КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено
Завідувач кафедри ІСТ
Олександр КУЧАНСЬКИЙ
(підпис) (ім'я, ПРІЗВИЩЕ)
"___" _____ 2022р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

спеціальності 126 «Інформаційні системи та технології»

освітньої програми «Програмні технології інтернет речей»

на тему: «Розробка бази даних для підвищення ефективності системи енергопостачання на основі технологій IoT»

Виконав: студент 4 курсу, групи ІР-41
(шифр групи)

Щербак Владислав _____
(Ім'я, ПРІЗВИЩЕ) (підпис)

Керівник д.т.н. професор Михайло СТЕПАНОВ _____
(посада, науковий ступінь, вчене звання, Ім'я, ПРІЗВИЩЕ) (підпис)

Консультант нормо контроль к.т.н., доц. Ростислав ЛІСНЕВСЬКИЙ
(назва розділу) (посада, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Рецензент доцент кафедри ІСТ ЧДТУ, к.т.н., Володимир ДРУЖИНІН
(посада, науковий ступінь, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у пояснювальна записка не має запозичень з праць інших авторів без відповідних посилань.

Здобувач освіти _____
(підпис)

Київ – 2022 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційні системи та технології
Освітній рівень Бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітня програма Програмні технології інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСТ
Олександр КУЧАНСЬКИЙ
_____
«__» _____ 2022 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувач освіти: Владислав ЩЕРБАК

Група: ІР-41

1. **Тема кваліфікаційна робота бакалавра**: «Розробка бази даних для підвищення ефективності системи енергопостачання на основі технологій IoT».

Затверджена протоколом засідання кафедри ІСТ №05/21_22 від 03.12.2021 року

2. **Строк подання студентом готової роботи** – «21» червня 2021 р.

3. **Вихідні дані до роботи**: дослідження в області енергопостачання та IoT, проект бази даних (функціональна модель процесу, ділова модель процесу, концептуальна схема, контекстна модель, логічна і фізична модель), програмний код серверної частини проекту, програмний код клієнтської частини проекту.

4. **Зміст роботи**: 1 АНАЛІЗ ТА ОПИС СФЕРИ ЗАСТОСУВАННЯ БАЗИ ДАНИХ ІОТ СИСТЕМИ ЕНЕРГОПОСТАЧАННЯ. ПОСТАНОВКА ЗАДАЧІ (опис об’єкту дослідження, проблематика старої системи моніторингу системи енергопостачання, аналіз аналогічних існуючих рішень); 2 ПРОЕКТУВАННЯ БАЗИ ДАНИХ СИСТЕМИ МОНІТОРИНГУ ЕНЕРГОПОСТАЧАННЯ

(проектування бази даних програмного продукту для моніторингу сисиеми енергопостачання, опис компонентів системи); 3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ ЕНЕРГОПОСТАЧАНЯ (створення бази даних, розробка серверної API частини програмного продукту, розробка клієнтської частини програмного продукту, верстка фронтєнду).

5. **Перелік графічного матеріалу**: функціональна модель розробки IoT системи, концептуальна схема роботи IoT системи, контекстна діаграма процесу діяльності підприємства IoT системи, діаграми декомпозиції системи по відділам, діаграма дерева вузлів підприємства, діаграма прецедентів, діаграма діяльності, послідовності, класів, логічна модель бази даних, фізична модель бази даних, схема MVC роботи веб-додатку.

6. **Календарний план виконання роботи:**

Етапи виконання кваліфікаційної роботи бакалавра	**Термін виконання**	**Результат виконання**
1. Вибір тематики кваліфікаційної роботи бакалавра	01.09.2021-01.10.2021	виконано
2. Наказ про затвердження тем кваліфікаційної роботи бакалавра та призначення керівників	03.12.2021	виконано
3. Розробка плану кваліфікаційної роботи бакалавра і його погодження з керівником	25.12.2021	виконано
4. Написання I розділу кваліфікаційної роботи	19.03.2022	виконано
5. Написання II розділу кваліфікаційної роботи	25.04.2022	виконано
6. Написання III розділу кваліфікаційної роботи	29.04.2022	виконано
7. Підготовка висновків і пропозицій	04.05.2021	виконано
8. Попередній захист кваліфікаційної роботи	12.05.2022	виконано
9. Перевірка на плагіат	13.05.2022-15.06.2022	виконано

10. Нормоконтроль	02.06.2022-06.06.2022	виконано
11. Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді	15.06.2022	виконано
12. Захист кваліфікаційної роботи бакалавра	23.06.2022-24.06.2022	

Дата видачі завдання «__» _____ 2022 р.

Керівник роботи: д.т.н. професор Михайло СТЕПАНОВ _____
(підпис)

Завдання прийняв до виконання:
Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41

Щербак ВЛАДИСЛАВ _____
(Власне Ім’я, ПРІЗВИЩЕ) (підпис)

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра Інформаційних систем та технологій

Освітня програма «Програмні технології інтернет речей»

Кваліфікаційна робота бакалавра Щербака ВЛАДИСЛАВА

Тема роботи: «Розробка бази даних для підвищення ефективності системи енергопостачання на основі технологій IoT».

Мета кваліфікаційної роботи бакалавра – розробка програмного продукту для системи моніторингу енергопостачання яке включає базу даних з серверною та клієнтською частиною.

Об'єкт дослідження – функціонування системи моніторингу енергопостачання для забезпечення моніторингу за "енергозонами".

Предмет дослідження – система моніторингу енергопостачанням.

Апробація результатів. Основні положення і результати досліджень, викладені у проекті, пройшли апробацію на XVIII Міжнародній науково-технічній конференції «Проблеми інформатизації» у грудні 2021-го року.

Кваліфікаційна робота бакалавра складається зі змісту, вступу, основної частини, яка включає три розділи, висновків та списку використаних джерел. Всього 55 сторінок.

КЛЮЧОВІ СЛОВА: IoT, база даних, енергетика, моніторинг, Microsoft SQL Server, додаток, програмний продукт, API, Angular, C#, HTML, TypeScript, CSS.

ABSTRACT

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

Faculty of Information Technology

Department of Information Systems and Technology

Educational Program "Software Technologies of the Internet of Things"

Qualification work of master Shcherbak Vladyslav.

Work topic: "Development of a database to increase the efficiency of the energy supply system based on IoT technologies."

The purpose of the bachelor's qualification work is software product development of a software product for the energy supply monitoring system, including a database, a server and a client part.

The object of research is using of the energy supply monitoring system to ensure the supervision of a large number of energy zones.

Approbation of results. The main provisions and results of the research presented in the project were tested at the XVIII International Scientific and Technical Conference "Problems of Informatization" in December 2021.

The subject of research energy supply monitoring system.

The bachelor's qualification work consists of the content, introduction, main part, which includes four sections, conclusions and a list of sources used. Total 55 pages.

KEY WORDS: IoT, Database, Energy, Monitoring, Microsoft SQL Server, Application, Software Product, API, Angular, C#, HTML, TypeScript, CSS.

ЗМІСТ

ВСТУП

Електроенергія створила революцію в людському житті, тому складно заперечувати важливість сфери енергетики в сучасному світі. Прихід автоматизації в наше життя не обійшов стороною і її. Галузь енергетики, одна з тих, де автоматизація відіграє дуже велику роль. Адже дуже важливо, щоб електроенергія поступала до людей безперебійно і надійно, ось чому важливо зменшити процент фактора людської похибки. В цій сфері автоматизація поділяється на автоматизацію «виготовлення» енергії і автоматизацію моніторингу. Автоматизація нагляду за різними елементами енергетики допомагають легше та якісніше контролювати процеси енергетики. Саме тому проблема впровадження інформаційних систем для цієї галузі є актуальною.

Енергетична галузь це складний та об’ємний механізм. Починаючи добутком сировини, її обробкою, виготовленням, зберіганням, постачанням електроенергії до споживача, а також супутні елементи (наприклад опалення приміщень), закінчуючи зберіганням та постачанням енергії. Ланцюг усіх цих елементів утворює єдину систему, налагодження якої відбувалося протягом багатьох років і продовжується в наш час, ставлячи перед нами завдання вивести цю систему на принципово новий рівень завдяки правильній та чіткій інтеграції в неї засобів інформаційних систем та технологій.

Задачею даної бакалаврської роботи є дослідження готових рішень, проектування бази даних, створення повноцінного програмного продукту враховуючи серверну і клієнтську частину а також верстка фронтєнду.

В результаті планується отримати програмний продукт з базою даних призначений для спостереження за великою кількістю енергозон в загальному вигляді. Програмний продукт повинен представляти з себе односторінковий додаток в мінімалістичному дизайні для зручності операторів які спостерігають за зонами. Це дозволить використовуючи мінімальну кількість людської робочої сили здійснювати загальний моніторинг великої кількості енерголіній, зон, районі, областей. В залежності від бажань замовника буде можливість масштабувати додаток.

1. АНАЛІЗ ТА ОПИС СФЕРИ ЗАСТОСУВАННЯ БАЗИ ДАНИХ ІОТ СИСТЕМИ ЕНЕРГОПОСТАЧАННЯ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис об’єкту дослідження

Зменшення природних ресурсів і збільшення втрат енергії привертають увагу кількох країн у всьому світі. Смарт рішення користуються більшим попитом за останні кілька років завдяки їх підвищеній ефективності, що сприяє процвітанню галузей. Очікується, що до 2025 року IoT на енергетичному ринку досягне 39,09 мільярдів доларів США з зведеним річним темпом зростання в 17,24%[1].

Інтернет речей — це передова концепція, яка використовує сенсорні пристрої для моніторингу кімнатної температури, виконання складних функцій, контролю та точного запису споживання енергії, тим самим ефективно зменшуючи загальні витрати на технічне обслуговування.

Енергія є найважливішим ресурсом для всіх видів промисловості. Різні промислові галузі використовують енергію в різних формах. США відомі як високоіндустріальна країна, де хімічний сектор є одним із найбільших споживачів енергії, за ним слідують нафтопереробний і гірничодобувний сектори. Більшість галузей промисловості купують електроенергію у комунальних підприємств або виробників електроенергії. Наприклад, виробничі підрозділи повністю покладаються на природний газ, нафту, електроенергію, вугілля та біопаливо. Крім того, паперові фабрики використовують природний газ і чорний щелок для переробки тепла та виробництва електроенергії. Можна сказати, що промисловості в основному використовують викопне паливо та відновлювані джерела енергії для:

- виробництва пару або гарячої води для технологічного опалення;
- сировини для виробництва пластмас або хімікатів;
- тепла в промислових процесах;
- опалення приміщень в будівлях.

Таким чином, енергія є найкориснішим ресурсом як для промислових, так і для повсякденних цілей. Крім того, очікується, що промислові потреби в енергії зростуть на 31% протягом наступних 25 років, що становитиме близько 38% загального споживання енергії в США.

Інтернет речей займає значну частку в наданні унікальних рішень для оптимізації інформаційного потоку, моніторингу даних, використання прогнозного управління активами, зниження витрат, скорочення робочої сили та підвищення надійності до традиційних систем. Крім того, IoT пропонує технологічні досягнення в енергетичному секторі для підвищення продуктивності промисловості та покращеної працездатності. Наприклад, нафтові родовища, розташовані у віддалених місцях, містять ізольовані центри обробки даних, які потребують централізованої системи управління як хмара, для покращення обробки інформації в ланцюгах поставок.

Існуючі рішення для моніторингу енергії:

- Моніторинг енергії в режимі реального часу.
- Структури споживання.
- Майбутні потреби в електроенергії.
- Технічне обслуговування обладнання.
- Microgrid для оптимального споживання електроенергії.

Говорячи про застосування розумної системи моніторингу енергії, IoT є найбільш улюбленою технологією сьогодні. Інноваційні концепції та розумні методи ведення обліку споживання енергії в режимі реального часу роблять IoT найбільш прийнятною та застосовуваною концепцією.

1.2 Готові рішення

1.2.1. EcoStruxure™ Power Monitoring Expert

Програмне забезпечення EcoStruxure™ Power Monitoring Expert – це повноцінна система енергоменеджменту, яка агрегує дані про розподільну мережу Вашого підприємства та представляє їх як зрозумілу інформацію через інтуїтивний веб-інтерфейс. Відкрита архітектура EcoStruxure Power Monitoring

Expert використовує стандартні промислові протоколи та дозволяє працювати з будь-якими пристроями як Schneider Electric, так і сторонніх виробників. Програмне забезпечення легко інтегрується з будь-якими системами обліку, моніторингу та автоматизації (наприклад, SCADA, BAC, DCS, ERP) та веб-сервісами (рис. 1.1.).[2]

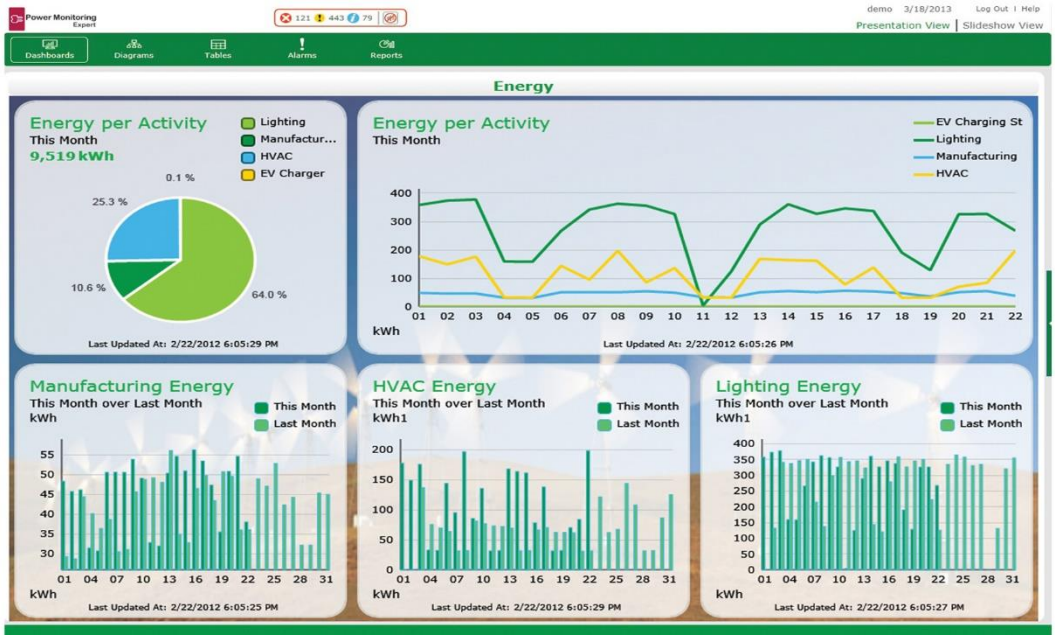


Рисунок 1.1 – Веб-додаток EcoStruxure™ Power Monitoring Expert

1.2.2. Energy Controller 2

Програма Energy Controller 2 призначена для моніторингу параметрів Джерел Безперебійного Живлення (ДБЖ) та управління функціями цих джерел. Спочатку Energy Controller створювався для взаємодії з ДБЖ IPPON Back Power Pro 400, але пізніше з'ясувалося, що програма нормально працює і з усіма іншими моделями ДБЖ IPPON. А в Energy Controller 2 закладена підтримка ще й ДБЖ Mustek, і деяких моделей ДБЖ Sven та Inelt. Інтерфейс програми зображений на (рис. 1.2.).[3]

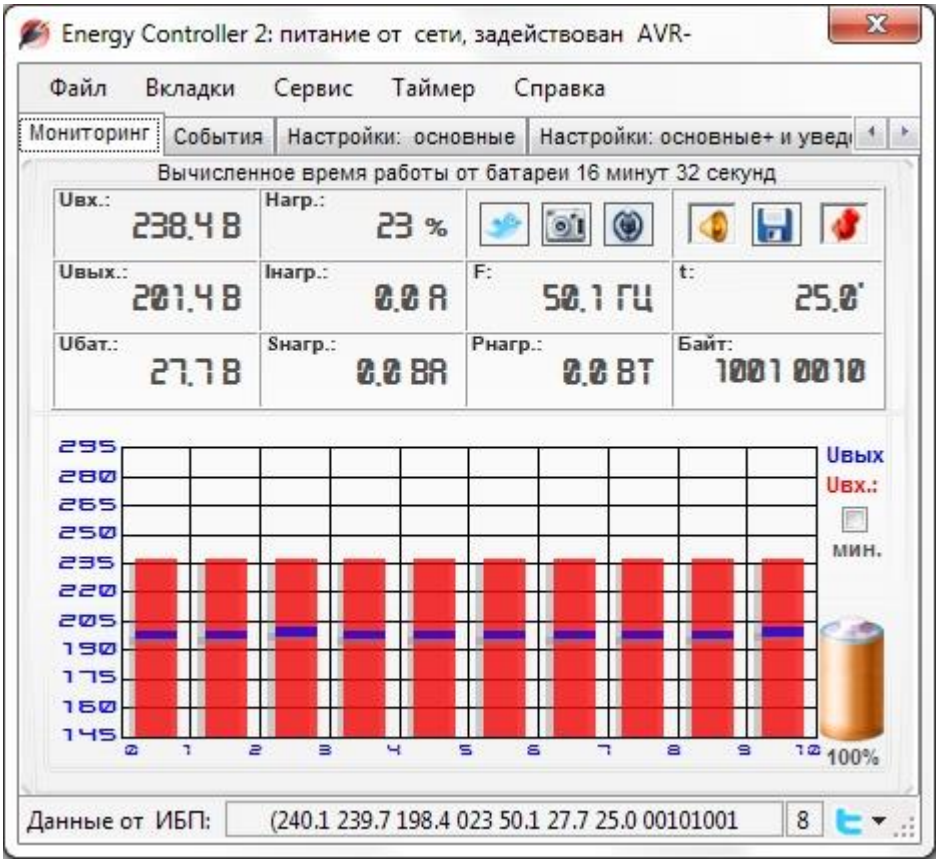


Рисунок 1.2 – Додаток Energy Controller 2

1.2.3. Solar-Log 1200

Реєстратори даних Solare Datensysteme Solar-Log 1200 є професійними пристроями для оптимізації віддаленого моніторингу систем із сонячними батареями. Поєднання сонячних інверторів із пристроєм Solar-Log гарантує повний контроль вашої фотоелектричної системи: всі операційні параметри, статус та повідомлення про помилки підключених силових перетворювачів можна розглядати просто та надійно. Пристрій зображено на (рис. 1.3.). [4]

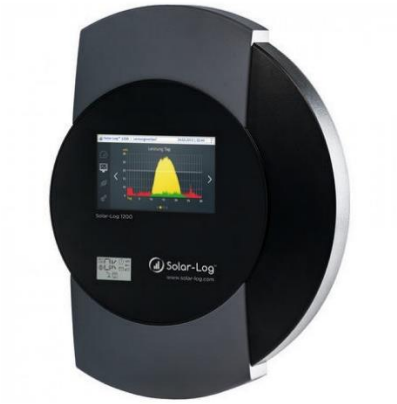


Рис 1.3 – Головний модуль Solar-Log 1200

1.2.4. AggreGate Network Manager для АТ "Об'єднана енергетична компанія"

Глибинний моніторинг та діагностика промислових ДБЖ виробництва APC та PowerCom, модульних ДБЖ FlatPack2 виробництва Eltek, а також інверторів TSI Bravo.

