КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

До захисту допущено
В.о. завідувача кафедри ІСТ
_____ Олексій КОЛЕСНІКОВ
(підпис) (ім'я, ПРІЗВИЩЕ)
"___"_____2021р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра

зі спеціальності 126 «Інформаційні системи та технології»

освітньої програми «Програмні технології інтернет речей»

на тему: Інформаційно-аналітична система моніторингу електричної мережі м. Мелітополь

Виконав: студент 4 курсу, групи ІР-41

Костянтин БОЙКО _____
(підпис)

Керівник д.т.н, доц. Олексій КОЛЕСНІКОВ _____
(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 _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, Ім'я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань.

Здобувач освіти _____
(підпис)

Київ – 2021 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
В.о. завідувача кафедри,
д.т.н., доцент
Олексій КОЛЕСНІКОВ

_____
«__» _____ 2021 року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувач освіти: Костянтин Бойко

Група: ІР-41

1. **Тема кваліфікаційна робота бакалавра**: «Інформаційно-аналітична система моніторингу електричної мережі м. Мелітополь».
Затверджена протоколом засідання кафедри ІСТ №18/20 від 01.12.2021 року
2. **Строк подання студентом готової роботи** − «26» червня 2021 р.
3. **Вихідні дані до роботи**: дослідження в області моніторингу електромережі; реалізація передачі даних та зберігання їх у базі даних; створення графічного додатку для моніторингу показів.
4. Зміст роботи:

- РОЗДІЛ 1. АНАЛІЗ ТА ОПИС СФЕРИ МОНІТОРИНГУ ЕЛЕКТРОМЕРЕЖІ (аналіз аналогів, основні поняття);
- РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ ТА АРХІТЕКТУРИ СИСТЕМИ (вибір технологій, проектування моделей БД, проектування архітектури системи);
- РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ (розгортання БД, реалізація передачі та зберігання даних, створення графічного інтерфейсу).

5. **Перелік графічного матеріалу**: опис аналогів, моделі бази даних, приклад розумного лічильника, структура рішення, діаграма роботи OpenADR, важливі частини коду, знімки екрану з результатами роботи.

6. **Календарний план виконання роботи:**

Етапи виконання кваліфікаційної роботи бакалавра	**Термін виконання**	**Результат виконання**
1. Вибір тематики кваліфікаційної роботи бакалавра	до 01.12.2020	виконано
2. Наказ про затвердження тем кваліфікаційної роботи бакалавра та призначення керівників	01.12.2020	виконано
3. Розробка плану кваліфікаційної роботи бакалавра і його погодження з керівником	25.12.2020	виконано
4. Написання І розділу кваліфікаційної роботи	20.01.2021	виконано
5. Написання ІІ розділу кваліфікаційної роботи	19.02.2021	виконано
6. Написання ІІІ розділу кваліфікаційної роботи	05.03.2021	виконано
7. Підготовка висновків і пропозицій	05.04.2021	виконано
8. Попередній захист кваліфікаційної роботи	20.04.2021	виконано
9. Перевірка на плагіат	до 15.06.2021	виконано
10. Нормоконтроль	до 17.06.21	виконано
11. Рецензування кваліфікаційної роботи бакалавра і представлення роботи на кафедру в друкованому вигляді	до 21.06.2021	виконано
11. Захист кваліфікаційної роботи бакалавра	23.06.2021	

Дата видачі завдання «_01_» _грудня_ 2020 р.

Керівник роботи: д.т.н, доц. Олексій КОЛЕСНІКОВ_____ (підпис)
Завдання прийняв до виконання:
Здобувач освіти на освітньому рівні «бакалавр» 4-го курсу групи ІР-41
Костянтин БОЙКО _____
(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 Костянтина БОЙКА

Тема роботи: «Інформаційно-аналітична система моніторингу електричної мережі м. Мелітополь».

Мета кваліфікаційної роботи бакалавра – проектування архітектури системи моніторингу електричної мережі. Реалізація передачі та збереження даних. Розробка графічного інтерфейсу для моніторингу.

Об’єкт дослідження – складові розумної електромережі.

Предмет дослідження – проектування системи, що складається з багатьох части, проектування та розгортання бази даних, розробка додатку з графічним користувацьким інтерфейсом для роботи з базою даних та географічними даними.

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

КЛЮЧОВІ СЛОВА: електромережа, лічильник, система моніторингу, база даних, графічний інтерфейс, OpenLEADR, Qt

ABSTRACT

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

Faculty of Information Technologies

Department of Information Systems and Technologies

Educational Program "Software Technologies of the Internet of Things"

Qualification work of master Kostiantyn BOIKO.

Work topic: "Information and analytical system for monitoring the electric network of Melitopol".

The purpose of the bachelor's qualification work is designing the architecture of the electrical network monitoring system. Implementation of data transfer and storage. Development of a graphical user interface for monitoring.

The object of research is components of a smart grid.

