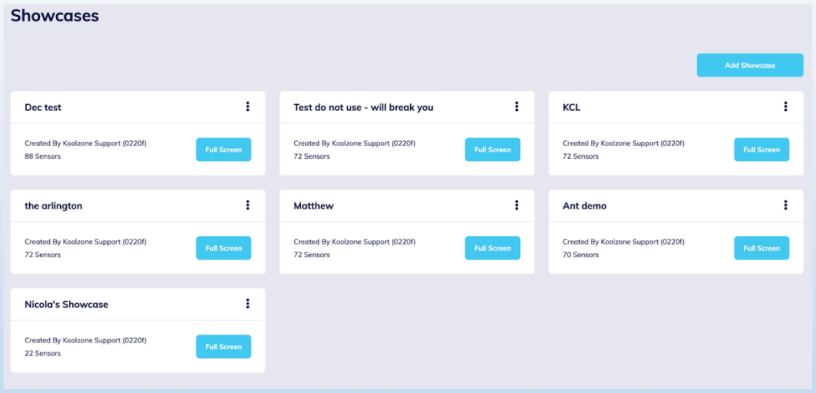

Рис 10.1 – Список групп девайсів

Рис 10.2 – Відображення групп девайсів

10

Alarm Incidents: 80

Device / Sensor	Location	Type	Name	Triggered	Duration	Reactions
Fridge (Combo Unit) - Product ...	Bewdley House > Kitchen	Reads Lower	Too Cold	14/01/2024 - 12:54	29m, 59s	
Freezer (Combo Unit) - Product...	Bewdley House > Kitchen	Reads Lower	Too Cold	14/01/2024 - 04:31	19m, 59s	
Fridge (Combo Unit) - Product ...	Bewdley House > Kitchen	Reads Lower	Too Cold	14/01/2024 - 03:24	1h, 19m	
Freezer (Combo Unit) - Product...	Bewdley House > Kitchen	Reads Higher	Too Warm	13/01/2024 - 06:51	1h, 29m	
Fridge (Combo Unit) - Product ...	Bewdley House > Kitchen	Reads Lower	Too Cold	13/01/2024 - 00:54	1h, 59m	
Fridge (Combo Unit) - Product ...	Bewdley House > Kitchen	Reads Lower	Too Cold	13/01/2024 - 00:34	9m, 59s	
Freezer (Combo Unit) - Product...	Bewdley House > Kitchen	Reads Higher	Too Warm	12/01/2024 - 11:11	1h, 39m	
Freezer (Combo Unit) - Product...	Bewdley House > Kitchen	Reads Lower	Too Cold	12/01/2024 - 05:51	1h, 19m	

Рис 11.1 – Список створених інциндентів

11

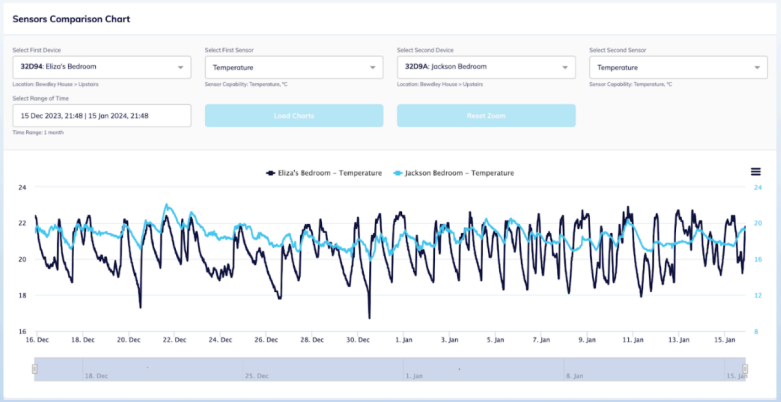

Рис 12.1 – Графік порівняння двох сенсорів різних девайсів за місяць

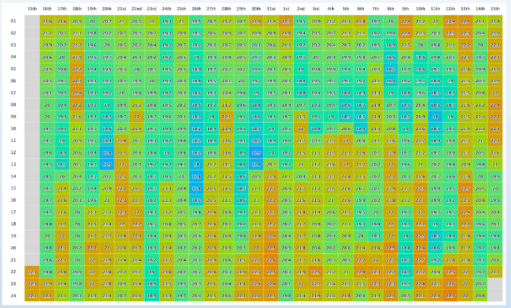

Рис 12.2 – Таблиця відображення температурного сенсора за місяць по годинам

12

ВИСНОВКИ

- У кваліфікаційній магістерській роботі досліджені основні перспективні технології для моніторингу, їх аналіз і візуалізація.
- Для моніторингу обрані сенсора: бінарні, температури, енергетичні і вологості. Через їх найбільшу вживаність в сучасному світі.
- Здійснений процес планування та прийняття рішення щодо вибору способів зберігання і візуалізації даних.
- Візуальна частина система створена на основі фреймворку Angular з технологією PWA. Серверна сторона побудована на мові Node.js з використанням фреймворку Express.js та NoSQL бази даних MongoDB.

13

ДЯКУЮ ЗА УВАГУ!

14