Найважливішими вимогами до роботи системи були висока надійність та безпека комунікаційної мережі. Компанія Юнітел Інжиніринг має комплексну територіально-розподілену систему комунікацій, що включає менеджмент кабельних систем, каналоутворювальне обладнання, безперебійне електропостачання і т.д. Інтерфейс зображений на (рис. 1.4.). [5]

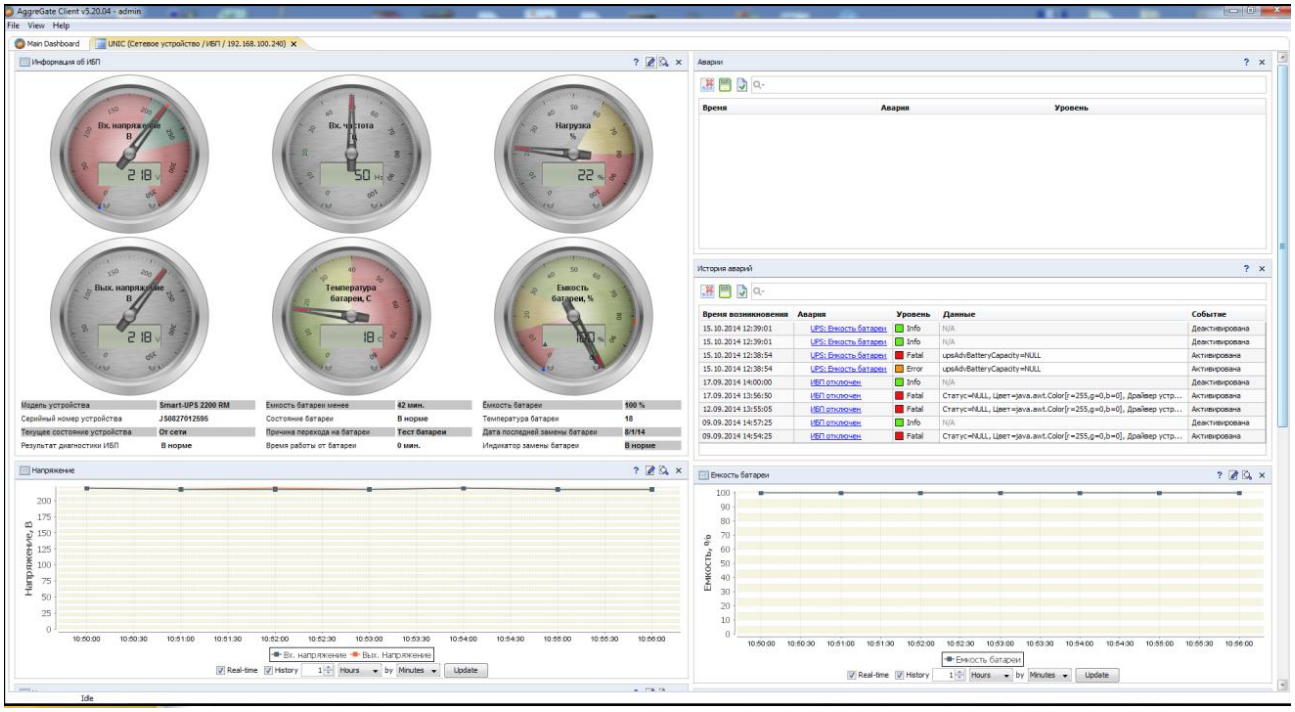


Рисунок 1.4 – Вікно моніторингу AggreGate Network Manager

1.2.5. AggreGate Network Manager для Flexenclosure

Створення на базі AggreGate SCADA/HMI референсного рішення щодо управління системами енергопостачання базових станцій стільникової мережі. Багато впроваджень у мережах міжнародних операторів зв'язку (Airtel, Telenor, Etisalat, та інші).

AggreGate Network Manager координує всі дані які відносяться до електропостачанню на одному простому інтерфейсі. За допомогою багаточисельних опцій генерації звітів, система дозволяє відображати всі характеристики мережі (від проектування і будівництва до матеріально-технічного забезпечення станцій і системи контролю). Інтерфейс програми зображений на (рис 1.5.). [6]

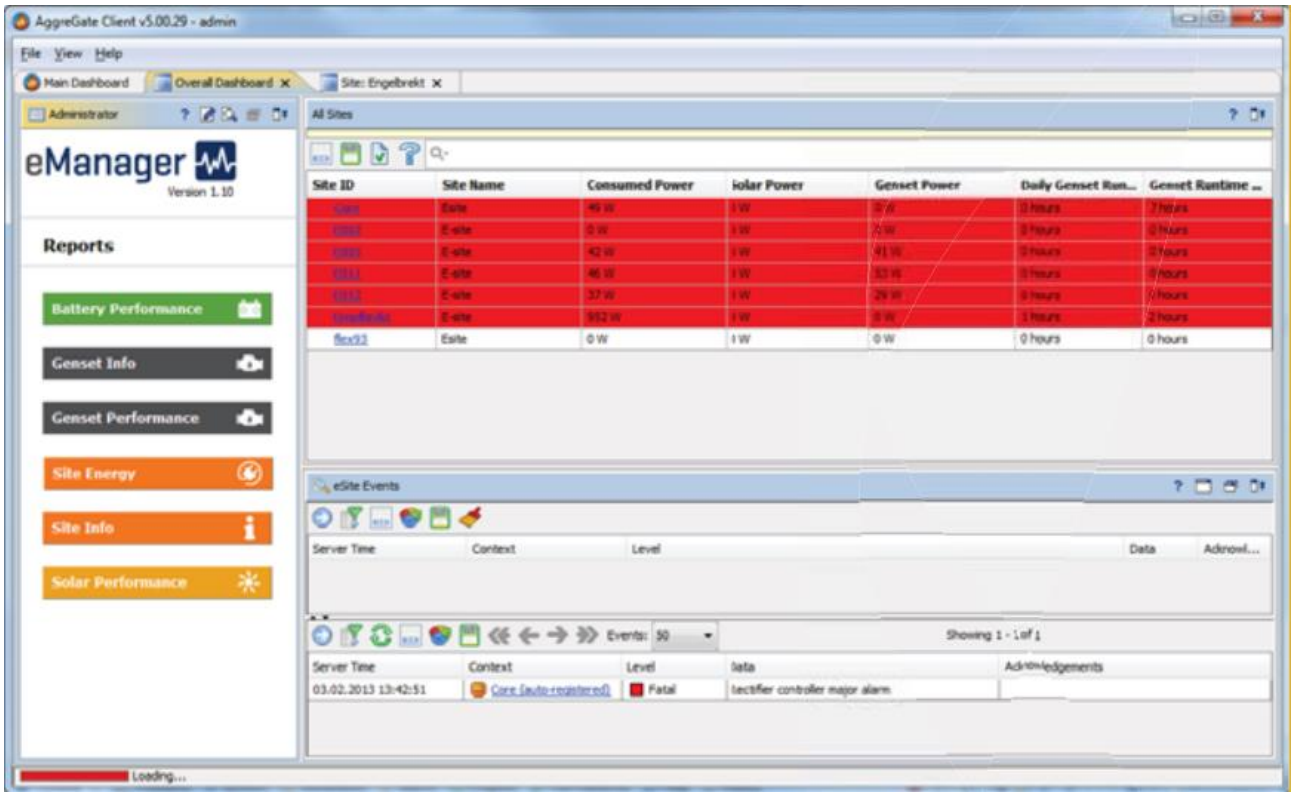


Рисунок 1.5 – Вікно моніторингу AggreGate Network Manager

1.3 Порівняння рішень

Для простішого порівняння створено таблицю з основними характеристиками розглянутих варіантів.

Таблиця 1.1 - Порівняння існуючих рішень

Критерій	**EcoStruxure™ Power Monitoring Expert**	**Energy Controller 2**	**Solar-Log 1200**	**AggreGate Network Manager**
1	**2**	**3**	**4**	**5**
Призначення	Енергоменеджмент розподілу мережі на підприємстві	Моніторинг параметрів Джерел Безперебійного Живлення	Віддалений моніторинг систем із сонячними батареями	Глибинний моніторинг діагностика та аналіз енергомережі. Налаштовується під замовника

Продовження Таблиці 2.1

Критерій	**EcoStruxure™ Power Monitoring Expert**	**Energy Controller 2**	**Solar-Log 1200**	**AggreGate Network Manager**
1	**2**	**3**	**4**	**5**
Архітектура	Клієнт-сервер	Декстоп-клієнт	Хмара	Клієнт-сервер, хмара
Мобільний додаток	Так	Ні	Ні	Ні
Рівні доступу	• Оператор • Адміністратор	• Користувач	• Користувач	• Настроюється
Сповіщення	• До мобільного додатку • Пошта	• Через ОС	• Звуковий • На сайті • Пошта	• Пошта • На сайті
Особливості	• Індивідуальне налаштування відображення • Можливість підключати різні модулі • Персональне обслуговування	• Низкі вимоги до тех. Характеристик • Мінімізований інерфейс	• Компактність • Доступ в любої точки світу	• Налаштовується під потреби замовника • Надзвичайно велика функціональність • Висока ціна
Складність	Висока	Низька	Низька	Дуже висока
Функціональність	Висока	Низька	Низька	Дуже висока
Ціна	~ 236 292₴	Бесплатно	~ 37 922 ₴	~ 70 887 - 744 321₴ Договірна

Згідно цієї таблиці система яка буде розроблятися в цій бакалаврській роботі матиме такі критерії:

Таблиця 1.2 – Критерії розроблюваного програмного продукту

Критерій	**EcoStruxure™ Power Monitoring Expert**
1	**2**
Призначення	Загальний моніторинг великої кількості енергозон
Архітектура	Клієнт-сервер
Мобільний додаток	Ні
Рівні доступу	• Оператор
Сповіщення	• На сайті
Особливості	• Відображення великої кількості зон • Масштабованість • Мінімалізм
Складність	Дуже низька
Функціональність	Низька
Ціна	~ 600-1700₴

На мою думку, такі критерії найкраще підходять для планованого проекту. Низька ціна і складність у встановлені і використанні дозволить програмному продукту бути загальнодоступним. Хоча функціональність і низька, але вона виконує все що потрібно для своїх цілей, виходить вузька спеціалізація у загальному моніторингу за маленьку ціну. Це підійде тим, хто не хоче переплачувати за функціонал який йде попутно в подібних додатках, але яким не планується користуватись.

1.4 Висновок

У даному розділі описано об’єкт дослідження та перспективи розвитку технологій IoT у сфері енергетики. Також розглянуто існуючі рішення та зведено їх критерії до єдиної порівняльної таблиці. Також були наведені критерії які отримає програмний продукт згідно порівняльної таблиці і зроблені допущення щодо корисності і переваги створюваного програмного продукту.

2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ СИСТЕМИ МОНІТОРИНГУ ЕНЕРГОПОСТАЧАННЯ

2.1 Створення концептуальної моделі бази даних

Опис предметного середовища в термінах деякої моделі називається концептуальною схемою або логічною схемою бази даних. Перед побудовою концептуальної моделі необхідно створити функціональну модель роботи системи, тобто у нашому випадку – системи моніторингу енергопостачання . Для моделювання бізнес-процесів використовується нотація BPMN, що дозволяє зобразити ці процеси у вигляді діаграм, що значно полегшує їх сприйняття та подальший аналіз для виділення сутностей, що беруть участь у конкретному процесі.

На (рис. 2.1) зображено функціональну модель системи моніторингу енергопостачання в нотації BPMN для відстежування стану енерголіній, що є ключовим для створюваної інформаційної системи.

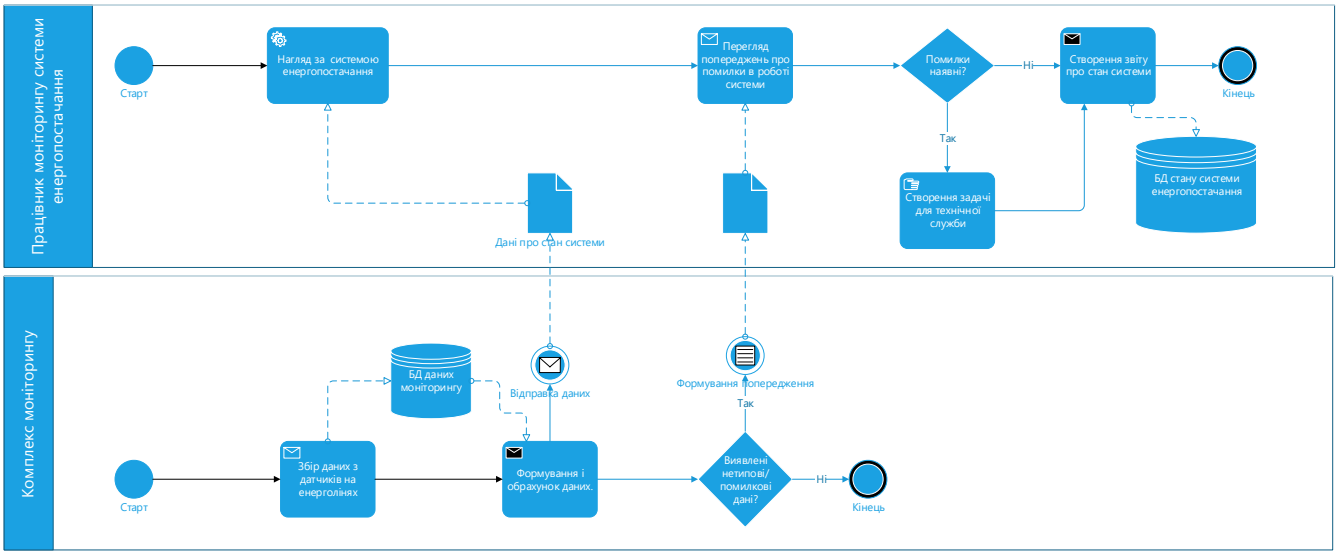


Рисунок 2.1. Функціональна модель процесу системи моніторингу енергопостачання у нотації BPMN

Система складається з двох суб’єктів, сам комплекс моніторингу, та особі, яка наглядає за даними.

Комплекс збирає інформацію з датчиків на електролініях, та додає їх до внутрішньої БД, для збереження. Потім всередині системи дані оброблюються в корисну і зрозумілу інформацію, та надаються користувачу.

Також система аналізує інформації, та на її основі надає попередження оператору, що до стану енерголіній. Наприклад, на основі натягу або провисання енерголіній, попереджує про утворення льоду на них.

Оператор отримує інформацію від системи, та аналізує її, наприклад на випадок, якщо якісь сценарії не передбачувані в програмі. Також він наглядає, за станом енерголіній, і орієнтуючись на попередженнях, посилає запит технічній команді для ремонту на проблемних ділянках.

Наступним кроком перед створенням концептуальної моделі є розробка ділової моделі для відповідного процесу. Ділова модель дозволяє виділити так звані класи досліджуваної моделі, їх функції та взаємодію між ними.

Ділова модель процесу моніторингу енергопостачання зображена на (рис. 2.2). Із рисунку видно, що було виділено 4 класів та 4 функції, за якими вони взаємодіють між собою. У зборі даних беруть участь сенсори та система моніторингу. Аналіз проводиться в два етапи, первинний автоматичний аналіз системою моніторингу, та вторинний ручний аналіз на невідповідності, оператором. Реагують, на помилки в роботі енерголіній, система, яка створює, та передає попередження оператору. Сумує зібрану інформацію за день оператор, та зберігає її у звітах, для архіву.

Класи / **Функції**	**Оператор**	**Система моніторингу**	**Попередження**	**Сенсори**
Збір даних		*		*
Аналіз даних	*	*		
Реагування на помилки	*	*	*	
Сумування інформації	*			

Рисунок 2.2. Ділова модель процесу системи моніторингу енергопостачання

На основі наведеної вище ділової моделі було створено концептуальна модель майбутньої бази даних (рис. 2.3).

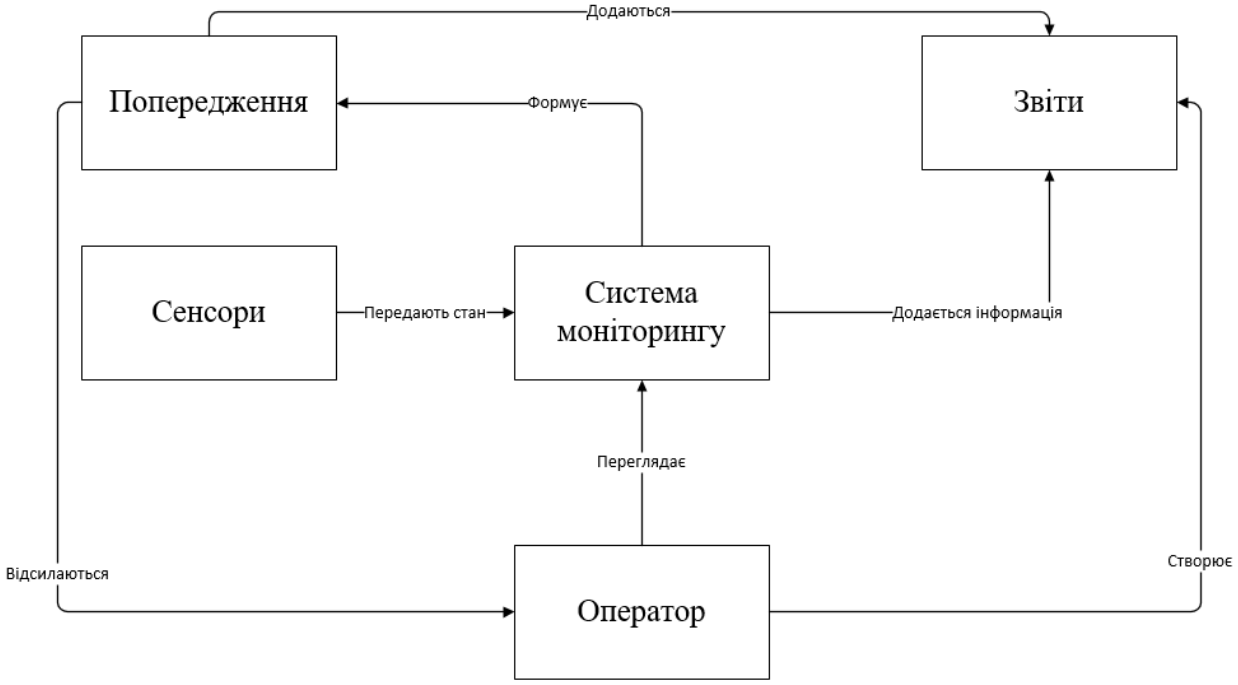


Рисунок 2.3. Концептуальна схема бази даних системи моніторингу енергопостачання

Із рисунку видно, що є два основних суб'єкта, Оператор, та Система моніторингу. Сенсори передають дані в систему, яка їх оброблює, та надає для перегляду чи/та редагуванню оператору. На основі цієї інформації оператор заповнює звіти. Також, якщо система бачить якісь невідповідності даних, нормі, то вона попереджує про це оператора..

На основі отриманої концептуальної схеми було розроблено проект таблиць, що буде містити майбутня база даних. У таблиці 1.2 представлено 5 діючих таблиць та 2 технічні, назви їх полів та тип даних, якому ці поля відповідають.

Таблиця 2.1 Проект таблиць бази даних системи моніторингу енергопостачання

Таблиця	Ім’я поля	Тип даних
1	**2**	**3**
Operator	operator_ID	INT
	operator_name	VARCHAR
	operator_status	VARCHAR
	hiring_date	DATE
	release_date	DATE
	operator_photo	VARCHAR
Zone	zone_ID	INT
	zone_name	VARCHAR
	place	VARCHAR
Sensors	sensor_ID	INT
	sensor_name	VARCHAR
	sensor_cost	INT
	sensor_type	VARCHAR
	installation_date	DATE
	zone_ID	INT
Sensor_Log	sensor_log_ID	INT
	value	INT
	date_sensor_log	DATE
	sensor_ID	INT
	sensor_status_ID	INT

Продовження Таблиці 2.1

Таблиця	Ім’я поля	Тип даних
1	2	3
Zone_Log	zone_log_ID	INT
	date_zone_log	DATE
	zone_ID	INT
	zone_status_ID	INT
	operator_ID	INT
Sensor_Status	sensor_status_ID	INT
	sensor_status	VARCHAR
Zone_Status	zone_status_ID	INT
	zone_status	VARCHAR

2.2 Створення контекстної моделі бази даних

Контекстна діаграма на (рис. 2.4) наведено контекстну діаграму процесу моніторингу енергопостачання. Основним процесом для моніторингу енергопостачання є забезпечення стабільності енергопостачання. Тому саме цей процес буде знаходитися на вершині дерева і описується у нотації IDEF0.