The subject of research is designing a system consisting of many parts, designing and deployment of a database, development of an application with a graphical user interface for working with a database and geographic data.

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

KEY WORDS: power grid, meter, monitoring system, database, graphical interface, OpenLEADR, Qt

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна електромережа складається з багатьох елементів, але інформація про її стан у багатьох випадках все ще збирається вручну з великими інтервалами. Результатом цього є повільне реагування при несправностях та фіксовані тарифи для електроенергії. За останні десятиріччя інфокомунікаційні мережі значно розвинулись. В додаток до цього технології виробництва покращились та значно знизили вартість обладнання. Це призвело до розвитку розумних технологій, що знайшли застосунок у різних галузях.

Розумні технології також можуть бути застосовані у сфері електроенергетики. Впровадження розумних лічильників дозволить автоматично збирати покази з короткими інтервалами. Зібрана інформація може мати різні застосування. Вона може бути використана для впровадження тарифів на електроенергію, залежних від часу дня. Це спонукатиме використовувати енергію у непікові години та продавати більше енергії у пікові. Як результат, це призведе до більш ефективного та рівномірного розподілу споживання енергії протягом доби. Інший спосіб полягає у використанні даних для виявлення крадіжок електроенергії та виявлення локацій, де мали місце несправності обладнання. У разі збоїв ця інформація може бути використана для автоматичного перенаправлення потоку енергії для пом'якшення проблем.

Метою даної роботи є комплексне дослідження, моделювання та розробка додатку згідно обумовленої індивідуальної теми “Інформаційно-аналітична система моніторингу електричної мережі м.”Мелітополь””.

Завдання: провести аналіз предметної області, створити моделі бази даних, на основі цього створити додаток для моніторинга смарт-лічильників.

Об’єктом дослідження є складові розумної електромережі.

Предметом дослідження є проектування та програмна реалізація додатку, його функціонування, керування окремими елементами цього додатку.

Практичне значення одержаних результатів. Створено додаток, що містить структуру електромережі та іншу релевантну інформацію. Отриманий додаток дозволяє наочно переглядати стан мережі та взаємодіяти з її елементами.

РОЗДІЛ 1. АНАЛІЗ ТА ОПИС СФЕРИ МОНІТОРИНГУ ЕЛЕКТРОМЕРЕЖІ

1.1 Поняття розумної електромережі

Електрична мережа - це мережа для доставки електроенергії від виробників та місць, де воно виробляється та трансформується (електростанції та підстанції) до кінцевих пунктів, де “споживається” електроенергія: домогосподарства, підприємства, різні споруди та споживач загалом.

Розумна електромережа - це електрична мережа, що забезпечує двосторонній потік електроенергії та даних, завдяки чому розумне вимірювання часто розглядається як перший крок. Розумні мережі - як концепція - стали відомими більше десяти років тому.

Розумна мережа має кілька цілей, і перехід від традиційних електричних мереж до інтелектуальних мереж зумовлений багатьма факторами, включаючи дерегуляцію енергетичного ринку, еволюцію в обліку, зміни рівня виробництва електроенергії, децентралізацію (розподілене виробництво енергії), зміна нормативних актів, зростання мікрогенерації та ізольованих мікромереж (Рис1.1.1) [1].

На практиці це дуже взаємопов'язана мережа з декількома компонентами, такими як підстанції, лінії електропередач та електропроводка, розподільні лінії, трансформатори тощо.

Традиційні електричні мережі майже не мають можливостей зберігання електроенергії, керуються попитом та мають ієрархічну структуру. В електричній мережі напруга поступово знижується, щоб електроенергія могла використовуватися цими різними споживачами: від рівнів напруги передачі до рівнів напруги розподілу до рівнів напруги обслуговування (насправді це одночасно і підвищення, і зниження, і, отже, дещо складніше).

Як правило, розрізняють електромережу (висока та надвисока напруга) та розподільну мережу (нижча напруга), де використовуються різні електропроводки та кабельні системи. Призначення електричної мережі полягає

в тому, щоб забезпечити, щоб електроенергія завжди забезпечувалася, коли і де це необхідно, неперервно і в цьому полягає безліч проблем, де розумна мережа вже може запропонувати рішення та відповіді.