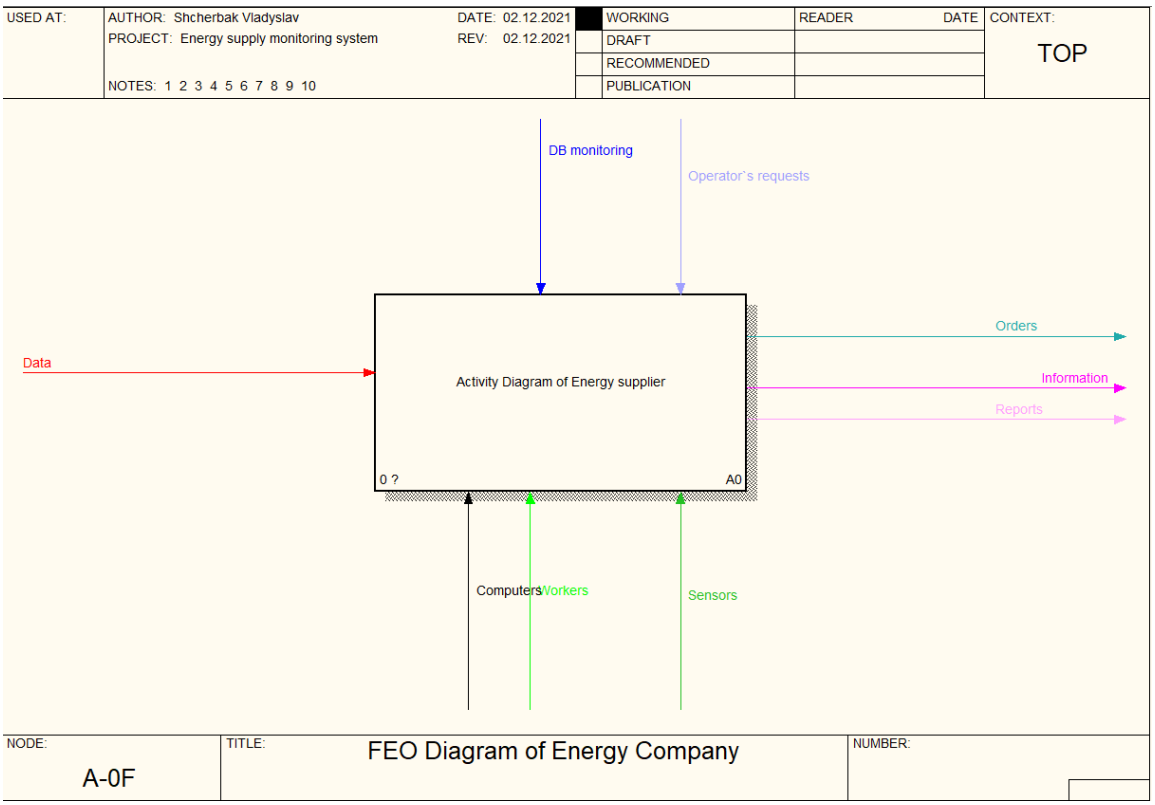


Рисунок 2.4 – Контекстна діаграма процесу моніторингу енергопостачання в нотації IDEF0

Із рисунку видно, що на вхід надходять дані зняті датчиками за енерголіній. У якості елементів керування виступають БД моніторингу, та запити Оператора. Механізмами, що виконують процес є комп'ютери, на яких стоїть система і за якими працюють оператори, сенсори які знімають показники з енерголіній, та самі працівники які працюють з системою. е. Як результат, на виході отримуємо опрацьовану інформацію, звіти, та накази для технічних служб.

Для можливості більш детального розгляду основного процесу було створено діаграму декомпозиції для цього процесу (рис. 2.5). Таким чином, основний процес було поділено на два підпроцеси: комплекс моніторингу та процеси оператора. Зв'язок між ними полягає у тому, що після обробки даних комплексом моніторингу, на вхід оператору передається уже цінна інформація.

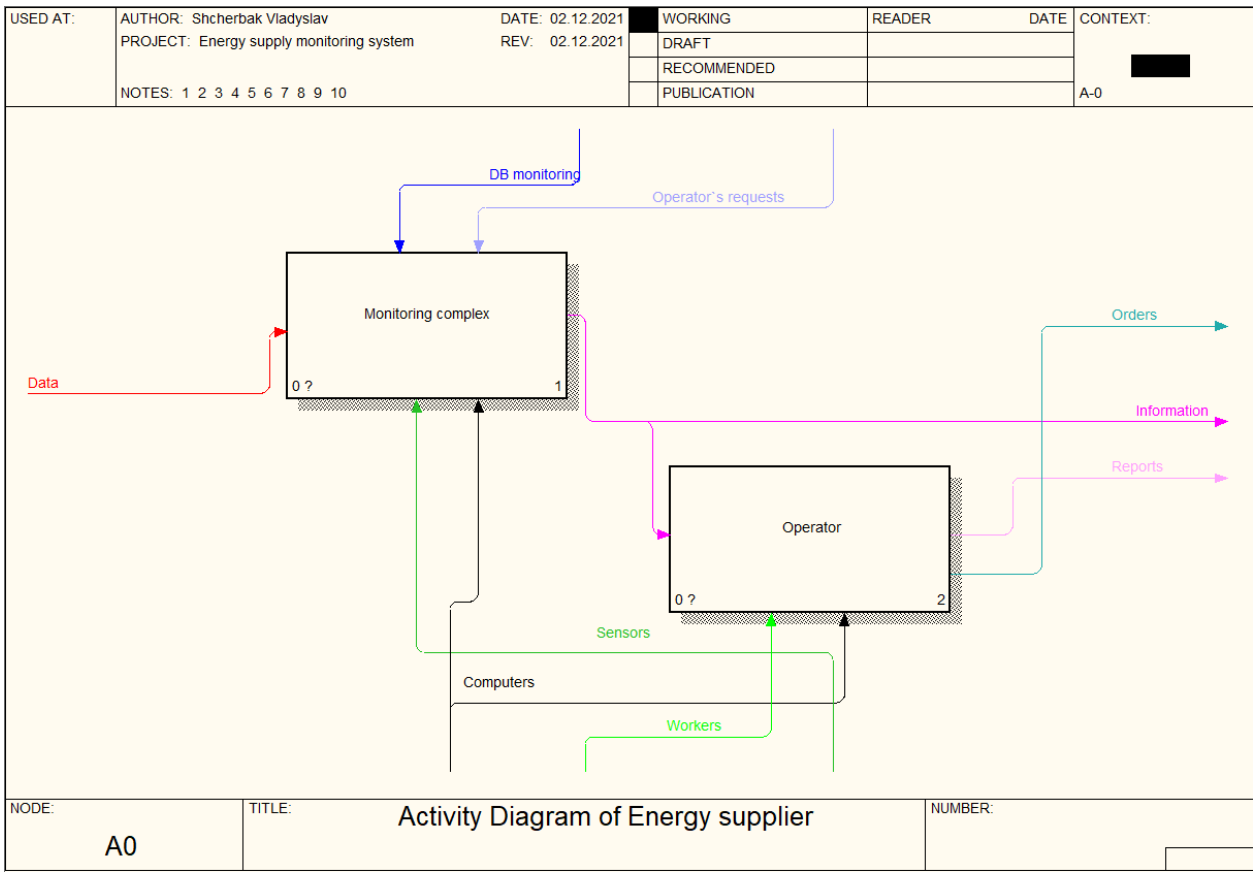


Рисунок 2.5 – Діаграма декомпозиції основного процесу

Для більш глибоко вивчення роботи системи для процесу комплекса моніторингу також було створено діаграму декомпозиції (рис. 2.6), а для обробки даних побудовано діаграму нижчого рівня у нотації IDEF3 (рис. 2.7).

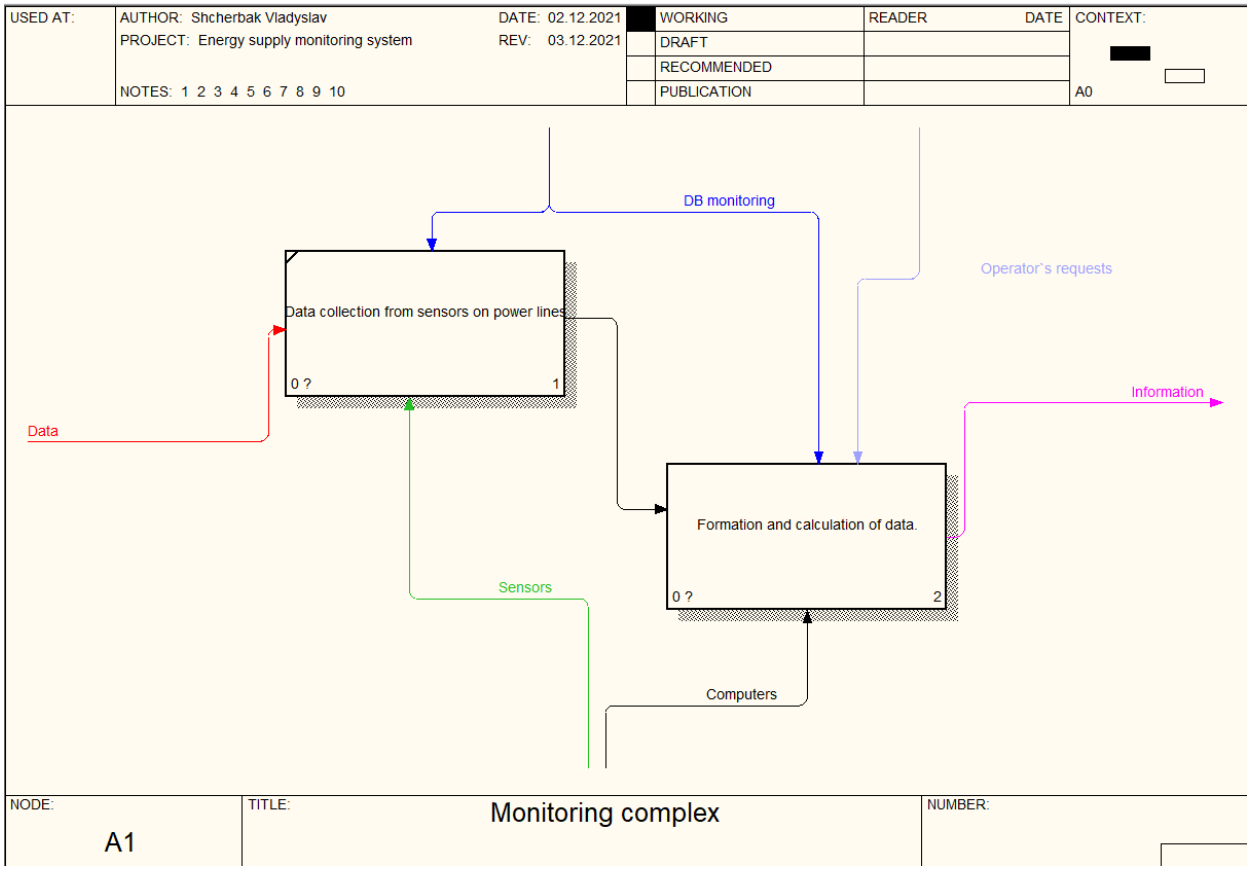


Рисунок 2.6 – Діаграма декомпозиції підпроцесу комплекс моніторингу

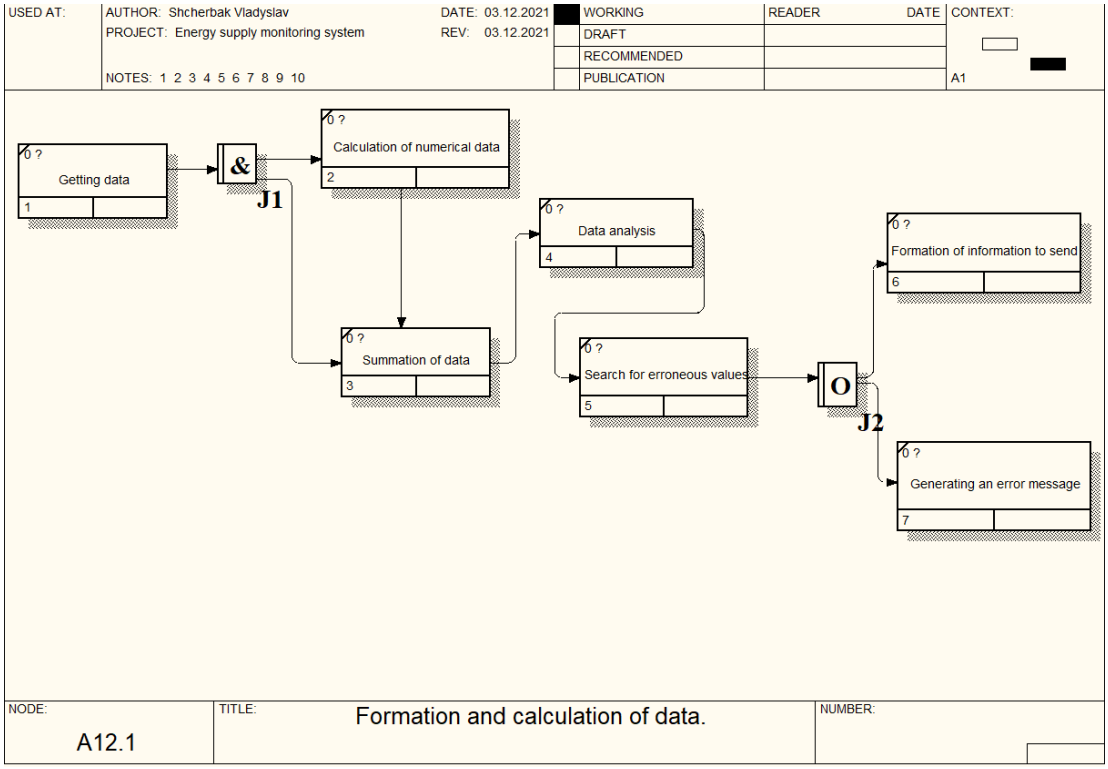


Рисунок 2.7 – Процес обробки даних у нотації IDEF3

Із (рис. 2.6) видно, що комплекс моніторингу також поділений на збір показників з енерголіній та підрахунок та обробку даних. Після процесу збору даних, вони передаються на обробку, де вони категоруються і опрацьовується. На виході ми отримуємо інформацію, яка вже передається далі.

Згідно з (рис. 2.7), процес обробки даних полягає в тому, що отримавши дані, поділяються на різні типи. А самі, дані логічні, та цифрові, які вже підраховує система. Потім дані аналізуються, на наявність нетипових показників. Залежно від цього аналізу, система передає цю інформацію, чи створює попередження для оператора, щоб він проаналізував цю інформацію та відреагував на неї.

У результаті виявлення основного процесу, декомпозиції його та підпроцесів було отримано дерево вузлів (Додаток Г).

Для опису процесу збору даних з сенсорів та їх зберігання і передачу використовуються діаграми потоків даних (DFD). Оскільки одним із основних елементів для розроблюваної інформаційної системи є процес збору, зберігання та трансферу даних, то саме його представлено у вигляді такої діаграми на (рис. 2.8).

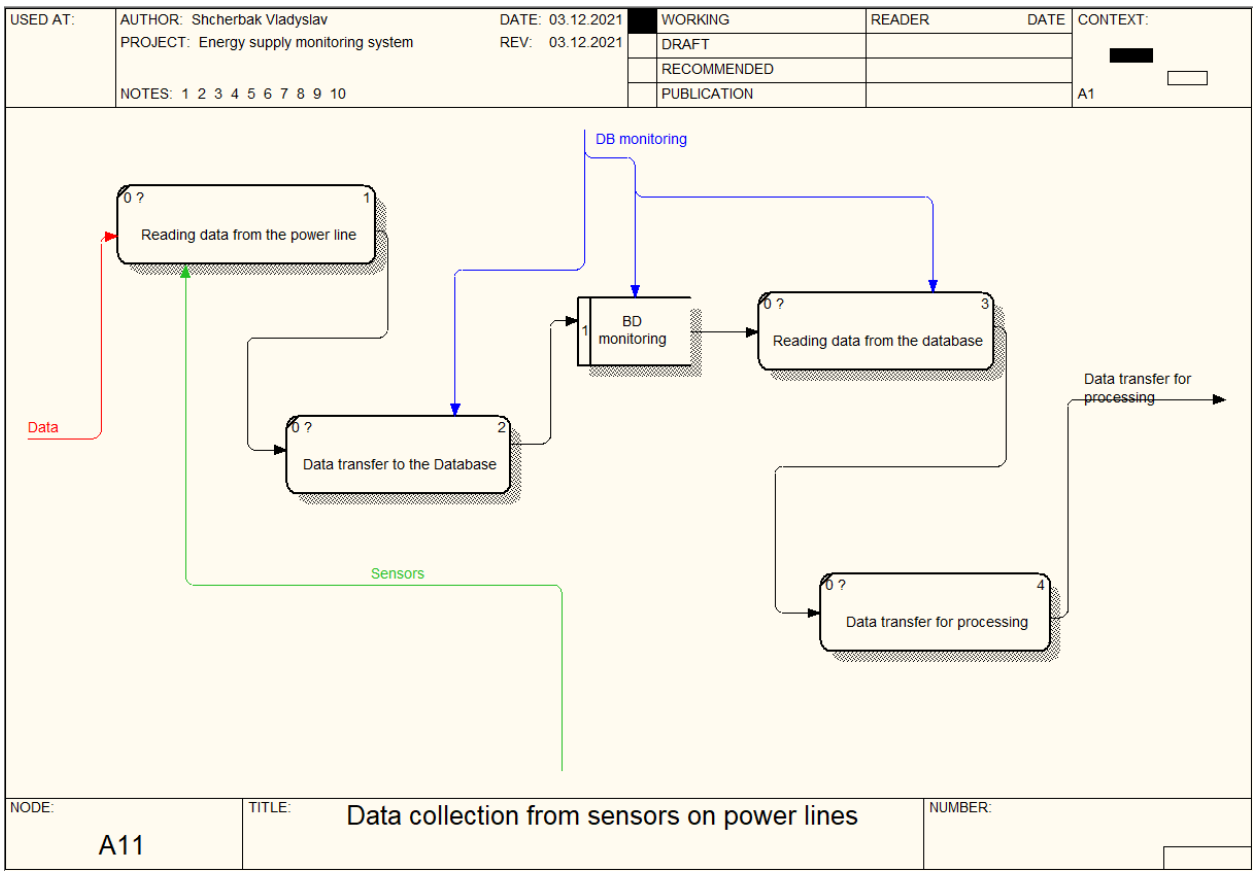


Рисунок 2.8 – Представлення процесу збору даних в нотації DFD

Як ми можемо спостерігати, процес є доволі лінійним. Первинні дані збираються сенсорами с енерголіній автоматично. Далі вони передаються до внутрішньої БД, для зберігання і використовування. Потім по запиту системи, ці дані передаються на обробку.

2.3 Створення логічної моделі бази даних

Наступним кроком після створення контекстної моделі бази даних є розробка її логічної моделі, тобто такої, що описує структуру бази даних незалежно від системи керування чи технологій збереження. Оскільки проект таблиць із полями та їх типами даних було описано у першому підрозділі (табл. 2.1), для побудови логічної моделі залишається тільки визначити типи зв’язків між ними.

Для кращого розуміння випадків функціонування системи у навколишньому середовищі було побудовано діаграму прецедентів (використання) для основних учасників системи моніторингу енергопостачання (рис. 2.9).

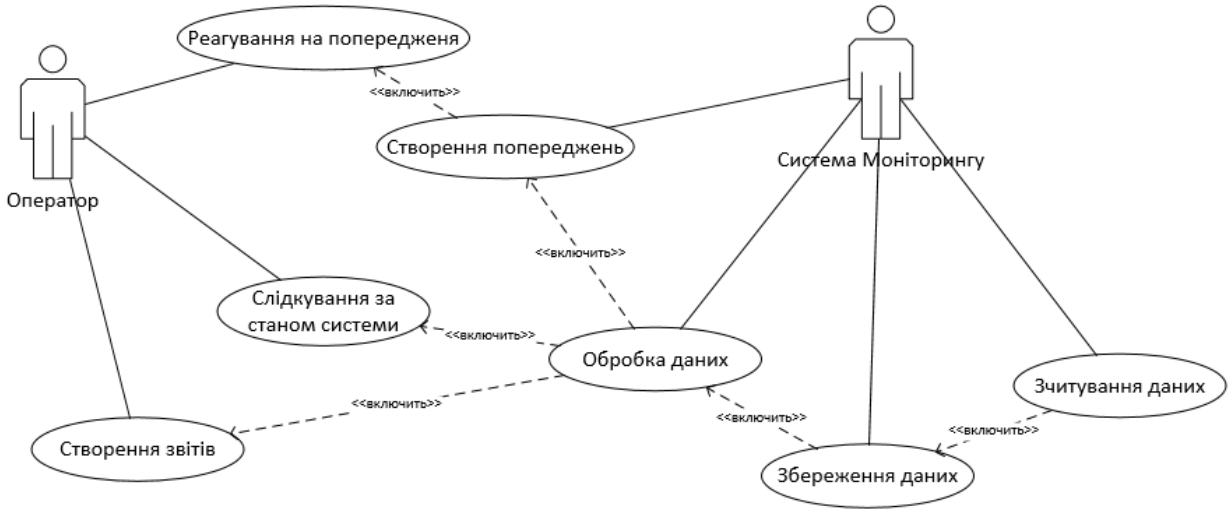


Рисунок 2.9 – Діаграма прецедентів системи моніторингу енергопостачання

Діаграма прецедентів демонструє, що процес починається зі зчитування даних. Які зберігаються, а потім оброблюються. Все це виконується системою моніторингу. Також система аналізуючи оброблену інформацію, створює попередження для оператора в разі необхідності.

Сам же оператор в основному слідкує за станом системи, та реагує на отримані попередження. А на основі опрацьованої інформації від системи моніторингу, оператор створює звіти для архіву.

Для впорядкування описаних вище взаємодій за часом їх прояву використовується діаграма діяльності, наведена на (рис. 2.10).