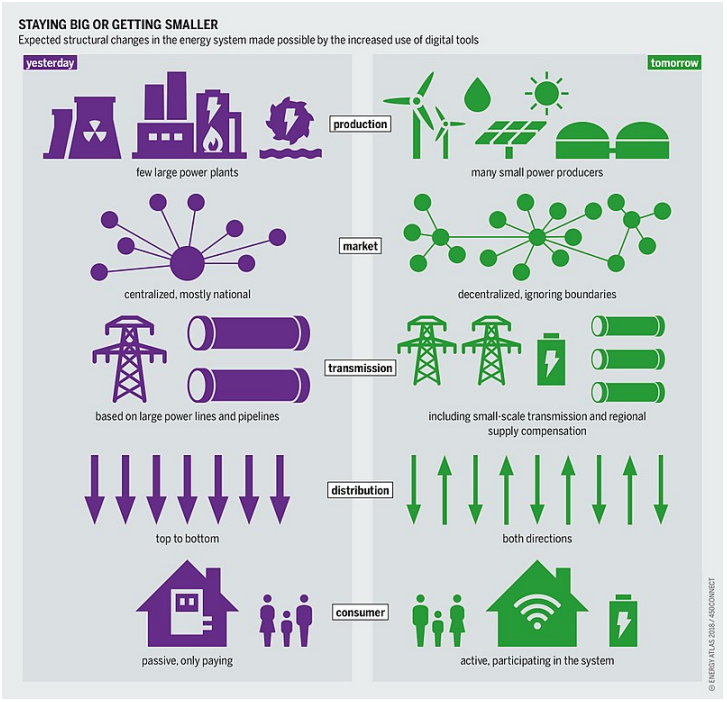


Рисунок 1.1.1 - Ілюстрація сучасних тенденцій розвитку електромережі

Двосторонній потік електроенергії та даних є важливою характеристикою інтелектуальної мережі. Це дозволяє передавати інформацію та дані різним зацікавленим сторонам на ринку електроенергії, які можна проаналізувати для оптимізації мережі, передбачити потенційні проблеми, швидко реагувати при виникненні проблем та нарощувати нові потужності - та послуги.

Як уже згадувалося, одним із перших та, можливо, основних аспектів ініціатив щодо створення інтелектуальних мереж, є вимірювання та так звані розумні лічильники. Розумні лічильники (рис. 1.1.2) - це наступний етап в еволюції, яка розпочалася десятиліття тому і призвела до появи перших технологій інтелектуальних мереж, таких як автоматичне вимірювання та вдосконалена інфраструктура вимірювання, або Advanced Metering Infrastructure (AMI). Розумні лічильники дозволяють записувати таку інформацію, як

споживання електричної енергії, рівні напруги, струм та коефіцієнт потужності. Розумні лічильники зазвичай реєструють енергію майже в режимі реального часу і регулярно подають звіти за короткими інтервалами протягом дня. Ця інформація дозволяє встановити ціну на електроенергію залежно від часу доби [2].

Рисунок 1.1.2 - Приклад розумного лічильника

Мікромережі відіграють важливу роль у побудові низьковуглецевого майбутнього, оскільки вони забезпечують стійкість основної мережі, оптимізують енергетичні витрати, дозволяють розміщувати енергію з відновлюваних джерел, збільшують інтеграцію електричних транспортних засобів та покращують доступність енергії.

1.2 Огляд продуктів

Оскільки системи для моніторингу електромереж не є продуктом для широкої аудиторії, знайти інформацію про такі системи вдається небагато. Частина систем розроблені спеціально для певного регіону і не публікують внутрішню інформацію. Існують деякі загальні системи, що можна застосувати для різних цілей і налаштувати під себе, але вони не викладають багато

інформації у загальний доступ. Вони спілкуються окремо з кожним клієнтом та презентують деталі за потреби.

1.2.1 Политариф-А

Першим знайденим прикладом такого продуту є “Политариф-А”. Це програмне забезпечення виконує заміри, зберігає, обробляє та відображає дані про енергоспоживання. Це ПЗ дозволяє автоматизувати усі процеси обліку електроенергії від збору інформації до виконання аналітики та виставлення рахунків оплати за послуги [3].

Розробники цієї системи зазначають, що практика впровадження та експлуатації різних каналів зв'язку показала, що важко реалізувати бюджетний варіант надійної системи з якої-небудь однієї схеми. Тому в своїх розробках вони зробили упор на змішаних, комбінованих каналах зв'язку, з використанням облікових і передавальних приладів широкої доступності. В цьому випадку логічний канал зв'язку може складатися з частин з різними способами передачі і від різних виробників. Такий підхід має наступні переваги:

- можливість застосування всіх можливих каналів зв'язку в рамках однієї системи, в залежності від ситуації на об'єкті;
- можливість використання приладів обліку різних виробників, в тому числі і в складі однієї системи;
- можливість використання приладів обліку різних виробників, в тому числі і в складі однієї системи;
- немає необхідності у використанні додаткового ретрансляційного обладнання для забезпечення реальної, а не заявленої дальності передачі інформації;
- часто немає необхідності заміни вже існуючих у споживача приладів обліку;
- немає залежності від обладнання одного постачальника, який до того ж може проводиться тільки їм;

1.2.2 WebNMS IoT Platform

WebNMS - це платформа для IoT рішень. На основі цієї платформи компанія створює різні рішення та надає повний пакет, що включає обладнання, програмне забезпечення та датчики. Рішення охоплює рішення від фізичних кінцевих пристроїв до інтерфейсу користувача, забезпечуючи повний та ефективний моніторинг [4].

Компанія присутня лише на ринках США, Індії та Об'єднаних Арабських Еміратів. Рішення схематично зображено на Рис. 1.2.1.

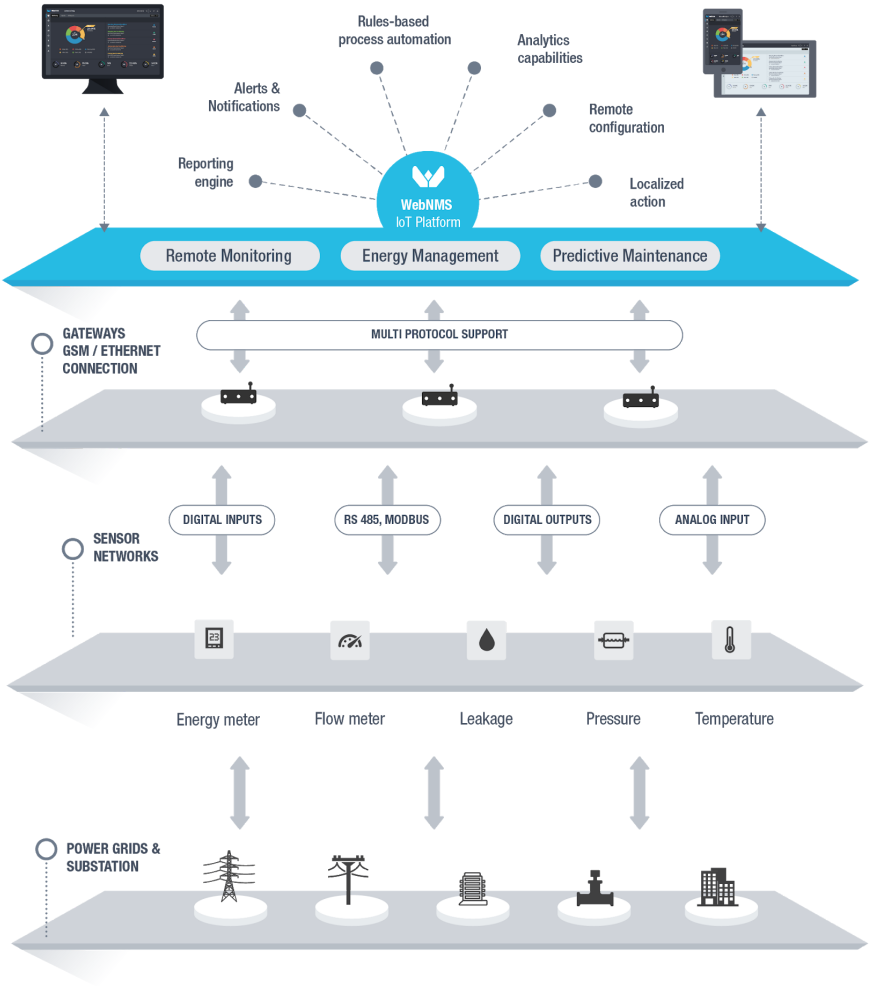


Рисунок 1.2.1 - Структура WebNMS IoT Platform

Це багатофункціональна система, що збирає значення приладів, аналізує та зберігає в хмарі. Доступ до платформи може відбуватись як з мобільного додатку, так і з веб-інтерфейсу. Також рішення дозволяє кінцевим споживачам

ознайомитись зі своїм використанням електроенергії та цінами на пікові години. Це допомагає кінцевим споживачам ефективно планувати своє споживання та уникати розбіжностей у виставлення рахунків.

1.2.3 WinPM.Net

Powermanager - рішення від Siemens, що у поєднанні з лічильниками Siemens та захисними пристроями низької напруги забезпечує комплексне рішення з управління енергією для бізнесу. Система дозволяє вимірювати, обробляти, аналізувати, зберігати інформацією про споживання енергії та статус на всьому вашому підприємстві. Він пропонує можливості управління, а також детальну звітність, яка допоможе вам зменшити витрати на енергію [5].

Серед особливостей системи:

- забезпечення детального аналізу якості електроенергії та накладень сигналів для кореляції міжфазних залежностей між напругами та струмами та каскадними несправностями;
- розумні події та кластеризація тривог для інтуїтивної фільтрації, пошуку та категоризації подій та тривог;
- графічна візуалізація для полегшення аналізу подій, місця та потенціального впливу;
- піддтримка Modbus RTU, Modbus TCP, ION, XML, SNMP, OPC, FTP, та PQDI.

1.3 Постановка задачі

Головною метою даної роботи є розробка системи для збирання та зберігання даних розумних лічильників, їх обробки та візуалізації.

У задачі системи входить наступне:

- Додавання, збереження, та видалення лічильників,
- Додавання, збереження, та видалення базових станцій,
- Наочна візуалізація пристроїв на карті
- Обробка та візуалізація значень розумних лічильників,
- Побудова звітів щодо виробленої та спожитої електроенергії

1.4 Висновки до розділу

У даному розділі проаналізовано й описано напрямок розвитку електромережі, користь покращення системи вимірювання. Проведено аналіз кількох рішень для моніторингу електромереж як на рівні міста, так і на рівні будівель. Системи, що покривають більшу територію використовують менший набір протоколів зв’язку. Це зумовлено необхідністю масштабування та радіусом дії протоколів. Сформульовано постановку задачі.

РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ ТА АРХІТЕКТУРИ СИСТЕМИ

2.1 Вибір технологій

Існують багато стандартів для збору даних лічильників, частина з них ще розвивається або не має загальнодоступної реалізації. Існує також проблема підтримки роботоздатності лічильників при зміні постачальника. Через це більшість виробників не ризикують інвестувати у певний стандарт та використовують TCP\IP стек протоколів.

Для бездротового зв’язку зазвичай використовують LoRaWan або NB-IoT. Ці технології відносяться до LPWA (Low-Power Wide-Area) технологій. Вони працюють на великих відстанях, відносно дешеві та споживають менше енергії. Однак, для підвищення надійності пропускна здатність таких протоколів знижена. Це задовольняє потреби IoT, тому що передача показів та технічної інформації

NB-IoT використовує підмножину стандарту LTE, але обмежує пропускну здатність до однієї вузькосмугової смуги в 200 кГц. Він використовує модуляцію OFDM для зв'язку в низхідній лінії зв'язку та SC-FDMA для зв'язку в висхідній лінії зв'язку [6].

Мережева архітектура LoRaWAN будується в топології зірка-зірка, в якій шлюзи передають повідомлення між кінцевими пристроями та центральним мережевим сервером. Шлюзи підключені до мережевого сервера за допомогою стандартних IP-з'єднань і діють як прозорий міст, просто перетворюючи RF-пакети в IP-пакети і навпаки. Бездротовий зв’язок використовує характеристики далекого діапазону фізичного рівня LoRa, дозволяючи односкачковий зв’язок між кінцевим пристроєм та одним або багатьма шлюзами. Усі пристрої здатні здійснювати двонаправлений зв’язок, і є підтримка груп багатоадресної адресації.

Заявлений радіус роботи LoRaWAN складає 10-15 км. Вони можуть покрити велику кількість кінцевих пристроїв, що робить їх встановлення більш актуальним у місцях з високою щільністю пристроїв. У віддалених регіонах

більш раціональним рішенням може бути використання NB-IoT, оскільки базові станції мобільних операторів вже покрили велику кількість території.

Обробка пакетів з кінцевих пристроїв відбувається на одному або декількох серверах додатків. На рівні додатків використовують різні протоколи, частина з яких пропрієтарна. Для подальшого розглядання обрано OpenADR, оскільки перша його версія була створена відносно давно і цей протокол стабілізувався. У свободному доступі можна знайти документацію та імплементацію цього протоколу, що дозволяє створити додаток з його використанням.

OpenADR протокол, створений організацією OpenADR Alliance. Альянс OpenADR був створений для стандартизації, автоматизації та спрощення реагування на попит електроенергії, щоб дозволити комунальним підприємствам та агрегаторам економічно ефективно управляти зростаючим попитом на енергію та децентралізованим виробництвом енергії, а споживачам контролювати своє енергетичне майбутнє. OpenADR - це відкрита, високозахищена та двостороння модель обміну інформацією та стандарт Smart Grid [8].

OpenADR розділяє учасників обміну повідомленнями на 2 групи:

VTN (Virtual Top Node) відіграє роль серверу. Цей вузол керує ресурсами, створює та передає повідомлення, запитує звіти

VEN (Virtual End Node) відіграє роль клієнта. Цей вузол отримує та обробляє повідомлення, генерує звіти.

На Рис.2.1.1 зображено порядок дій при встановленні з’єднання між VTN та VEN.