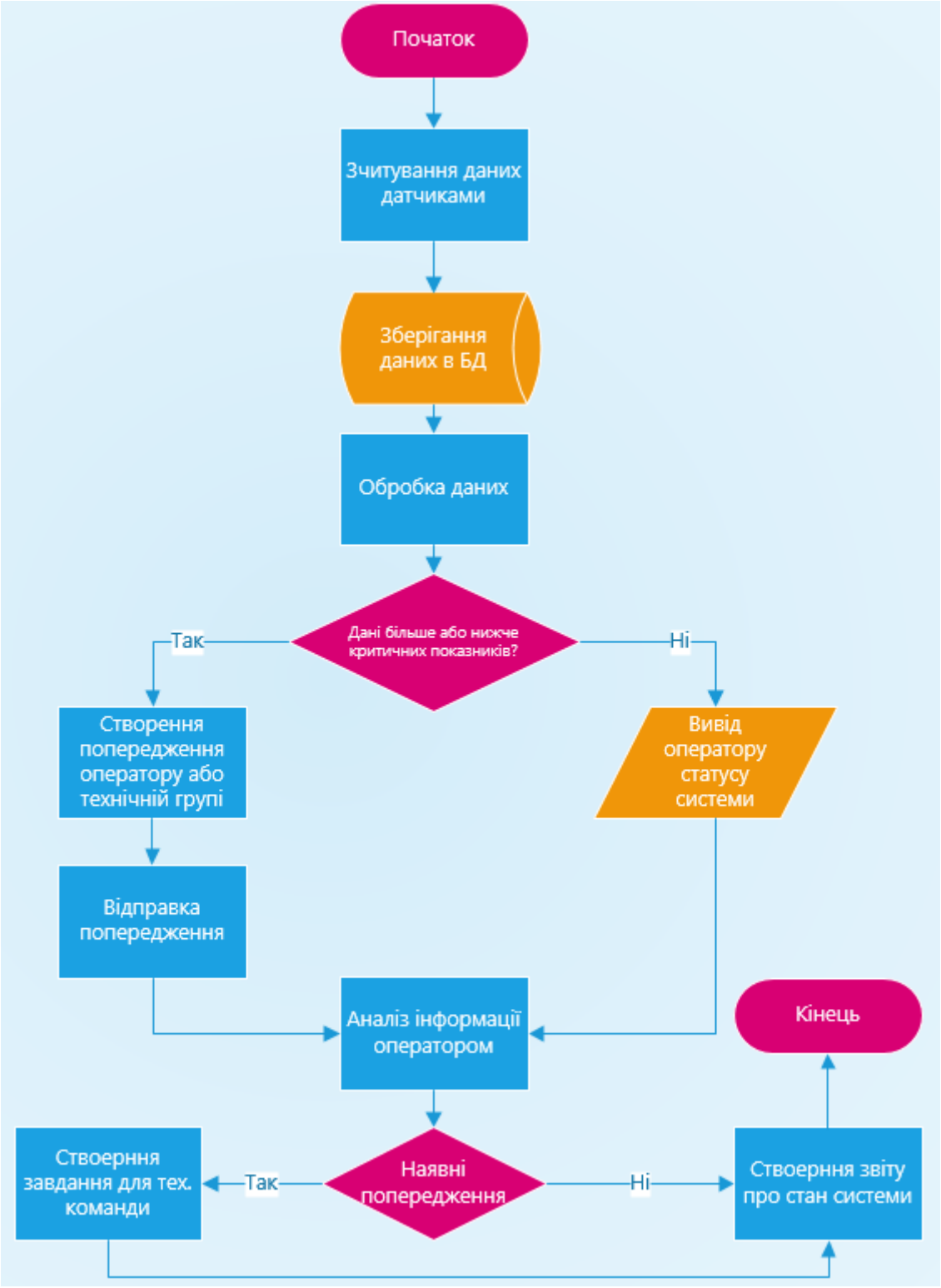


Рисунок 2.10 – Діаграма діяльності системи моніторингу енергопостачання

Останнім кроком перед побудовою логічної моделі бази даних є створення діаграми послідовності, яка визначає змінні, що враховує кожен процес та інформацію, що повертається у результаті його виконання. Так із діаграми на (рис. 2.11) видно, що спостереження за станом енерголіній

відбувається за рахунок отриманої інформації від системи. Для цього вона передає запит в БД, до якої дійшли дані від сенсорів, після, вона опрацьовує дані, та зберігає інформацію в БД, а також передає цю інформацію до оператора. Оператор може і сам створювати запити на отримання позачергової інформацію до системи. Система на цей запит звертається до БД, збирає інформацію та виконує запит оператора.

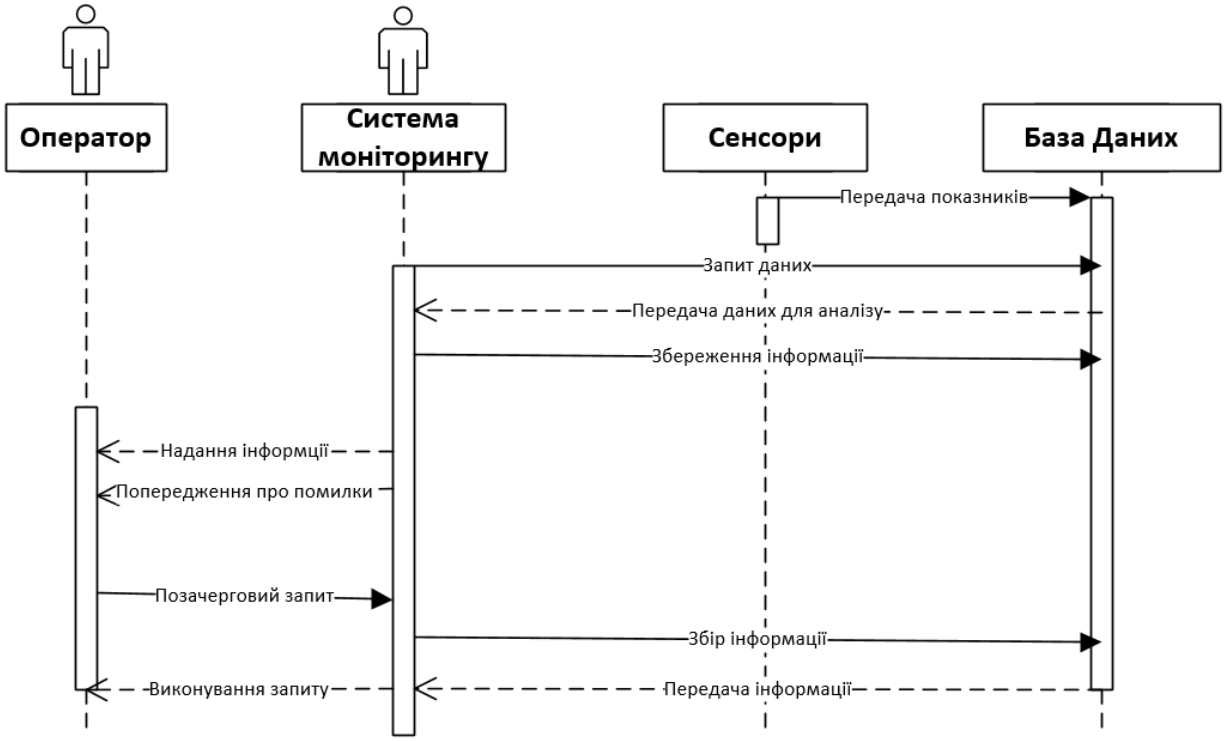


Рисунок 2.11 – Діаграма послідовності спостереження за станом енергопостачання

У результаті було отримано логічну модель, зображену на (рис. 2.12) з усіма таблицями та полями, вказаними у таблиці 2.1.

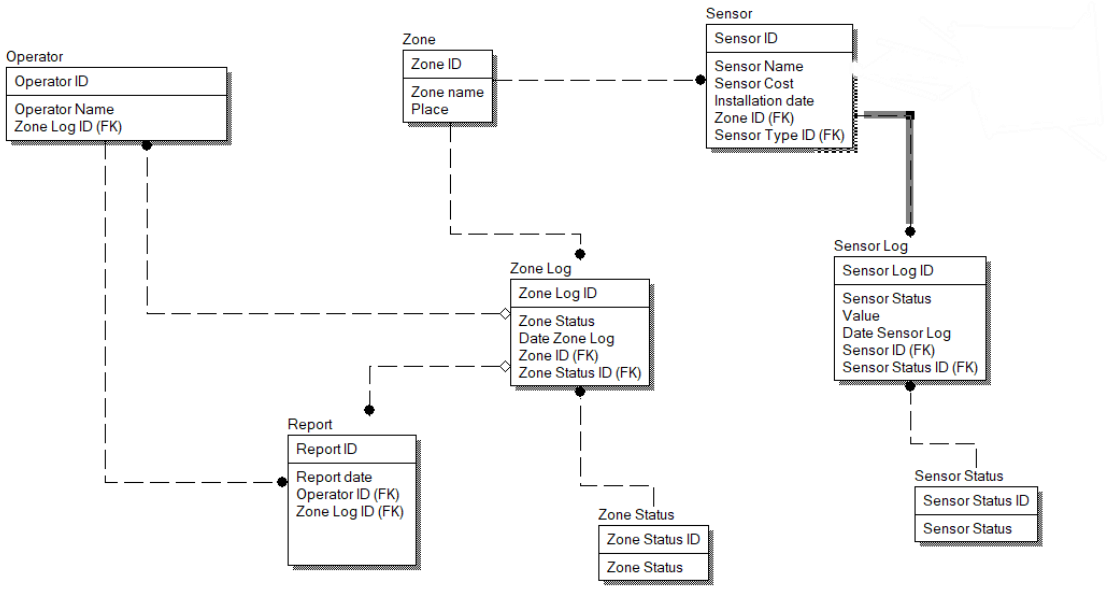


Рисунок 2.12 – Логічна модель бази даних системи віддаленого моніторингу стану пацієнта у середовищі Erwin Data Modeler

2.4 Створення фізичної моделі бази даних

Фізична модель даних описує, як дані насправді зберігаються в базі даних. Вона містить специфікацію всіх таблиць та стовпців всередині них. Специфікація таблиці включає назву таблиці та номер стовпців, а специфікація стовпців включає ім’я та тип даних. Фізична модель даних також містить первинні ключі кожної таблиці, і показує взаємозв’язок між таблицями за допомогою зовнішніх ключів.

Головною відмінністю фізичної моделі від логічної полягає в тому, що фізична модель може включати такий тип сутності як представлення, тобто тимчасова таблиця, що містить у собі значення вже існуючих таблиць.

Створити тимчасову таблицю у пакеті Erwin Data Modeler можна обравши окремо кожен компонент SQL-запиту а самостійно задати необхідний запит. У даному випадку (рис. 2.13) використовується користувацький запит, який створює представлення, що буде містити середні показники різних типів. У майбутньому таке представлення значно полегшить розробку програмного додатку.

```
CREATE VIEW dbo.Report_on_subordinate_Zones
(operator_name,zone_name,AVG_Value)
AS
SELECT Operator.operator_name,Zone.zone_name,AVG(Sensor_Log.value)
        FROM Operator,Zone,Sensor_Log
go
```

Рисунок 2.13 – Створення тимчасової таблиці (представлення) середніх значень показників у середовищі Erwin Data Modeler

Окрім створення представлень середовище Erwin Data Modeler автоматично створює фізичну модель бази даних на основі логічної. Тому на (рис. 2.14) зображено готову фізичну модель бази даних системи моніторингу енергопостачання зі всіма первинними та зовнішніми ключами, типами даних полів таблиць, тимчасовими таблицями та зв’язками.

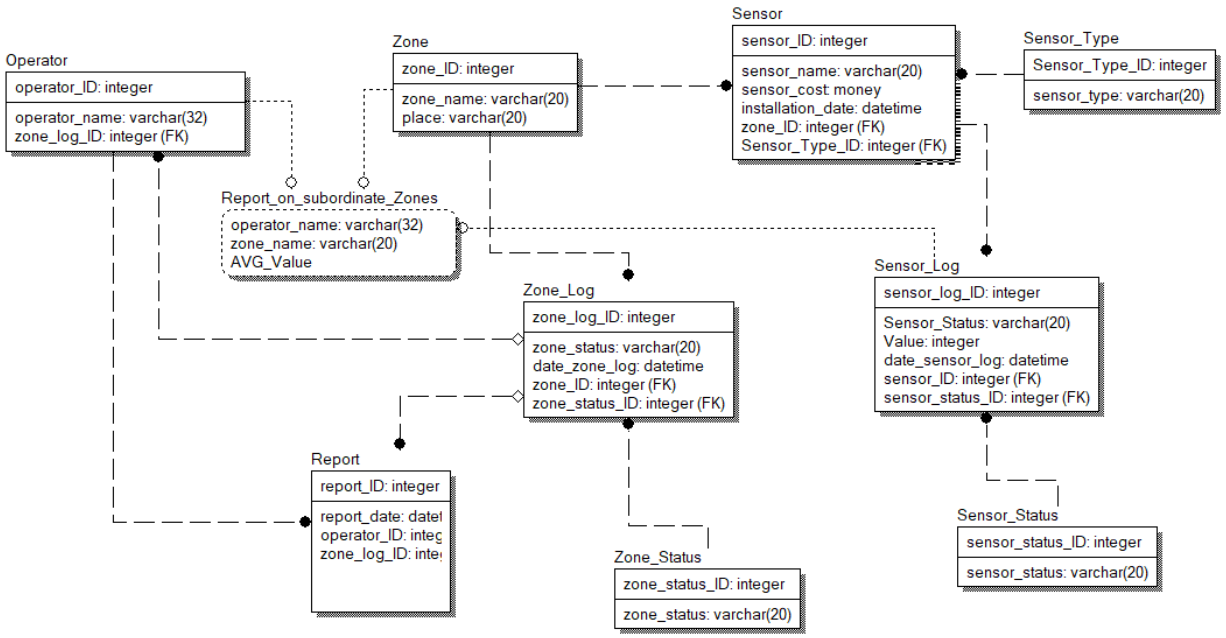


Рисунок 2.14 – Фізична модель бази даних системи віддаленого моніторингу енергопостачання у середовищі Erwin Data Modeler

2.5 Висновок

У даному розділі було створено концептуальну схему майбутньої бази даних та на її основі описано таблиці, що будуть до неї входити. Також було створено та описано контекстну модель бази даних з усіма її компонентами. На основі контекстної моделі було розглянуто діяльність енергетичної компанії, пов’язану з системою моніторингу енергопостачання за допомогою діаграм прецедентів, діяльності та послідовності. У середовищі програмного пакету Erwin Data Modeler було створено логічну та фізичну моделі майбутньої бази даних, які у наступному розділі будуть використані для створення її у системі управління базами даних Microsoft SQL Server.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ ЕНЕРГОПОСТАЧАНЯ

3.1 Опис створення бази даних

Для створення та управління базою даних було використано СУБД Microsoft SQL Server Management Studio 18 (далі – SQL Server). Така версія програми була обрана через, те, що 2008-а версія має критичні баги які роблять неможливу подальшу розробку. Хоча 2008-а версія має найкращу сумісність з Erwin Data Modeler, у якому було створено модель бази даних. Для запитів використовується спеціально розроблена компанією мова Transact-SQL.

Розроблена у попередньому розділі модель бази даних була перенесена до SQL Server за допомогою вбудованого інструменту Forward Engineer, що використовується для генерації фізичної схеми бази даних з її фізичної моделі[7]. Цей інструмент автоматично створює SQL-скрипт (додаток А) та передає його для виконання у СУБД. В моєму випадку цей скрип потрібно було переносити вручну.

У результаті в середовищі SQL Server було створено базу даних, діаграму якої зображено на (рис. 3.1).

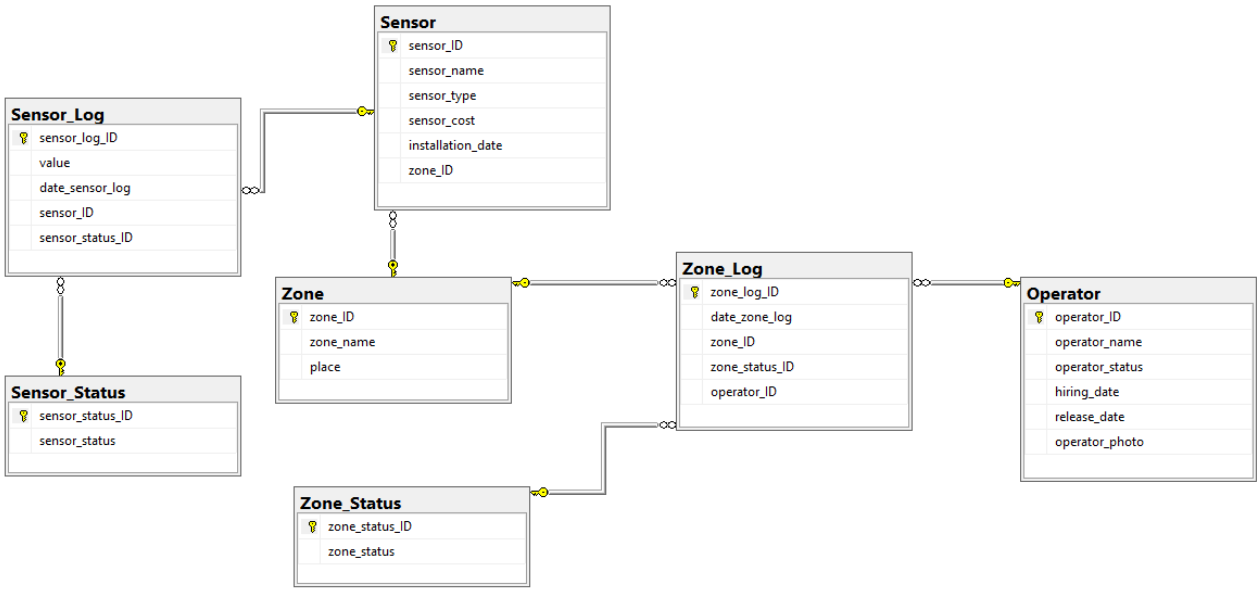


Рисунок 3.1 – Діаграма бази даних в середовищі Server Management Studio 18

3.2 Опис засобів реалізації

Для створення інтерфейсу програмного додатку було використано фреймворк для розробки веб-додатків Angular 13, розроблений компанією Google. Дана платформа використовує мову JavaScript і TypeScript та дозволяє створювати динамічні веб-сторінки зі складною бізнес-логікою. Основними перевагами Angular для задачі, що реалізовується, є[8]:

- Наявність інструментів розробника, доступних з командного рядка.
- Суворо типізований код.
- Наявність шаблонів для розширення HTML.
- Кросбраузерна підтримка HTTP, WebSockets, Service Workers.
- Підтримка великої кількості додаткових бібліотек.

Зв’язок між інтерфейсом додатку та базою даних здійснюється за допомогою протоколу HTTP. Для цього було додатково створено проект серверної API частини на платформі .NET[9] з використанням мови програмування C#. Такий інтерфейс дозволяє встановити зв'язок між базою даних та додатком за допомогою REST-запитів: GET, POST, PUT, DELETE – що відповідають операціям SELECT, INSERT, UPDATE та DELETE.

3.3 Опис реалізації серверної частини

Для написання коду серверної частини веб-додатку використано мовку C#. Для виконання обрано шаблон проектування MVC [10] – Model View Controller. Даний шаблон проектування використовується для того, щоб чітко розділити серверну частину, де буде відбуватись обробка запитів в базу даних та логіка передачі та взаємодії за даними, та клієнтську частину.

Для підключення проекту до розробленої бази даних до файлу налаштувань Web.config було додано відповідний тег, зі вказаною адресою бази даних. Відповідну частину для підключення бази даних до сервера показано на (рис. 3.2).

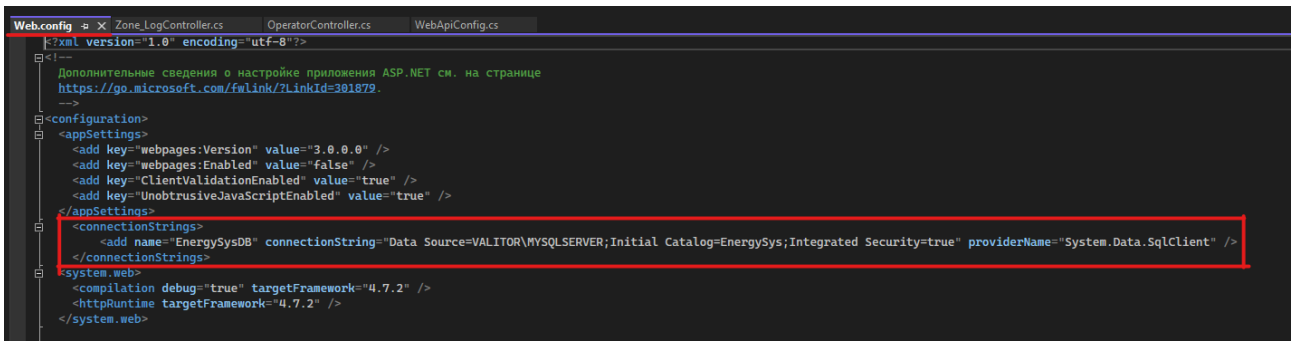


Рисунок 3.2 Тег і адреса для підключення бази даних

У цій роботі розроблено такі класи контролерів:

- **OperatorController** – обробник запитів до таблиці операторів.
- **Sensor_LogController** – обробник запитів до таблиці Звітів Сенсорів.
- **SensorController** – оброник запитів до таблиці Сенсорів.
- **Zone_LogController** – обробник запитів до таблиці Звітів Енергозон.
- **ZoneController** – обробник запитів до таблиці Енергозон.

У роботі розроблено такі класи моделей:

- **Operator –** програмне представлення таблиці Operator в серверній частині додатку.
- **Sensor –** програмне представлення таблиці Sensor в серверній частині додатку.
- **Sensor_Log–** програмне представлення таблиці Sensor_Log в серверній частині додатку.
- **Zone –** програмне представлення таблиц Zone в серверній частині додатку.
- **Zone_Log –** програмне представлення таблиці Zone_Log в серверній частині додатку.

Повна ієрархія усіх файлів веб додатку представлена на (рис 3.3). Написання програмного коду проведено у MS Visual Studio, редакторі коду для безпосереднього створення та редагування сучасних додатків.

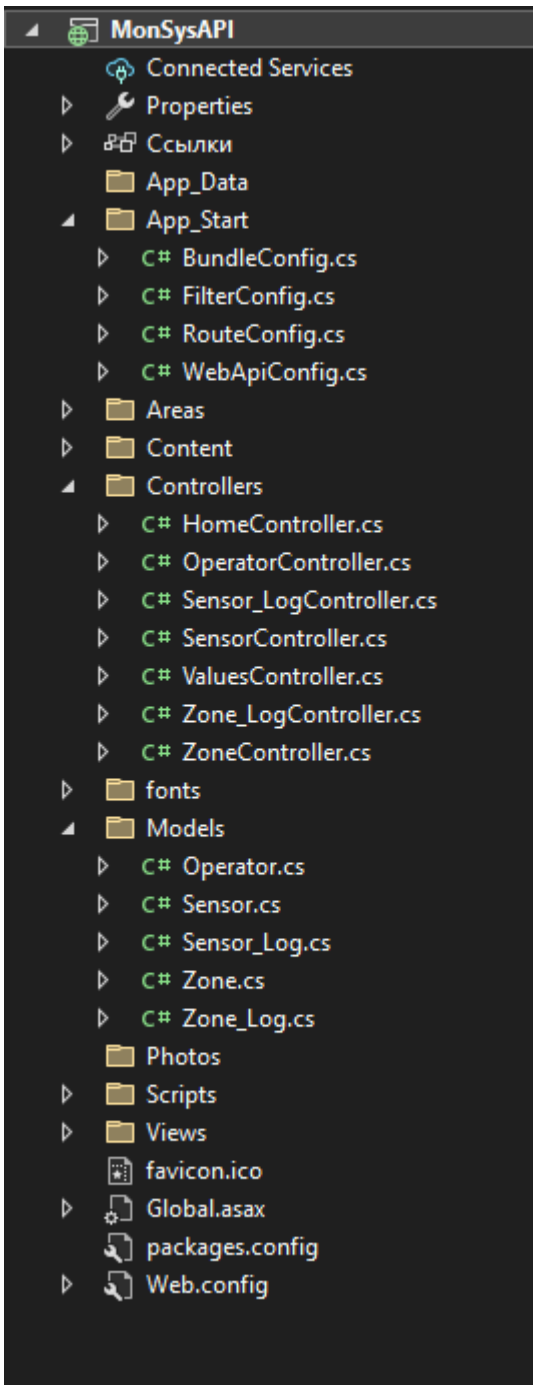


Рисунок 3.3 Ієрархія файлів клієнскої API Частини

Розглянуто приклад контролеру, який використовується у серверній частині для управління таблицею Звітів Енергозон та доступу до відповідної таблиці в базі даних. Називається клас цього контролеру Zone_LogController.

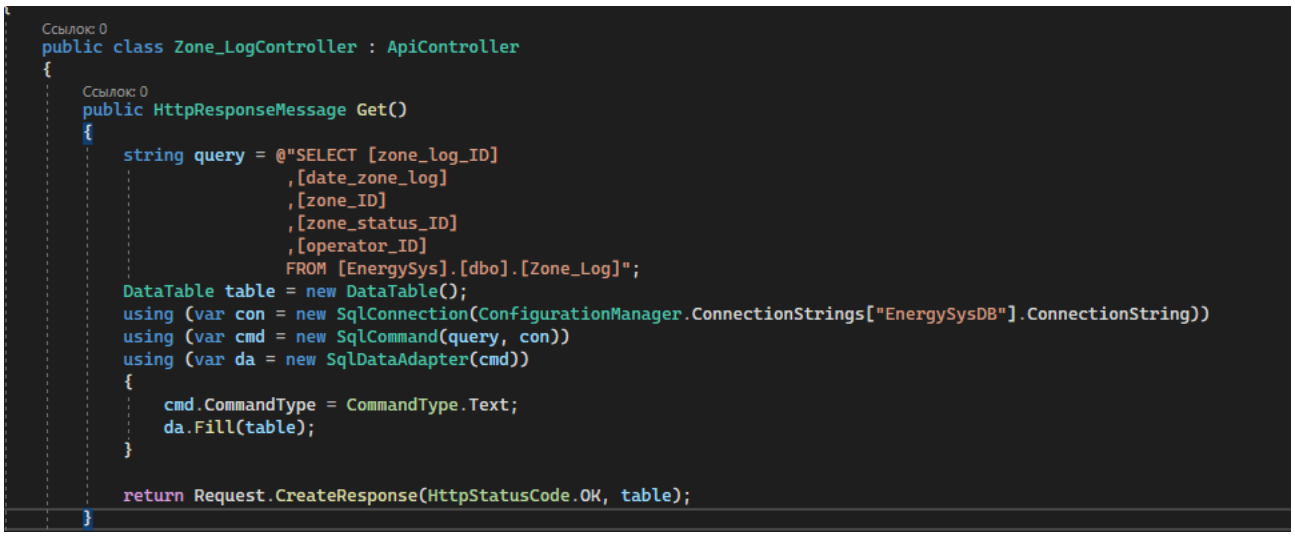
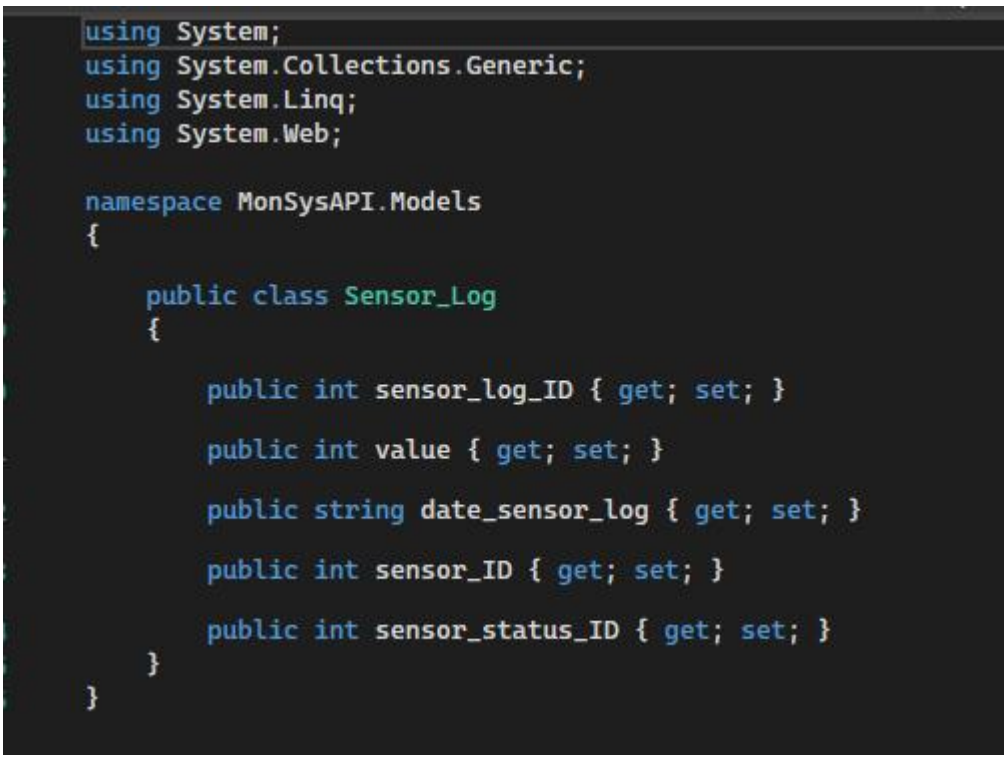


Рисунок 3.4 – Код класу Zone_LogController який виступає контролером таблиці Звітів Енергозон

Як видно на (рис 3.4) в контролерах прописуються API методи з інжекцією SQL запиту. В контролерах також прописуються способи оперування даними для кожного класу, що знаходяться в каталозі Controllers (додаток Б). Вони описують всі види REST-запитів, що будуть доступні для кожного класу за відповідною URL-адресою.

В даному випадку, видно, що до класу Zone_LogController підключається модель Zone_Log, яка описує властивості цієї таблиці за бази даних у програмному вигляді (рис 3.5).



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MonSysAPI.Models
{

    public class Sensor_Log
    {

        public int sensor_log_ID { get; set; }

        public int value { get; set; }

        public string date_sensor_log { get; set; }

        public int sensor_ID { get; set; }

        public int sensor_status_ID { get; set; }
    }
}
```

Рисунок 3.5 – код класу моделі Zone_Log

Вище показано приклад реалізації об’єктно-орієнтованої парадигми програмування разом із шаблоном проектування MVC. Дана зв’язка використовується і в інших частинах та модулях системи для забезпечення гнучкості розробки, масштабування, підвищенню читаності коду та обмеження доступу однієї частини програми до іншої. Дана практика наразі є найбільш поширеною серед розробників веб-застосунків, тому що добре зарекомендувала себе на великий дистанції серед тисяч розробників.

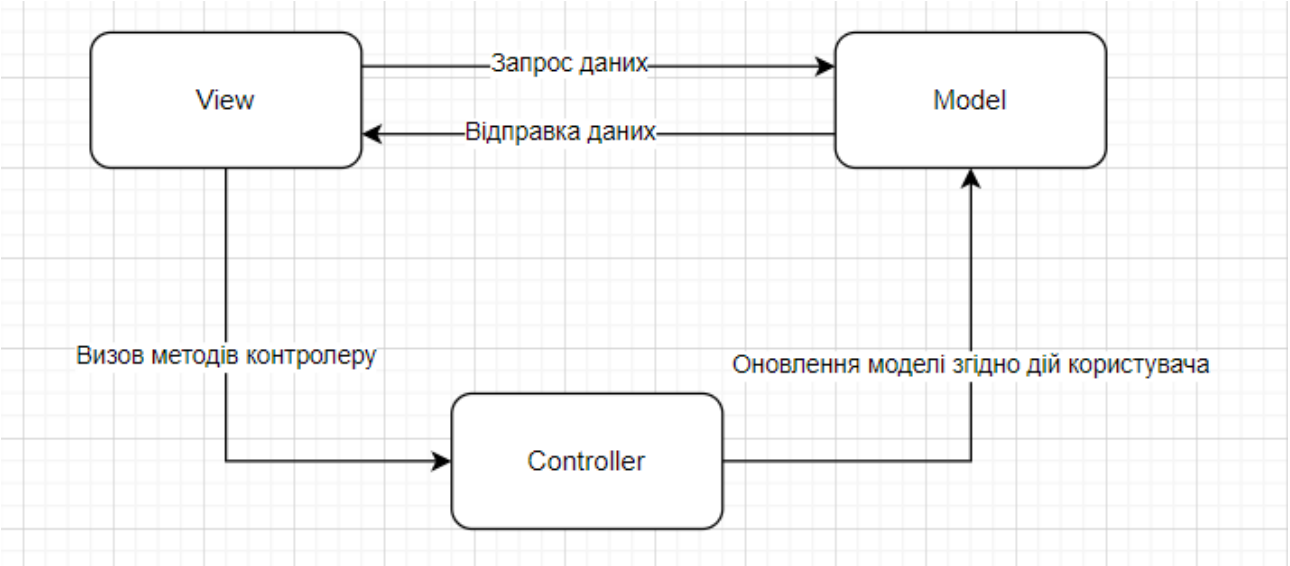


Рисунок 3.6 – принцип роботи веб-додатку за шаблоном проектування MVC

Для кращого розуміння роботи веб-додатку розглянуто схему взаємодії окремих частин: моделей, контролерів та уявлень. Схема представлена на (рис 3.6). Як можна побачити, користувач заходить до системи через певне уявлення – спочатку це сторінка авторизації, а потім головна сторінка системи, і після цього кожний запит користувача йде через певний контролер частини системи. Контролер в свою чергу оновлює дані в базі даних через відповідну йому модель, які користувач потім може побачити на певному уявлені, яке в свою чергу отримує інформацію безпосередньо з моделі. В нашому випадку, контролер і модель знаходяться на окремому серверному додатку. А частина уявлення, а саме фронтенд, буде написаний в другому клієнтському додатку з використанням Angular 13.

У результаті роботи тільки серверної частини було отримано повноцінний сервіс, роботу якого можна перевірити, перейшовши за відповідним посиланням (рис. 3.7) та виконавши таким чином GET-запит. У результаті інтерфейс веб API поверне масив об’єктів JSON, кожен з яких є рядком таблиці, поверненої базою даних у відповідь на запит SELECT, описаний у контролері.

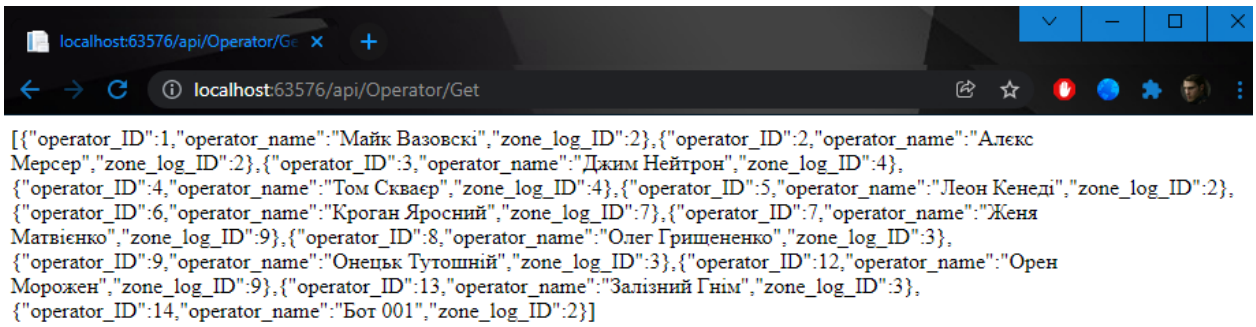


Рисунок 3.7 – Відповідь інтерфейсу веб API на GET-запит браузера

3.4 Опис реалізації клієнтської частини

Наступним кроком є створення Angular проекту веб-додатку за допомогою командного рядка. Такий спосіб дозволяє автоматично згенерувати структуру додатку. Для зв’язку між веб-додатком та базою даних через інтерфейс веб API було згенеровано сервіс shared, у якому описано функції, що використовуються для створення REST-запитів (рис. 3.8) (додаток В). Навігація між сторінками додатку описується у модулі app-routing.module.ts. (рис 3.9)

```typescript
import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs'

@Injectable({
  providedIn: 'root'
})
export class SharedService {
  readonly APIUrl = "http://localhost:5477/api";
  readonly photoUrl = "http://localhost:5477/Photos";

  constructor(private http:HttpClient) { }

  getOperatorList(): Observable<any[]> {
    return this.http.get<any>(this.APIUrl + '/Operator');
  }
  addOperator(val: any) {
    return this.http.post(this.APIUrl + '/Operator', val);
  }

  updateOperator(val: any) {
    return this.http.put(this.APIUrl + '/Operator', val);
  }

  deleteOperator(val: any) {
    return this.http.delete(this.APIUrl + '/Operator/' + val);
  }

  //Photo
  UploadPhoto(val:any){
    return this.http.post(this.APIUrl + '/Operator/SaveFile', val);
  }

  getSensorList(): Observable<any[]> {
    return this.http.get<any>(this.APIUrl + '/Sensor');
  }
  addSensor(val: any) {
    return this.http.post(this.APIUrl + '/Sensor', val);
  }
```

Рисунок 3.8 – Клас SharedService

Як можна споглядати, сам в цьому класі прописані методи які буде використовувати клієнтській додаток, щоб отримати доступ до серверної частини через адресу в полі APIUrl та API методам з мови TypeScript.

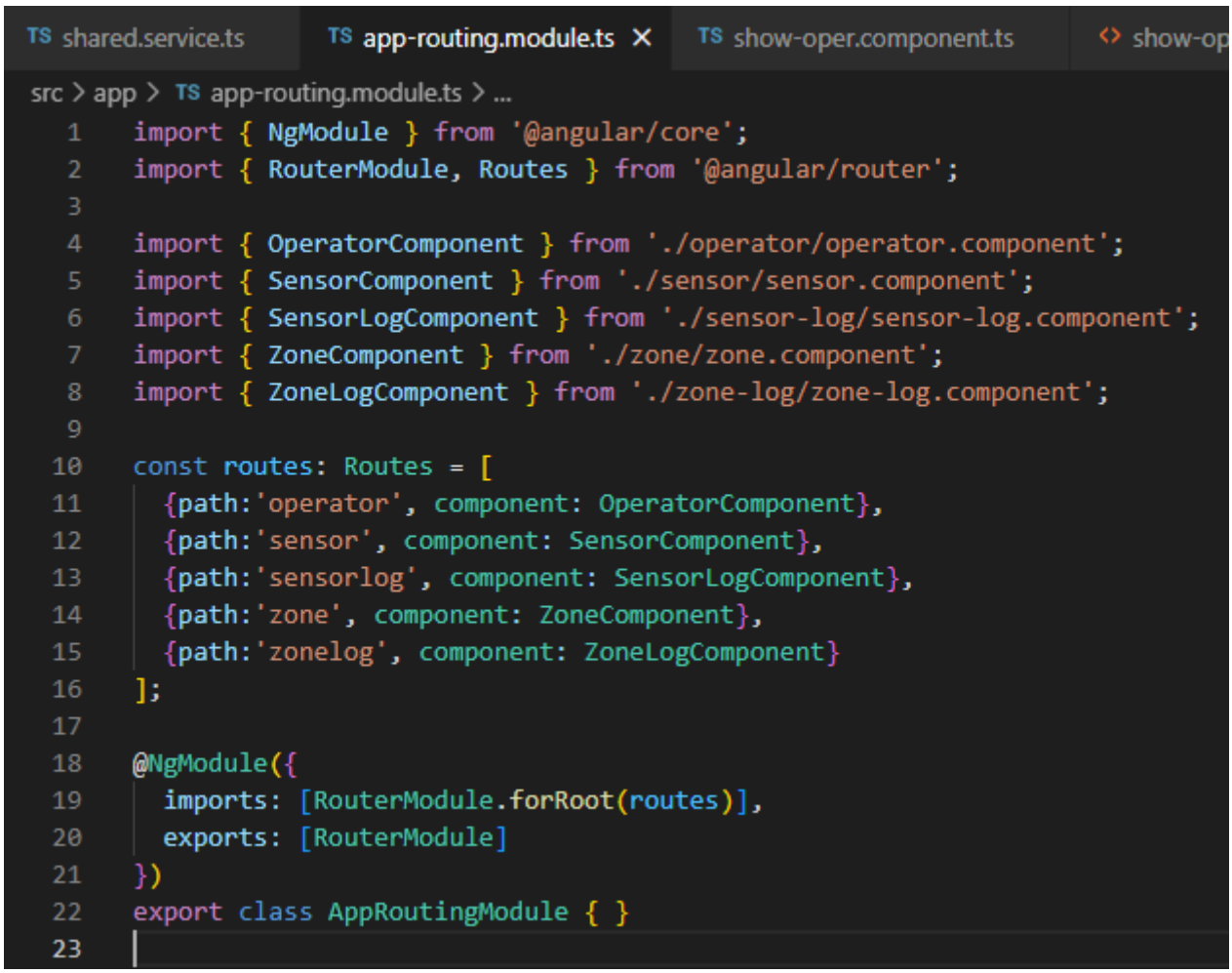


Рисунок 3.9 – Модуль app-routing.module.ts

Саме в цьому класі прописуються шляхи між сторінками для маршрутизації між сторінками. Сторінками будуть компоненти які були створені при створені додатку з консолі. Як видно з (рис 3.9) ці компоненти відповідають таблиці які були створені в базі даних раніше. Але це не обов’язково. Компонентів-сторінок може бути як більше, так і менше кількості діючих таблиць. Хоча такий додаток і називається одно сторінковий, але він буде таким тільки візуально для зручності користувачів.

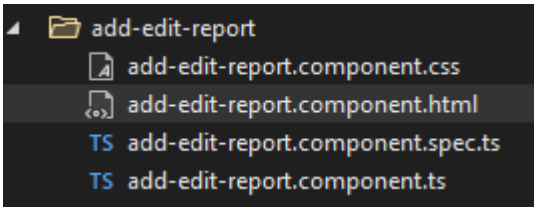


Рис 3.10 – Приклад структури Angular-компоненту

Кожна сторінка задається у вигляді окремого компонента Angular. Кожний такий компонент містить окремі дочірні компоненти, що є фрагментами цієї сторінки. Компонент складається із чотирьох файлів (рис. 3.10):

- CSS файл зі стилями компоненту.
- Файл з HTML-структурою компоненту.
- TypeScript файл для генерації компоненту.
- TypeScript файл з описом логіки обробки даних.

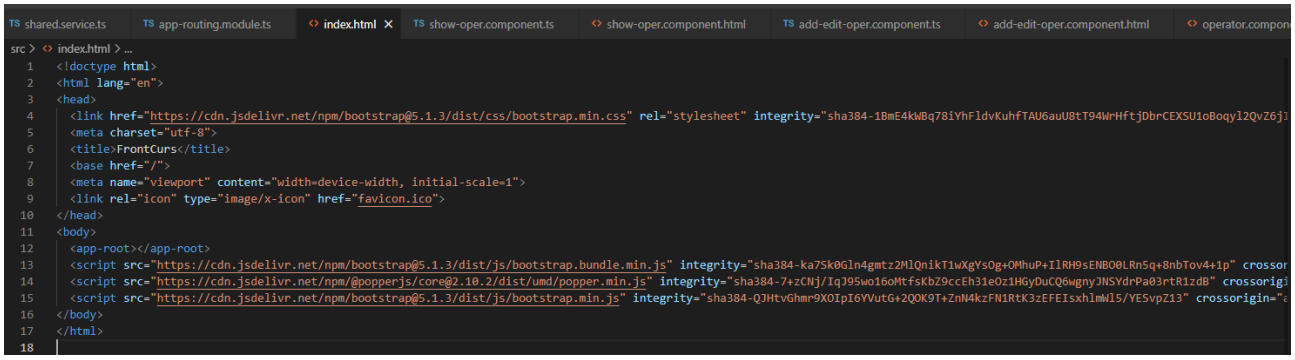


Рисунок 3.11 – Модуль Index.html

Також для зручності і спрощення верстки був добавлений Bootstrap в Index.html файл (рис. 3.11).

Bootstrap — це безкоштовний набір інструментів з відкритим кодом, призначений для створення вебсайтів та вебдодатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних вебсайтів і вебдодатків.

Bootstrap — це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері. [11]

Це набагато спростить розробку інтерфейсу за допомогою використання вже готових рішень. Навіть НАСА використовує Bootstrap.

Далі, коли вже є, пусті поки, компоненти, є налаштована маршрутизація між ними, прописані методи доступу до сервера і є модуль Bootstrap можна почати розробку клієнтської частини. Спершу створюється меню навігації для

орієнтування між таблицями в модулі app.component.html за допомогою HTML коду (рис. 3.12). Досягається навігація тим, що при натисканні відповідної кнопки, буде змінюватись частина в адресній строчці браузера.