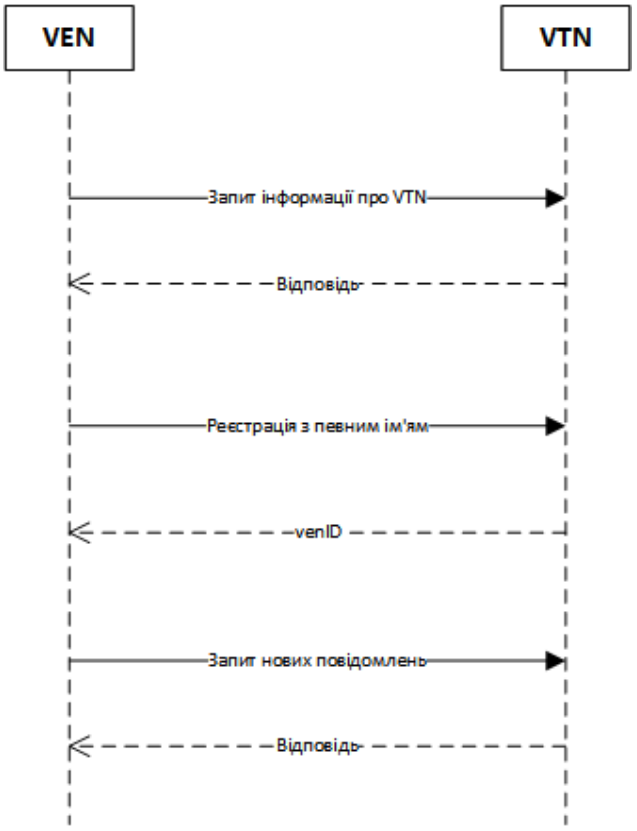


Рисунок 2.1.1 - Встановлення з’єднання OpenADR

З’єднання ініціює VEN, роблячи запит інформації про VTN за певною IP адресою. Далі клієнт надсилає інформацію для реєстрації та якщо сервер підтверджує реєстрацію, він повертає ідентифікатор клієнта. Після успішної реєстрації клієнт періодично робить запити нових повідомлень від сервера.

На Рис.2.1.2 зображено процес звітування. Спочатку клієнт надсилає серверу види звітів, що він може надсилати: напруга, спожита енергія, сила току, тощо. Разом з цим серверу надаються можливі періоди звітування та інша технічна інформація. Сервер вибирає необхідні звіти серед пропонованих та обирає період звітування. Після цього клієнт надсилає звіти з обраною періодичністю.

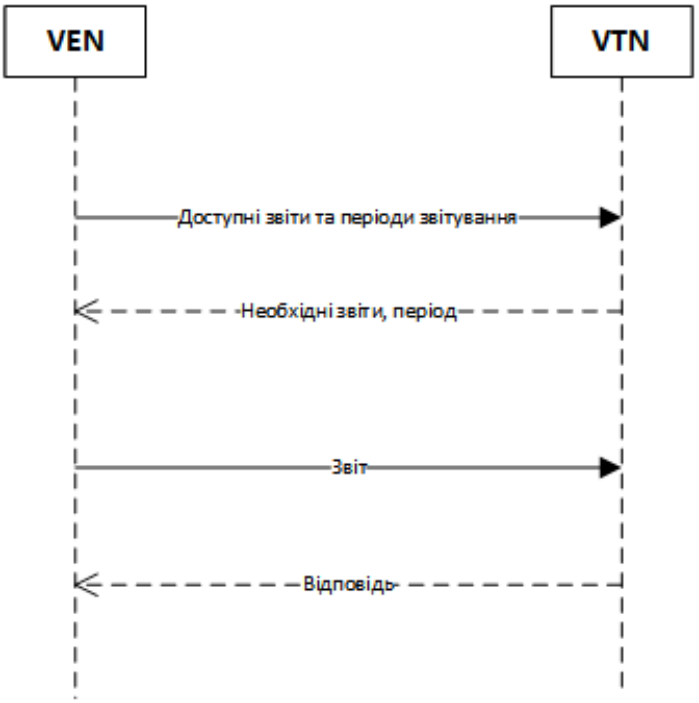


Рисунок 2.1.2 - Процес реєстрації звіті та періодичності

OpenLEADR - це реалізація OpenADR для Python 3. Цей модуль надає повністю сумісну реаліцію OpenADR 2.0b, дозволяє створювати VTN та VEN без реалізації багатьох рутинних операцій. Цей модуль побудований на асинхронній моделі: користувач створює корутини, що будуть обробляти певні події та викликаються асинхронно. Ця реалізація притримується стилю Python: усі повідомлення автоматично перетворюються з XML формату у прості словники Python [9].

Окрім цього є можливість надати сертифікат і ключ у форматі PEM, усі вихідні повідомлення будуть підписані криптографічно. Якщо також надати fingerprint для посвідчення іншої сторони, вхідні повідомлення можна надійно автентифікувати та перевірити.

Існує багато мов програмування і фреймворків для розробки графічного інтерфейсу. Для розробки додатку відбувався вибір серед мовою програмування Java і фреймворком Swing та мовою програмування C++ і фреймворком Qt5. Вибір відбувався між цими стеками технологій через попередній досвід роботи

з ними. Вивчення нового стеку технологій може зайняти багато часу, що не залишить часу на виконання завдання курсової роботи.

Swing було відкинуто, оскільки цей фреймворк доволі старий, бракує певного функціоналу. Значною перевагою Qt для виконання поставленої задачі є підтримка роботи з географічними картами.

Qt розширює можливості програміста за допомогою набору макросів, метаінформації, сигнально-слотових з'єднань та інших засобів. При цьому використовуються засоби мови C++. Qt є сумісний з усіма поширеними сучасними її компіляторами. Разом з тим велика кількість платформ, що підтримуються, робить Qt досить універсальним інструментом, який стає можливо використовувати для найрізноманітніших задач [11].

Qt має два фреймворка для створення користувацького інтерфейсу: QtWidgets та QtQuick. QtWidgets - більш старий модуль, що надає стандартний функціонал віджетів для створення десктопних додатків. Він надає C++ API для роботи з ним та можливість задання структури віджетів у окремих XML файлах.