```
<div class="container">
  <h3 class="d-flex justify-content-center">Monytoring System</h3>

  <nav class="navbar navbar-expand-sm navbar-dark" style="justify-content: center;">
    <ul class="navbar-nav" >
      <li class="nav-item">
        <button routerLink="operator"
                class="m1 btn btn-light btn-outline-primary" Button>
          Operator
        </button>
      </li>

      <li class="nav-item">
        <button routerLink="sensor" style="margin-left: 10px;"
                class="m1 btn btn-light btn-outline-primary" Button>
          Sensor
        </button>
      </li>

      <li class="nav-item">
        <button routerLink="zone" style="margin-left: 10px;"
                class="m1 btn btn-light btn-outline-primary" Button>
          Zone
        </button>
      </li>

      <li class="nav-item">
        <button routerLink="sensorlog" style="margin-left: 10px;"
                class="m1 btn btn-light btn-outline-primary" Button>
          Sensor Reports
        </button>
      </li>

      <li class="nav-item">
        <button routerLink="zonelog" style="margin-left: 10px;"
                class="m1 btn btn-light btn-outline-primary" Button>
          Zone Reports
        </button>
      </li>
    </ul>
  </nav>
</div>


<router-outlet></router-outlet>
```

Рисунок 3.12 – Модуль app.component.html

Далі прописуються компоненти які відповідають таблицям. Саму таблицю за бази даних в клієнтській частині представляє собою багатоскладний компонент який, в даному випадку, складається з (рис 3.13):

- Загального компоненту.
- Компоненту “show” для відображення таблиці і інших елементів які відносяться до відповідної сторінки.
- Компоненту “add-edit” який відповідає за спливаюче вікно для змін або редагування вже існуючих записів.

Кожний компонент і підкомпонент складається з чотирьох складових як писалось раніше. Тут і далі для наглядності я буду використовувати компонент operator.

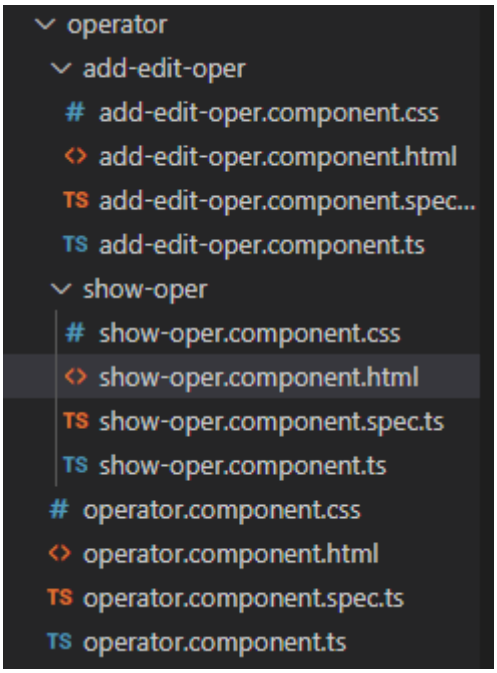


Рисунок 3.13 – Компонент operator і його підкомпоненти

Далі розробка складається в створені функціональної частини відображення таблиці в компоненті show-oper.component.ts. (рис. 3.14)

```
import { Component, OnInit } from '@angular/core';
import { SharedService } from 'src/app/shared.service';

@Component({
  selector: 'app-show-oper',
  templateUrl: './show-oper.component.html',
  styleUrls: ['./show-oper.component.css']
})
export class ShowOperComponent implements OnInit {

  constructor(private service: SharedService) { }

  OperatorList: any = [];
  ModalTitle: string | undefined;
  ActivateAddEditOperator: boolean = false;
  oper: any;

  OperatorIdFilter: string = "";
  OperatorNameFilter: string = "";
  OperatorStatusFilter: string = "";
  OperatorHiringDateFilter: string = "";
  OperatorReleaseDateFilter: string = "";
  OperatorListWithoutFilter: any = [];

  ngOnInit(): void {
    this.refreshOperList();
  }

  refreshOperList() {
    this.service.getOperatorList().subscribe(data => {
      this.OperatorList = data;
      this.OperatorListWithoutFilter = data;
    });
  }
```

Рисунок 3.14 – Компонент show-oper.component.ts

Тут ми прописує поля і методи для відображення таблиці в веб інтерфейсі і для того щоб взагалі зв’язати функціональну частину і фронтенд.

Ці поля використовуються для того щоб зв’язати фронтенд і функціональну частину і вивести таблицю користувачу:

- OperatorList;
- oper.

Метод refreshOperList() відповідає за запити до серверу, а через нього до

бази даних та отримання самих даних.

Метод ngOnInit() оновляє відображену таблицю при змінах в неї. Тобто при додаванні, модифікації чи видалені даних. Самі ці методи теж тут прописуються і зв'язані з компонентом add-edit (рис 3.15).

```
addClick() {
  this.oper = {
    operator_ID: 0,
    operator_name: "",
    operator_status: "",
    hiring_date: "",
    release_date: "",
  }
  this.ModalTitle = "Add Operator";
  this.ActivateAddEditOperator = true;
}

editClick(item: any) {
  this.oper = item;
  this.ModalTitle = "Edit Operator";
  this.ActivateAddEditOperator = true;
}

deleteClick(item: any) {
  if (confirm('Are you sure?')) {
    this.service.deleteOperator(item.operator_ID).subscribe(data => {
      alert(data.toString());
      this.refreshOperList;
    });
  }
}

closeClick() {
  this.ActivateAddEditOperator = false;
  this.refreshOperList();
}
```

Рисунок 3.15 – Методи додавання, модифікації, видалення та закриття спливаючого вікна

Також тут прописуються методи сортування та фільтрації записів таблиці (рис 3.16). Крім цих методів для цього використовуються спеціальні модифікації в HTML коді (рис 3.17) та поля які на (рис 3.14).

```
FilterFn() {
  var OperatorIdFilter = this.OperatorIdFilter;
  var OperatorNameFilter = this.OperatorNameFilter;
  var OperatorStatusFilter = this.OperatorStatusFilter;
  var OperatorHiringDateFilter = this.OperatorHiringDateFilter;
  var OperatorReleaseDateFilter = this.OperatorReleaseDateFilter;

  this.OperatorList = this.OperatorListWithoutFilter.filter(function (el: any) {
    return el.operator_ID.toString().toLowerCase().includes(
      OperatorIdFilter.toString().trim().toLowerCase()
    ) &&
      el.operator_name.toString().toLowerCase().includes(
        OperatorNameFilter.toString().trim().toLowerCase()
      ) &&
      el.operator_status.toString().toLowerCase().includes(
        OperatorStatusFilter.toString().trim().toLowerCase()
      ) &&
      el.hiring_date.toString().toLowerCase().includes(
        OperatorHiringDateFilter.toString().trim().toLowerCase()
      ) &&
      el.release_date.toString().toLowerCase().includes(
        OperatorReleaseDateFilter.toString().trim().toLowerCase()
      )
  });
}

sortResult(prop: any, asc: boolean) {
  this.OperatorList = this.OperatorList.sort(function (a: any, b: any) {
    if (asc) {
      return (a[prop] > b[prop]) ? 1 : (a[prop] < b[prop]) ? -1 : 0;
    }
    else {
      return (b[prop] > a[prop]) ? 1 : (b[prop] < a[prop]) ? -1 : 0;
    }
  });
}
```

Рисунок 3.16 – Методи фільтрації та сортування

```
<input [(ngModel)]="OperatorNameFilter" class="form-control" (keyup)="FilterFn()" placeholder="Name" type="text" style="text-align: center;">
<button type="button" class="btn btn-light"
        (click)="sortResult('operator_name',true)">
  <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-sort-up-alt" viewBox="0 0 16 16">
    <path d="M3.5 13.5a.5.5 0 0 1-1 0V4.707L1.354 5.854a.5.5 0 1 1-.708-.708l2-1.999.007-.007a.498.498 0 0 1 .7.006l2 2a.5.5 0 1 1-.707.708L3.5 4.707V13.5zm4-9.5a.5.5 0 0
  </svg>
</button>
<button type="button" class="btn btn-light"
        (click)="sortResult('operator_name',false)">
  <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-sort-down" viewBox="0 0 16 16">
    <path d="M3.5 2.5a.5.5 0 0 0-1 0v8.793l-1.146-1.147a.5.5 0 0 0-.708.708l2 1.999.007.007a.497.497 0 0 0 .7-.006l2-2a.5.5 0 0 0-.707-.708L3.5 11.293V2.5zm3.5 1a.5.5 0
  </svg>
</button>
```

Рисунок 3.17 – Модифікації в HTML коді

Коли функціональна частина готова, то йде верстка фронтенду в відповідному show.html файлі. Бутстрап набагато спрощує роботу с версткою.

Після цього у сторінка для таблиці Operator буде готова. Вона буде в себе включати (рис 3.18):

1. Вивід даних таблиці.
2. Кнопки які визивають спливаюче вікно для:
    a. Додавання записів.
    b. Модифікації записів.
    c. Видалення записів.
3. Поля фільтраціх.
4. Кнопки сортування.

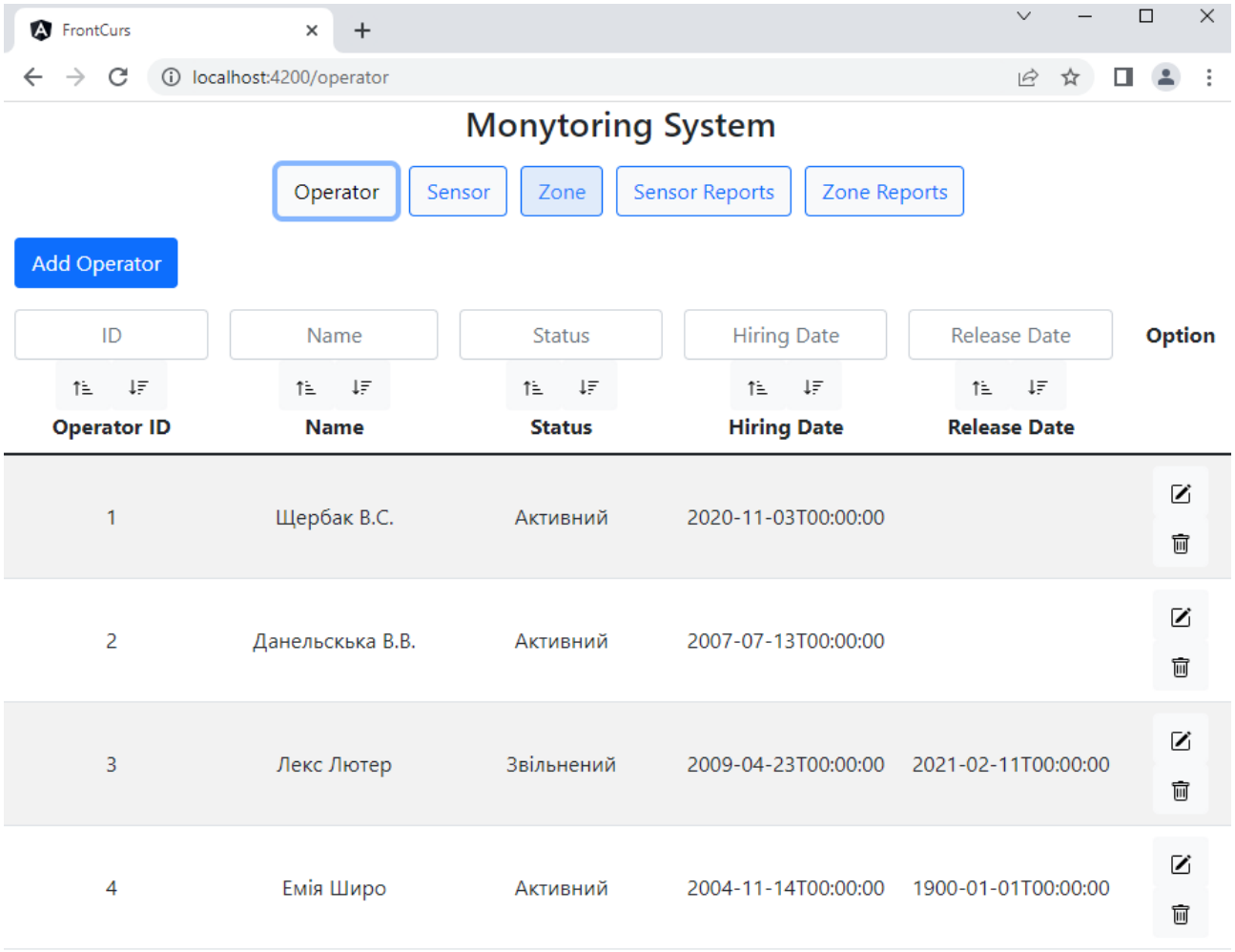


Рисунок 3.18 – Сторінка Операторів

Наступним кроком буде створення спливаючого вікна для редагування та додавання нових записів. Технічно, це буде одне і теж саме вікно, але воно буде змінювати свою функціональність в залежності від того, чи надходить до класу ІД оператора, чи ні. Якщо воно відсутнє, то буде викликатись метод добавлення нового оператора (рис 3.19). Якщо ІД надходить, то вікно автоматично заповнюється записами цього оператора і його дані можна змінювати (рис 3.20).

Рисунок 3.19 – Вікно добавлення нового Оператора

Рисунок 3.20 – Вікно модифікації Оператора

В залежності від того якою кнопкою ми його викликаєм буде змінюватися його заголовок і кнопка, але в коді, це все одно одне і теж вікно. А функціонал вікна походить з компонента add-edit-oper.component.ts. Функції цього компоненту (рис 3.21):

- @Input() oper – отримує записи з show.oper;
- ngOnInit() – синхронізує записи show.oper з add-edit-oper;
- addOperator() – отримує дані які вводить користувач та передає їх на сервер з командую добавити запис в БД;
- updateOperator() – отримує дані які були у вікні (змінені вони, чи ні), та передає на сервер з командою оновити запис в БД.

```
export class AddEditOperComponent implements OnInit {

  constructor(private servise: SharedService) { }

  @Input() oper:any;
  operator_ID:string | undefined;
  operator_name:string | undefined;
  operator_status:string | undefined;
  hiring_date:string | undefined;
  release_date:string | undefined;

  ngOnInit(): void {
    this.operator_ID = this.oper.operator_ID;
    this.operator_name = this.oper.operator_name;
    this.operator_status = this.oper.operator_status;
    this.hiring_date = this.oper.hiring_date;
    this.release_date = this.oper.release_date;
  }

  addOperator() {
    var val = {
      operator_ID: this.operator_ID,
      operator_name: this.operator_name,
      operator_status: this.operator_status,
      hiring_date: this.hiring_date,
      release_date: this.release_date
    };
    this.servise.addOperator(val).subscribe(res => {
      alert(res.toString());
    });
  }

  updateOperator() {
    var val = {
      operator_ID: this.operator_ID,
      operator_name: this.operator_name,
      operator_status: this.operator_status,
      hiring_date: this.hiring_date,
      release_date: this.release_date
    };
    this.servise.updateOperator(val).subscribe(res => {
      alert(res.toString());
    });
  }
```

Рисунок 3.21 – Компонент add-edit-oper.component.ts

HTML компонент компонента add-edit.oper представляє собою звичайний документ з визначенням полів, їх властивостей, лейблів і т.д. Крі останіх двух строк. Сама із-за них спливаюче єдине вікно має різноманітний функціонал в залежності від натиснутої кнопки і отриманого значення ІД (рис. 3.22).