QtQuick - більш новий модуль, для створення кросплатформенного користувацького інтерфейсу. Цей модуль використовує власну мову QML та вставки Javascript для розробки додатку і дає можливість розширення засобами C++.

QML - це специфікація користувацького інтерфейсу та декларативна мова, яка була розроблена, щоб забезпечити динамічне зв’язування компонентів, і вона дозволяє легко повторно використовувати та налаштовувати компоненти в інтерфейсі користувача [12]. Використовуючи модуль QtQuick, дизайнери та розробники можуть легко створювати плавні анімовані користувальницькі інтерфейси в QML і мають можливість підключення цих користувацьких інтерфейсів до будь-яких внутрішніх бібліотек C ++.

2.2 Проектування концептуальної моделі

Концептуальна модель - модель предметної області, що складається з переліку пов'язаних понять, які використовуються для опису області, разом з

властивостями і характеристиками, класифікацією цих понять, за видами, ситуацій, ознаками в даній області і алгоритмів протікання процесів в ній [13].

Для обраної теми та завдання була створена модель типу “Сутність - Зв’язок”, яка визначає основні елементи, що приймають участь при вимірюванні споживання електроенергії та як вони зв'язані між собою (рис. 2.2.1).

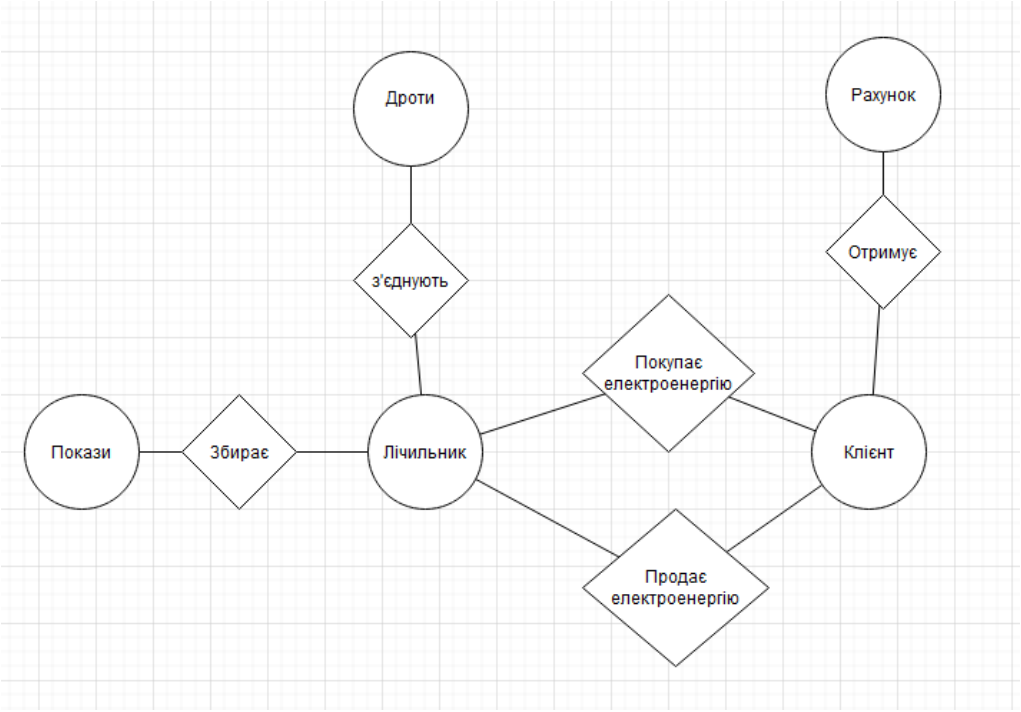


Рисунок 2.2.1 - Модель “Сутність зв’язок” для об'єктів задіяних в вимірюванні показів

Так, центральним об’єктом є лічильник. Він збирає покази по споживанню електроенергії. Ці лічильники, як і інші частини електромережі з’єднані між собою за допомогою дротів. Їх відносини визначають структуру електромережі. Головна ціль мережі - це обслуговування клієнтів, для цього лічильник може використовуватись для виміру як спожитої електроенергії так і виробленої. На основі показів, клієнт отримує рахунок.

2.3 Проектування логічної моделі

Для побудови логічної моделі будемо використовувати програму Data Modeler від компанії Oracle. Це програмне забезпечення надає можливість

створювати об’єкти, визначати їх атрибути, їх параметри, такі як тип, обов’язковість наявності, визначати ключі та зв'язки між об’єктами. Створена логічна модель (рис. 2.3.1). Символ “*” біля атрибутів означає обов'язковість цього атрибуту. Штрихована частина зв’язку індикує необов’язковість цього зв'язку . “Куряча лапка” означає зв'язок 1:N.