```
<button (click)="addOperator()" *ngIf="oper.operator_ID==0" class="btn btn-primary">
  Add
</button>

<button (click)="updateOperator()" *ngIf="oper.operator_ID!=0" class="btn btn-primary">
  Update
</button>
```

Рисунок 3.22 – Розгалуження функціоналу спливаючого вікна

3.5 Висновок

У даному розділі було створено додаток, що складається з динамічного модульного вікна. Для цього було розроблено серверний інтерфейс веб API для зв’язку клієнтського інтерфейсу з базою даних. Для написання клієнтського додатку було використано фреймворк Angular. Для створення та керування базою даних було використано СУБД Microsoft SQL Server Management Studio 18. У результаті було отримано базу даних та веб-додаток, що дозволяє слідкувати за станом зон енерголіній, датчиків та їх показниками у зручному форматі.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи бакалавра створено програмний продукт загального моніторингу декілька енергозон з ефективним використанням ресурсів.

В ході даної роботи було виконано:

- У ході виконання роботи, розглянуто аналоги існуючих систем моніторингу які застосовуються в різних сферах енергетики, в енергопостачанні включно.
- Спроектовано та розроблено концептуальну, логічну і фізичну моделі бази даних системи моніторингу енергопостачання. На основі побудованих моделей, розроблено базу даних на базі MS SQL SERVER за допомогою MS SQL Server Management Studio 18.
- Розроблений сервер для зв'язку клієнтського додатку з Базою даних. Сервер був спроектований на моделі Model-View-Controller і створений за допомогою мови програмування C#, а також використання технологій ASP.NET Core, .NET Core. Сервер представляє з себе приймач і обробник API запитів зі сторони клієнтського додатку. І завдяки підключені бази даних до сервера спілкується за базою даних напряму через SQL.
- Спроектовано та розроблено дизайн клієнтського додатку за допомогою поєднання мови гіпертексту HTML5, мови стилю сторінок CSS3, а також фреймворку Bootstrap. Для впровадження функціоналу в клієнтську частину було використано скриптову мову TypeScript. Сам додаток був спроектований і створений за допомогою платформи для розробки веб-додатків Angular 13.

У результаті було отримано систему, що дозволяє здійснювати загальний моніторинг великої кількості енергомереж малими силами. Веб-додаток дозволяє у зручному форматі переглядати дані, додавати нові записи до таблиць бази даних та вносити зміни до вже існуючих записів.

Поставлена задача виконана, мета кваліфікаційної роботи бакалавра досягнута.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Стаття «Favoring IoT Technology to Monitor Industrial Energy» [Електронний ресурс] – Режим доступу до ресурсу: https://www.biz4intellia.com/blog/favoring-iot-technology-to-monitor-industrial-energy/ (дата звертання 14.05.2022)
2. Документація Life Is On [Електронний ресурс] – Режим доступу до ресурсу: https://www.se.com/ru/ru/product-range/61280-ecostruxure-power-monitoring-expert/?parent-subcategory-id=4170#overview (дата звертання 16.05.2022)
3. Стаття «Energy Controller 2 - мониторинг параметров ИБП» [Електронний ресурс] – Режим доступу до ресурсу: https://sector.biz.ua/docs/ups_energy_controller_2/ups_energy_controller_2.phtml#.YqzdPHZByUl (дата звертання 16.05.2022)
4. Сторінка товару «Устройство для мониторинга Solar Log 1200» [Електронний ресурс] – Режим доступу до ресурсу: https://alfa.solar/ru/ustrojstvo-dlya-monitoringa-solar-log-1200-id614.html (дата звертання 16.05.2022)
5. Документація «Всебічний моніторинг системи електропостачання комунікаційного обладнання» [Електронний ресурс] – Режим доступу до ресурсу: https://aggregate.digital/downloads/open/aggregate/case_study_united_energy_company_ru.pdf (дата звертання 16.05.2022)
6. Документація «Централізований моніторинг електропостачання базових станцій стільникової мережі» [Електронний ресурс] – Режим доступу до ресурсу: https://aggregate.digital/downloads/open/aggregate/case_study_flexenclosure_ru.pdf (дата звертання 16.05.2022)
7. Документація про Erwin Forward Engineering/Schema Generation [Електронний ресурс] – Режим доступу до ресурсу: https://bookshelf.erwin.com/bookshelf/public_html/2020R2/Content/User%

20Guides/erwin%20Help/foward_engineering_schema_generation.html (дата звертання 18.05.2022)

8. Стаття «Удивительный Angular / Хабр» [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/348818/ (дата звертання 20.05.2022)
9. Документація NET | Free. Cross-platform. Open Source [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/ (дата звертання 18.05.2022)
10. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Model-View-Controller (дата звертання 18.05.2022)
11. Документація Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.2/getting-started/introduction/ (дата звертання 12.05.2022)

ДОДАТОК А

SQL-скрипт для створення бази даних

Листів 4

Розробник Щербак В.С.

Керівник Степанов М.М.

Київ - 2022

Частина скрипту для створення бази даних, згенерований середовищем Erwin Data Modeler

```
CREATE DEFAULT normal_status
	AS 'Нормальний'
go


CREATE TABLE Operator
(
	operator_ID	integer IDENTITY ( 1,1 ) ,
	operator_name	varchar(32)  NOT NULL ,
	zone_log_ID	integer  NULL
)
go


ALTER TABLE Operator
	ADD CONSTRAINT XPKOperator PRIMARY KEY CLUSTERED (operator_ID ASC)
go


CREATE TABLE Report
(
	report_ID	integer IDENTITY ( 1,1 ) ,
	report_date	datetime  NOT NULL ,
	operator_ID	integer  NOT NULL ,
	zone_log_ID	integer  NULL
)
go


ALTER TABLE Report
	ADD CONSTRAINT XPKReport PRIMARY KEY CLUSTERED (report_ID ASC)
go


CREATE TABLE Sensor
(
	sensor_ID	integer IDENTITY ( 1,1 ) ,
	sensor_name	varchar(20)  NOT NULL ,
	sensor_cost	money  NOT NULL ,
	installation_date	datetime  NULL ,
	zone_ID	integer  NOT NULL ,
	Sensor_Type_ID	integer  NOT NULL
)
go


ALTER TABLE Sensor
	ADD CONSTRAINT XPKSensor PRIMARY KEY CLUSTERED (sensor_ID ASC)
go


CREATE TABLE Sensor_Log
(
	sensor_log_ID	integer IDENTITY ( 1,1 ) ,
	Sensor_Status	varchar(20)  NOT NULL ,
	Value	integer  NOT NULL ,
```

```
	date_sensor_log	datetime  NOT NULL ,
	sensor_ID	integer  NOT NULL ,
	sensor_status_ID	integer  NOT NULL
)
go

ALTER TABLE Sensor_Log
	ADD CONSTRAINT XPKSensor_Log PRIMARY KEY CLUSTERED (sensor_log_ID ASC)
go

CREATE TABLE Sensor_Status
(
	sensor_status_ID	integer IDENTITY ( 1,1 ) ,
	sensor_status	varchar(20)  NOT NULL
)
go

ALTER TABLE Sensor_Status
	ADD CONSTRAINT XPKSensor_Status PRIMARY KEY  CLUSTERED (sensor_status_ID ASC)
go

CREATE TABLE Sensor_Type
(
	Sensor_Type_ID	integer IDENTITY ( 1,1 ) ,
	sensor_type	varchar(20)  NOT NULL
)
go

ALTER TABLE Sensor_Type
	ADD CONSTRAINT XPKSensor_Type PRIMARY KEY CLUSTERED (Sensor_Type_ID ASC)
go

CREATE TABLE Zone
(
	zone_ID	integer IDENTITY ( 1,1 ) ,
	zone_name	varchar(20)  NOT NULL ,
	place	varchar(20)  NOT NULL
)
go

ALTER TABLE Zone
	ADD CONSTRAINT XPKZone PRIMARY KEY CLUSTERED (zone_ID ASC)
go

CREATE UNIQUE NONCLUSTERED INDEX XAK1Zone ON Zone
(
	zone_name	ASC
)
go

CREATE TABLE Zone_Log
(
	zone_log_ID	integer IDENTITY ( 1,1 ) ,
	zone_status	varchar(20)  NOT NULL ,
```

```
	date_zone_log	datetime  NOT NULL ,
	zone_ID		integer  NOT NULL ,
	zone_status_ID	integer  NOT NULL
)
go

ALTER TABLE Zone_Log
	ADD CONSTRAINT XPKZone_Log PRIMARY KEY CLUSTERED (zone_log_ID ASC)
go

CREATE TABLE Zone_Status
(
	zone_status_ID	integer IDENTITY ( 1,1 ) ,
	zone_status	varchar(20)  NOT NULL
)
go

ALTER TABLE Zone_Status
	ADD CONSTRAINT XPKZone_Status PRIMARY KEY CLUSTERED (zone_status_ID ASC)
go

CREATE VIEW Report_on_subordinate_Zones(operator_name,zone_name,AVG_Value)
AS
SELECT Operator.operator_name,Zone.zone_name,AVG(Sensor_Log.value)
	FROM Operator,Zone,Sensor_Log
go

ALTER TABLE Operator
	ADD CONSTRAINT R_33 FOREIGN KEY (zone_log_ID) REFERENCES Zone_Log(zone_log_ID)
		ON DELETE NO ACTION
		ON UPDATE NO ACTION
go

ALTER TABLE Report
	ADD CONSTRAINT R_28 FOREIGN KEY (operator_ID) REFERENCES Operator(operator_ID)
		ON DELETE NO ACTION
		ON UPDATE NO ACTION
go

ALTER TABLE Report
	ADD CONSTRAINT R_30 FOREIGN KEY (zone_log_ID) REFERENCES Zone_Log(zone_log_ID)
		ON DELETE NO ACTION
		ON UPDATE NO ACTION
go

ALTER TABLE Sensor
	ADD CONSTRAINT R_25 FOREIGN KEY (zone_ID) REFERENCES Zone(zone_ID)
		ON DELETE NO ACTION
		ON UPDATE NO ACTION
go

ALTER TABLE Sensor
	ADD CONSTRAINT R_34 FOREIGN KEY (Sensor_Type_ID) REFERENCES Sensor_Type(Sensor_Type_ID)
		ON DELETE NO ACTION
		ON UPDATE NO ACTION
```

```
go

ALTER TABLE Sensor_Log
        ADD CONSTRAINT R_27 FOREIGN KEY
(sensor_ID) REFERENCES Sensor(sensor_ID)
                ON DELETE NO ACTION
                ON UPDATE NO ACTION
go

ALTER TABLE Sensor_Log
        ADD CONSTRAINT R_36 FOREIGN KEY
(sensor_status_ID) REFERENCES
Sensor_Status(sensor_status_ID)
                ON DELETE NO ACTION
                ON UPDATE NO ACTION
go

ALTER TABLE Zone_Log
        ADD CONSTRAINT R_29 FOREIGN KEY (zone_ID)
REFERENCES Zone(zone_ID)
                ON DELETE NO ACTION
                ON UPDATE NO ACTION
go

ALTER TABLE Zone_Log

        ADD CONSTRAINT R_35 FOREIGN KEY
(zone_status_ID) REFERENCES
Zone_Status(zone_status_ID)
                ON DELETE NO ACTION
                ON UPDATE NO ACTION
go

CREATE TRIGGER tD_Operator ON Operator FOR DELETE
AS
/* ERwin Builtin Trigger */
/* DELETE trigger on Operator */
BEGIN
 DECLARE  @errno   int,
       @errmsg  varchar(255)
  /* ERwin Builtin Trigger */
  /* Operator  Report on parent delete no action */
  /* ERWIN_RELATION:CHECKSUM="0001fdf1",
PARENT_OWNER="", PARENT_TABLE="Operator"
  CHILD_OWNER="", CHILD_TABLE="Report"
  P2C_VERB_PHRASE="", C2P_VERB_PHRASE="",
  FK_CONSTRAINT="R_28", FK_COLUMNS="operator_ID"
*/
  IF EXISTS (
    SELECT * FROM deleted,Report
    WHERE
      /*  %JoinFKPK(Report,deleted," = "," AND") */
      Report.operator_ID = deleted.operator_ID
  )
```

ДОДАТОК Б

Програмний код

Листів 6

Розробник Щербак В.С.

Керівник Степанов М.М.

Київ - 2022

Програмний код

Фрагмент коду опису класу Логів Енергозон для інтерфейсу веб API

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Monitoring_System.Models
{
   public class Zone_Log
   {
      public int zone_log_ID { get; set; }
      public DateTime date_zone_log { get; set; }
      public int zone_ID { get; set; }
      public int zone_status_ID { get; set; }
      public int voltage { get; set; }
      public int tension { get; set; }
      public int wind { get; set; }
   }
}
```

Фрагмент коду контролеру класу Енергозони

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using Monitoring_System.Models;

namespace Monitoring_System.Controllers
{
   public class ZoneController : ApiController
   {
      public HttpResponseMessage Get()
      {
         string query = @"
                  select zone_id, zone_name, place
                  from [Energy Monitoring].dbo.Zone
                  ";
         DataTable table = new DataTable();
         using (var con = new
SqlConnection(ConfigurationManager.
            ConnectionStrings["Monitoring
System"].ConnectionString))
            using (var cmd = new SqlCommand(query,
con))
         using (var da = new SqlDataAdapter(cmd))
         {
            cmd.CommandType = CommandType.Text;
            da.Fill(table);
         }
         return
Request.CreateResponse(HttpStatusCode.OK, table);

      }

      public string Post(Zone dev)
      {
         string query = @"
                     insert into [Energy
Monitoring].dbo.Zone (zone_name, place) values
                     ('" + dev.zone_name + @"','"
                     + dev.place + @"')";
         try
```

```
        {


            DataTable table = new DataTable();
            using (var con = new
SqlConnection(ConfigurationManager.ConnectionStr
ings["Monitoring System"].ConnectionString))
            using (var cmd = new SqlCommand(query,
con))
            using (var da = new SqlDataAdapter(cmd))
            {
                cmd.CommandType =
CommandType.Text;
                da.Fill(table);
            }


            return "Added Successfully!";
        }
        catch
        {
            return "Failed to Add!";
        }
    }


    public string Put(Zone dev)
    {
        string query = @"
                    update [Energy
Monitoring].dbo.Zone set place=
                    '" + dev.place + @"'
                    where zone_name like '" +
dev.zone_name + @"'";
        try
        {


            DataTable table = new DataTable();
            using (var con = new
SqlConnection(ConfigurationManager.ConnectionStr
ings["Monitoring System"].ConnectionString))
            using (var cmd = new SqlCommand(query,
con))
            using (var da = new SqlDataAdapter(cmd))
            {
                cmd.CommandType =
CommandType.Text;
                da.Fill(table);
            }


            return "Update Successfully!";
        }
        catch
        {
            return "Failed to Update!";
        }
    }


    [HttpDelete]
    public string Delete(int id)
    {
        string query = @"
                    delete from [Energy
Monitoring].dbo.Zone
                    where zone_ID=" + id + @"";
        try
        {


            DataTable table = new DataTable();
            using (var con = new
SqlConnection(ConfigurationManager.ConnectionStr
ings["Monitoring System"].ConnectionString))
            using (var cmd = new SqlCommand(query,
con))
            using (var da = new SqlDataAdapter(cmd))
```

```
            {
                cmd.CommandType = CommandType.Text;
                da.Fill(table);
            }

            return "Delete Successfully!";
        }
        catch
        {
            return "Failed to Delete!";
        }
    }

  }
}
```

Фрагмент коду Angular-сервісу додатку

```
import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
 providedIn: 'root'
})
export class SharedService {
 readonly APIUrl = "http://localhost:63576/api";

 constructor(private http: HttpClient) { }

 getOperList(): Observable<any[]> {
  return this.http.get<any>(this.APIUrl + '/Operator');
 }

 addOper(val: any) {
  return this.http.post(this.APIUrl + '/operator', val);
 }

 updateOper(val: any) {
  return this.http.put(this.APIUrl + '/operator', val);
 }

 delateOper(val: any) {
  return this.http.delete(this.APIUrl + '/operator' + val);
 }

 getZone_LogList(): Observable<any[]> {
  return this.http.get<any>(this.APIUrl + '/Zone_Log');
 }

 addZone_Log(val: any) {;
  return this.http.post(this.APIUrl + '/Zone_Log', val);
 }

 updateZone_Log(val: any) {
  return this.http.put(this.APIUrl + '/Zone_Log', val);
 }

 delateZone_Log(val: any) {
  return this.http.delete(this.APIUrl + '/Zone_Log' + val);
 }

 getAllZoneStatus(): Observable<any[]> {
  return this.http.get<any[]>(this.APIUrl + '/Zone_Log/GetAllZoneStatus');
```

```
}

getZoneList(): Observable<any[]> {
  return this.http.get<any>(this.APIUrl + '/Zone');
}

addZone(val: any) {
  return this.http.post(this.APIUrl + '/Zone', val);
}

updateZone(val: any) {
  return this.http.put(this.APIUrl + '/Zone', val);
}

delateZone(val: any) {
  return this.http.delete(this.APIUrl + '/Zone' + val);
}

getSensorList(): Observable<any[]> {
  return this.http.get<any>(this.APIUrl + '/Sensor');
}

addSensor(val: any) {
  return this.http.post(this.APIUrl + '/Sensor', val);
}

updateSensor(val: any) {
  return this.http.put(this.APIUrl + '/Sensor', val);
}

delateSensor(val: any) {
  return this.http.delete(this.APIUrl + '/Sensor' +
val);
}

getReportList(): Observable<any[]> {
  return this.http.get<any>(this.APIUrl + '/Report');
}

addReport(val: any) {
  return this.http.post(this.APIUrl + '/Report', val);
}

updateReport(val: any) {
  return this.http.put(this.APIUrl + '/Report', val);
}

delateReport(val: any) {
  return this.http.delete(this.APIUrl + '/Report' +
val);
}

getSensor_LogList(): Observable<any[]> {
  return this.http.get<any>(this.APIUrl +
'/Sensor_Log');
}

addSensor_Log(val: any) {
  return this.http.post(this.APIUrl + '/Sensor_Log',
val);
}

updateSensor_Log(val: any) {
  return this.http.put(this.APIUrl + '/Sensor_Log',
val);
}

delateSensor_Log(val: any) {
  return this.http.delete(this.APIUrl + '/Sensor_Log'
+ val);
```

```
 }

}
```

Фрагмент HTML коду Angular-компоненту вкладки Звітів

```
<button type="button" class="btn btn-primary float-right m-2"
      data-toggle="modal" data-target="#exampleModal"
      (click)="addClick()"
      data-backdrop="static" data-keyboard="false">
 Add Report
</button>


<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">
 <div class="modal-dialog modal-dialog-centered modal-xl" role="document">
   <div class="modal-content">
    <div class="modal-header">
      <h5 class="modal-title" id="exampleModalLabel">{{ModalTitle}}</h5>
      <button type="button" class="close" data-dismiss="modal" aria-label="Close"
            (click)="closeClick()">
       <span aria-hidden="true">&times;</span>
      </button>
    </div>
    <div class="body">
      <app-add-edit-report [report]="report" *ngIf="ActivateAddEditReport">
      </app-add-edit-report>
    </div>
   </div>
 </div>
</div>


<table class="table table-striped">
 <thead>
  <tr>
   <th>Report ID</th>
   <th>Report Date</th>
   <th>Operator ID</th>
   <th>Zone Log ID</th>
  </tr>
 </thead>
 <tbody>
  <tr *ngFor="let dataItem of ReportList">
   <td>{{dataItem.report_ID}}</td>
   <td>{{dataItem.report_date}}</td>
   <td>{{dataItem.operator_ID}}</td>
   <td>{{dataItem.zone_log_ID}}</td>
   <td>
    <button type="button" class="btn btn-light mr-1"
          data-toggle="modal" data-target="#exampleModal"
          (click)="editClick(dataItem)"
          data-backdrop="static" data-keyboard="false">
      <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-pencil" viewBox="0 0 16 16">
        <path d="M12.146.146a.5.5 0 0 1 .708 0l3 3a.5.5 0 0 1 0 .708l-10 10a.5.5 0 0 1-.168.11l-5 2a.5.5 0 0 1-.65-.65l2-5a.5.5 0 0 1 .11-.168l10-10zM11.207 2.5 13.5 4.793 14.793 3.5 12.5 1.207 11.207 2.5zm1.586 3L10.5 3.207 4 9.707V10h.5a.5.5 0 0 1 .5.5v.5h.5a.5.5 0 0 1 .5.5v.5h.293l6.5-6.5zm-9.761 5.175-.106.106-1.528 3.821 3.821-1.528.106-.106A.5.5 0 0 1 5 12.5V12h-.5a.5.5 0 0 1-.5-.5V11h-.5a.5.5 0 0 1-.468-.325z" />
      </svg>
```

```
      </button>

      <button type="button" class="btn btn-light mr-
1"

            (click)="deleteClick(dataItem)">

        <svg xmlns="http://www.w3.org/2000/svg"
width="16" height="16" fill="currentColor" class="bi
bi-trash" viewBox="0 0 16 16">

          <path d="M5.5 5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-
1 0V6a.5.5 0 0 1 .5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0
0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0 0-1 0v6a.5.5
0 0 0 1 0V6z" />

          <path fill-rule="evenodd" d="M14.5 3a1 1 0 0
1-1 1H13v9a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2V4h-.5a1 1
0 0 1-1-1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0 0
1 1 1h3.5a1 1 0 0 1 1 1v1zM4.118 4 4 4.059V13a1 1
0 0 0 1 1h6a1 1 0 0 0 1-1V4.059L11.882
4H4.118zM2.5 3V2h11v1h-11z" />

        </svg>

      </button>

    </td>

  </tr>

 </tbody>

</table>
```

ДОДАТОК В

Інструкція користувачеві веб-додатком

Листів 5

Розробник Щербак В.С.

Керівник Степанов М.М.

Київ - 2022

1. Підключившись до системи, Оператор опиниться, на панелі вкладок. Користувачу слід натиснути на ту вкладку, з якою хоче працювати.

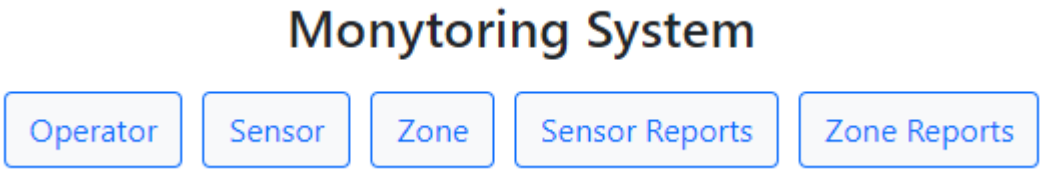


Рисунок 1 – Панель вкладок

2. Опинившись на вкладці з необхідними даними, Оператор може: спостерігати/аналізувати дані.

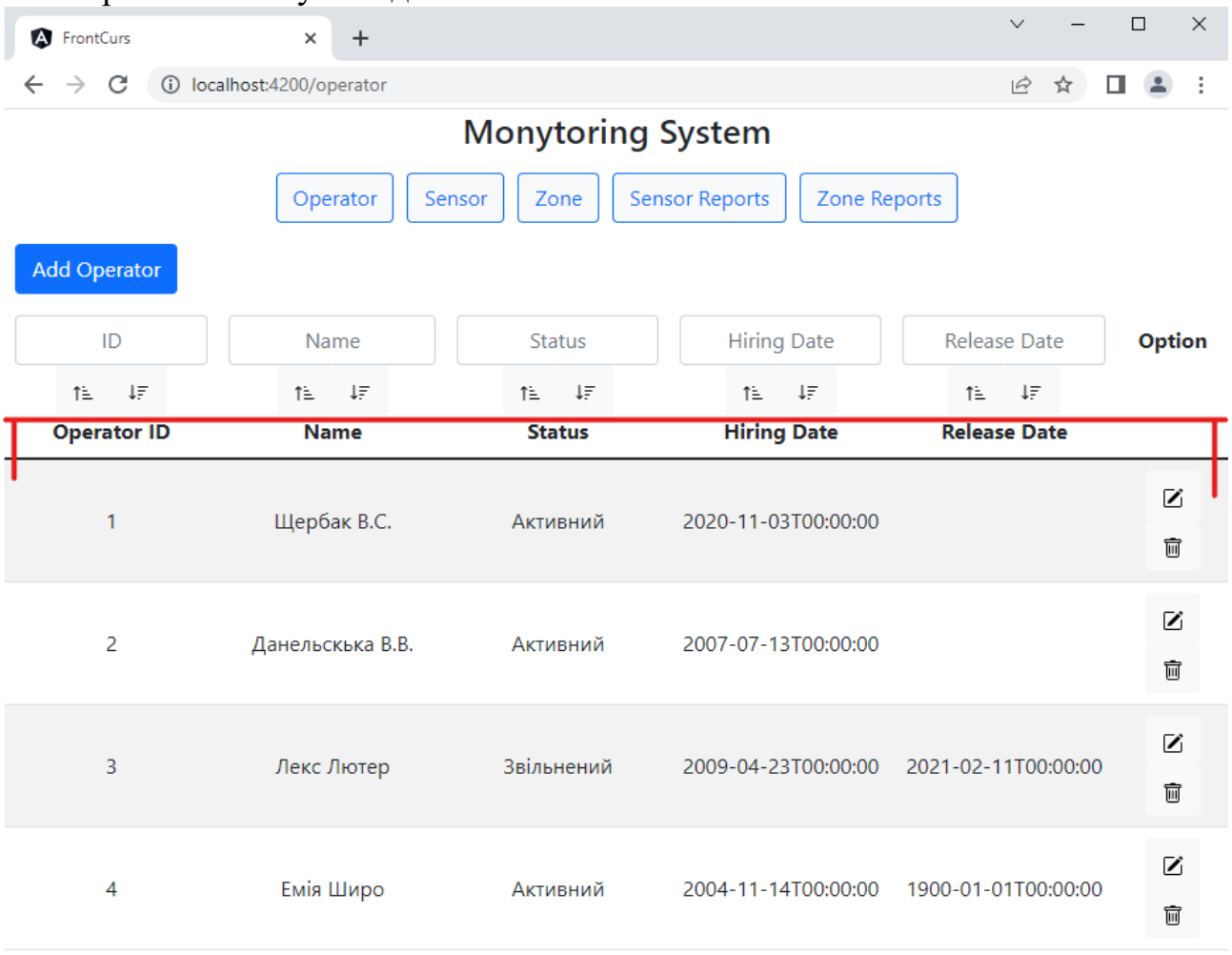


Рисунок 2 – Зона виводу даних

3. Внести нові дані. Для цього слід натиснути на синю кнопку, справ

вверху таблиці. В кожній вкладці кнопка має іншу назву і прив'язку. Але вона завжди на тому місці.

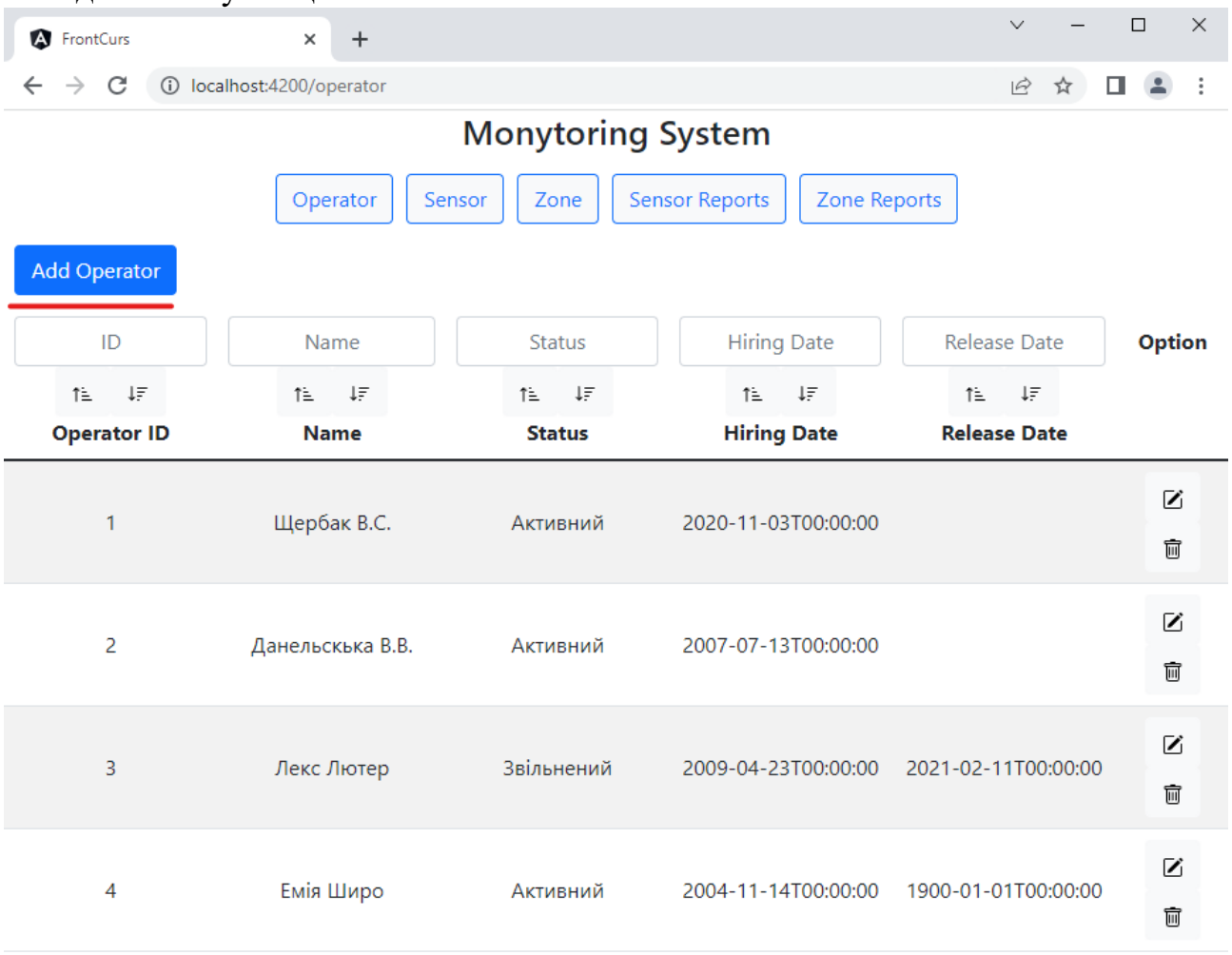


Рисунок 3 – Кнопка внесення даних в БД

4. Слід необхідно заповнити поля в новому вікні і натиснути кнопку «Add».

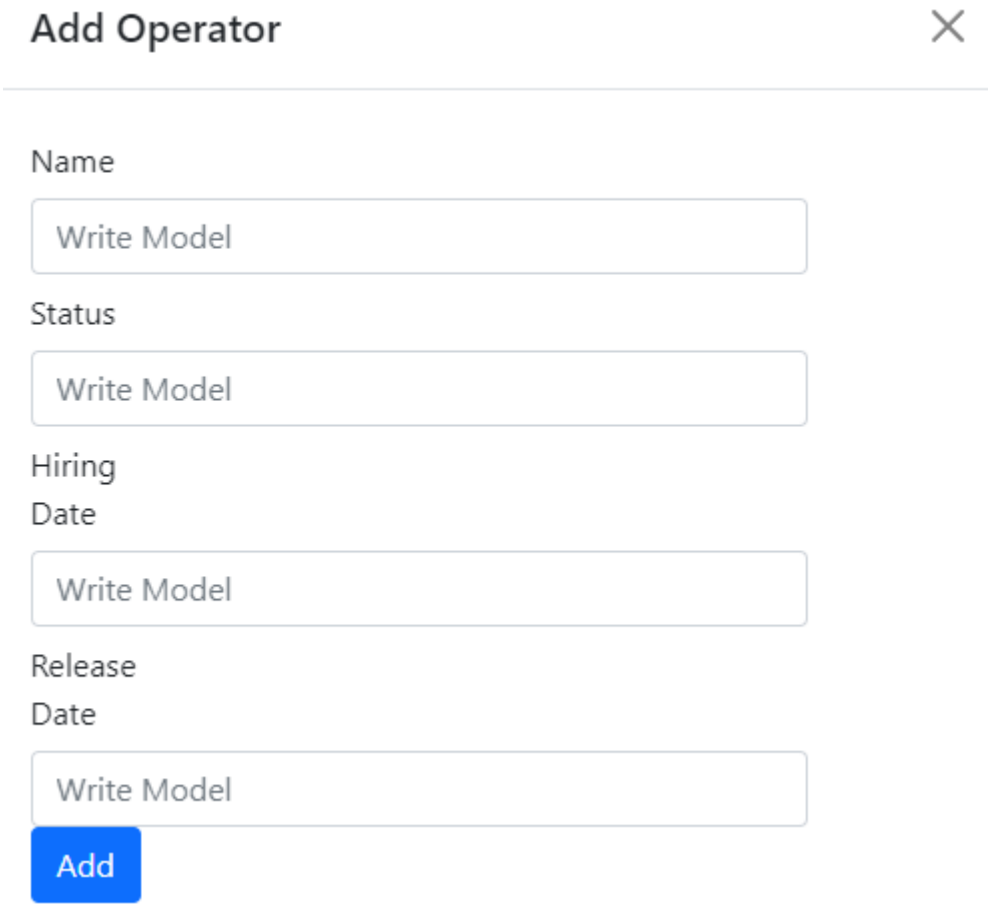


Рисунок 4 – Вікно внесення даних в БД

5. Щоб редагувати запис, необхідно натиснуту на копку з олівцем напроти того запиту, який ви хочете редагувати. Далі як в пункті 4.

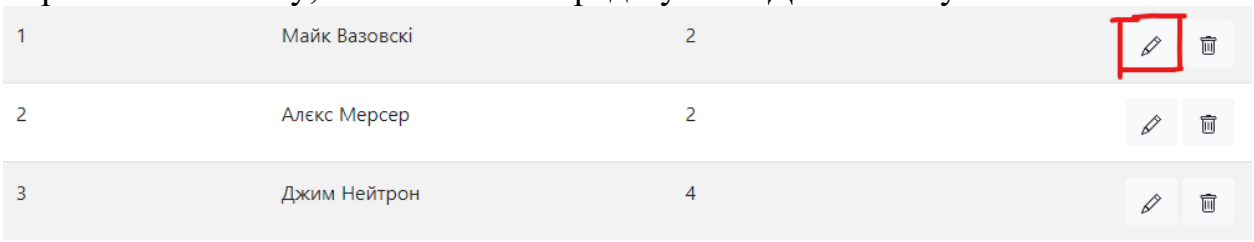


Рисунок 5 – Кнопка відкриття редактора даних

6. Для видалення запису, слід натиснути кнопку с мусорною корзиною напроти непотрібного запису.

Рисунок 6 – Кнопка видалення даних

7. Поля фільтрації записів. Достатньо ввести в любе поле значенні і додато автоматично відфільтрує записи

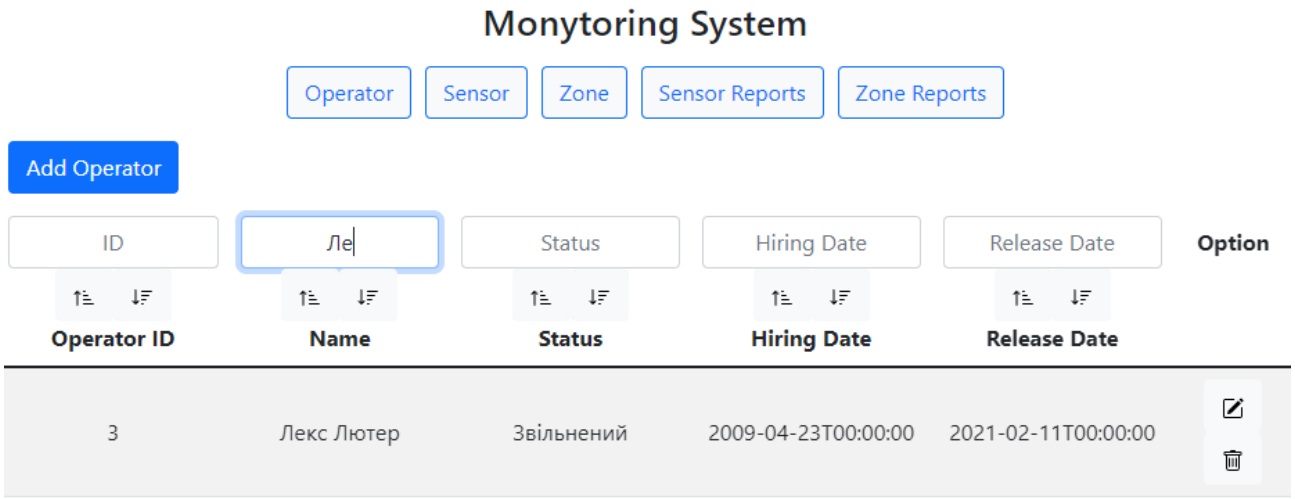


Рисунок 7 – Поля фільтрації

8. Кнопки сортування. Сортують ту колонку над якою знаходяться. Сортування можливо в «вверх» і «вниз» залежно від настигнутої кнопки.

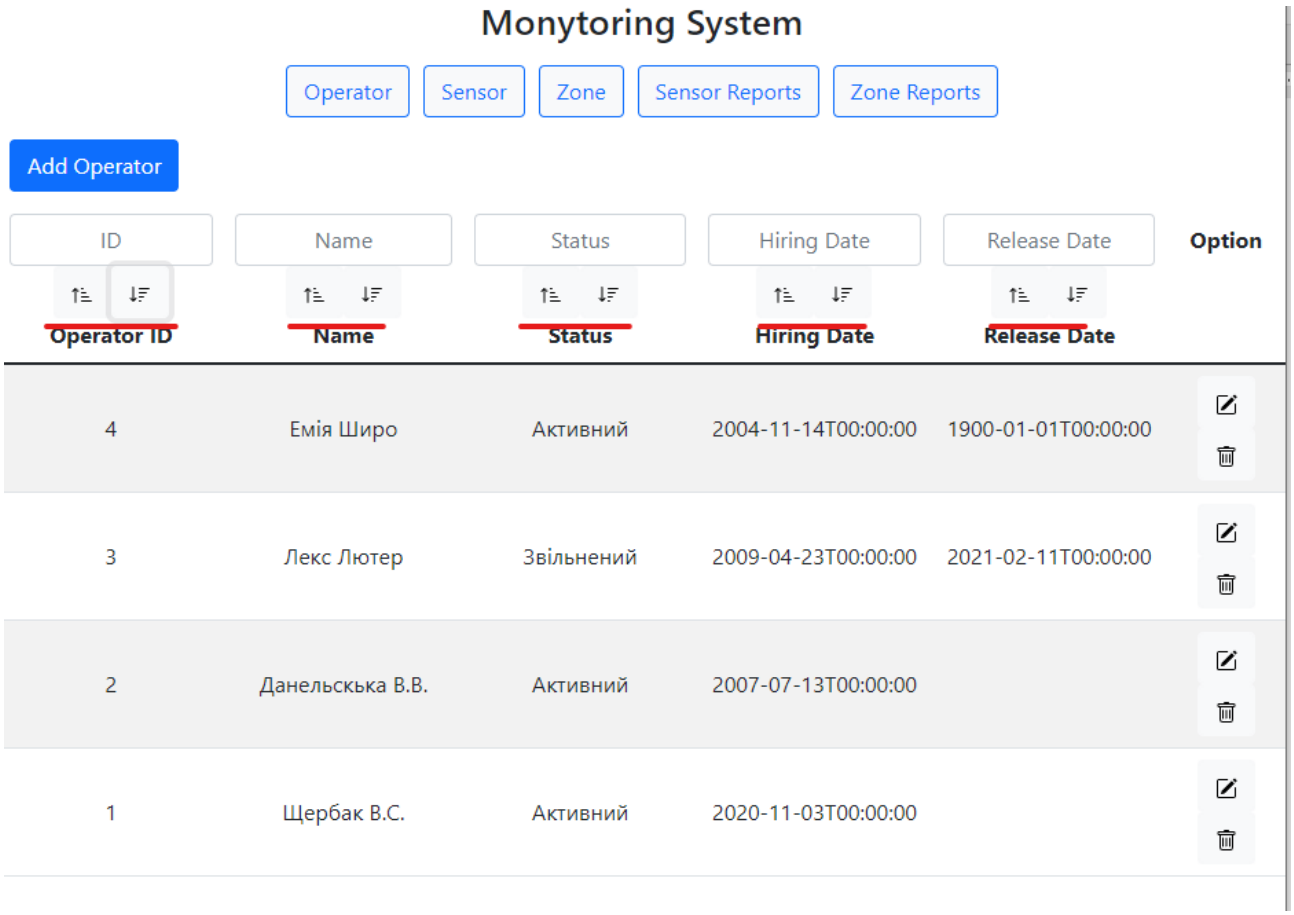


Рисунок 7 – Кнопки сортування

ДОДАТОК Г

Діагарама дерева вузлів

Листів 1

Розробник Щербак В.С.

Керівник Степанов М.М.

Київ - 2022

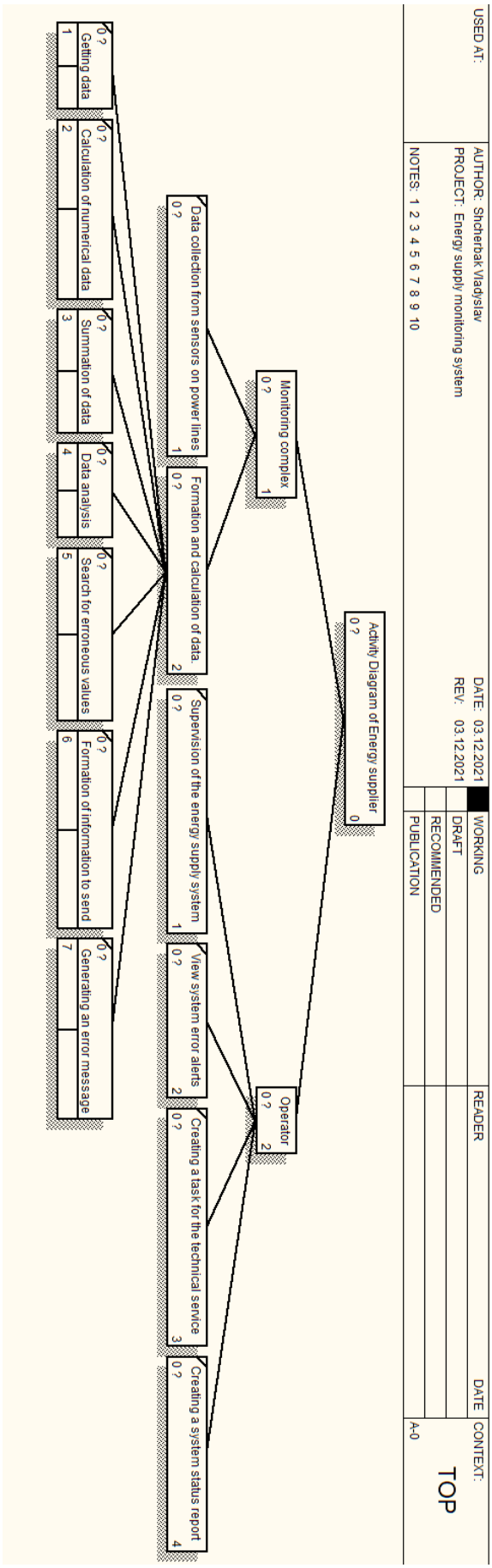


Рисунок 1 – Діаграма дерева вузлів системи моніторингу енергопостачання