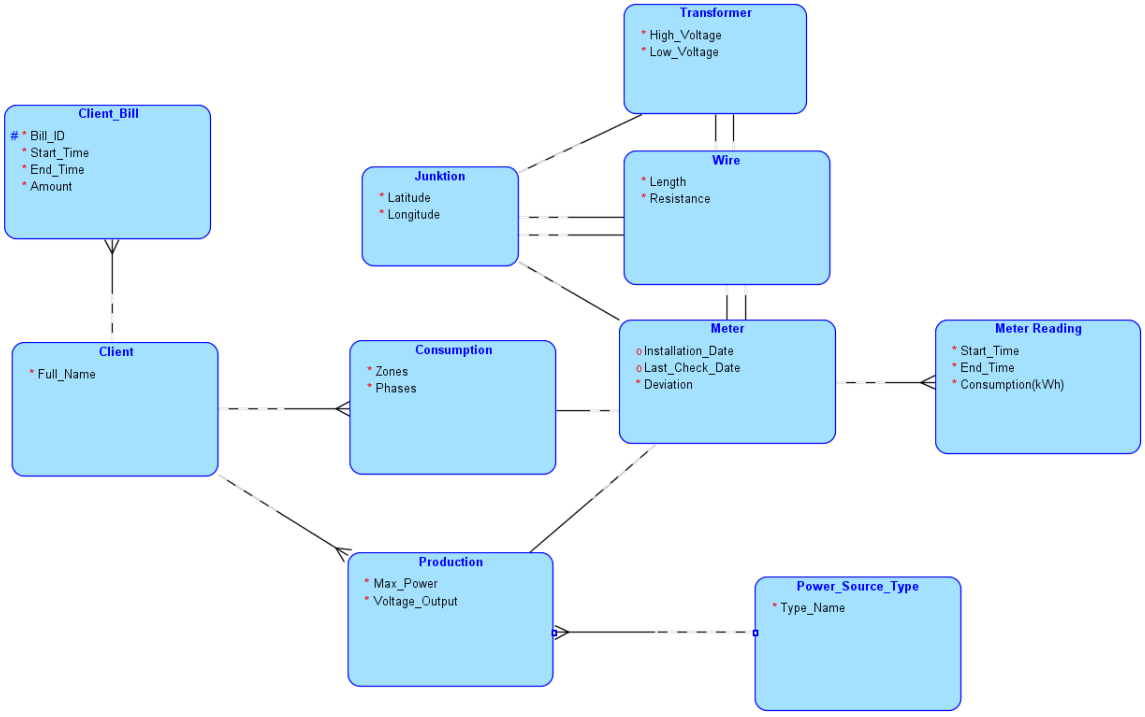


Рисунок 2.3.1 - Логічна модель бази даних системи моніторингу

Дана модель містить 10 сутностей, які мають наступні атрибути та зв’язки:

- Junction: дана таблиця міститеме з’єднання дротів. Це вершина в топології мережі. Інші елементи мережі, такі як розумні лічильники можуть бути прикріплені до цих об’єктів.

- Transformer: Змінює напругу. Вказує на Junction, до якого прикріплений.

- Meter: Лічильник. Вимірює пройдену крізь нього електроенергію. Вказує на Junction, до якого прикріплений.. Зберігає дати встановлення та останньої перевірки. Має певну похибку.

- Meter_Reading: Зберігає записи лічильників - проміжок часу запису та спожиту енергію. Операцією ділення можна вирахувати середню потужність за цей час.

- Client: таблиця клієнтів. Кожен клієнт має Ім’я. інші дві таблиці зв’язують клієнта з лічильниками у якості виробника або споживача електроенергії.

- Consumption: таблиця споживачів. Вказує кількість фаз та зон, що надаються споживачу та з'єднує клієнта з його лічильником.

- Production: таблиця постачальників. Вказує напругу, тип джерела, максимальну потужність та з'єднує клієнта з його лічильником.

- Power_Source_Type: таблиця типів джерел електроенергії

- Client_Bill: Таблиця рахунків. Встановлює суму для клієнта на основі показників лічильників.

Отже, ми визначили важливі сутності, їх атрибути та зв’язки між ними. Далі необхідно отримати фізичну модель, що буде описувати специфічні деталі моделі, такі як зовнішні ключі, обмеження, індекси тощо.

2.4 Проектування фізичної моделі

Фізична модель даних - це модель даних, яка описує те як дані насправді зберігаються в БД. Ця модель містить інформацію про всі таблиці стовпці цих таблиць. Специфікація складається з назви таблиці, імені стовпців, тип даних стовпців, визначення первинних ключів та встановлює зв'язок між різними таблицями за допомогою зовнішніх ключів.

Засобами Oracle Data Modeler з правильно налаштованої логічної моделі можна отримати фізичну модель. Ця функція доступна у контекстному меню при натисканні «Engineer to Relational Model». За допомогою цього функціоналу була згенерована фізична модель. Після цього вона була відкоригована та прийняла вигляд, зображений на Рис. 2.4.1.

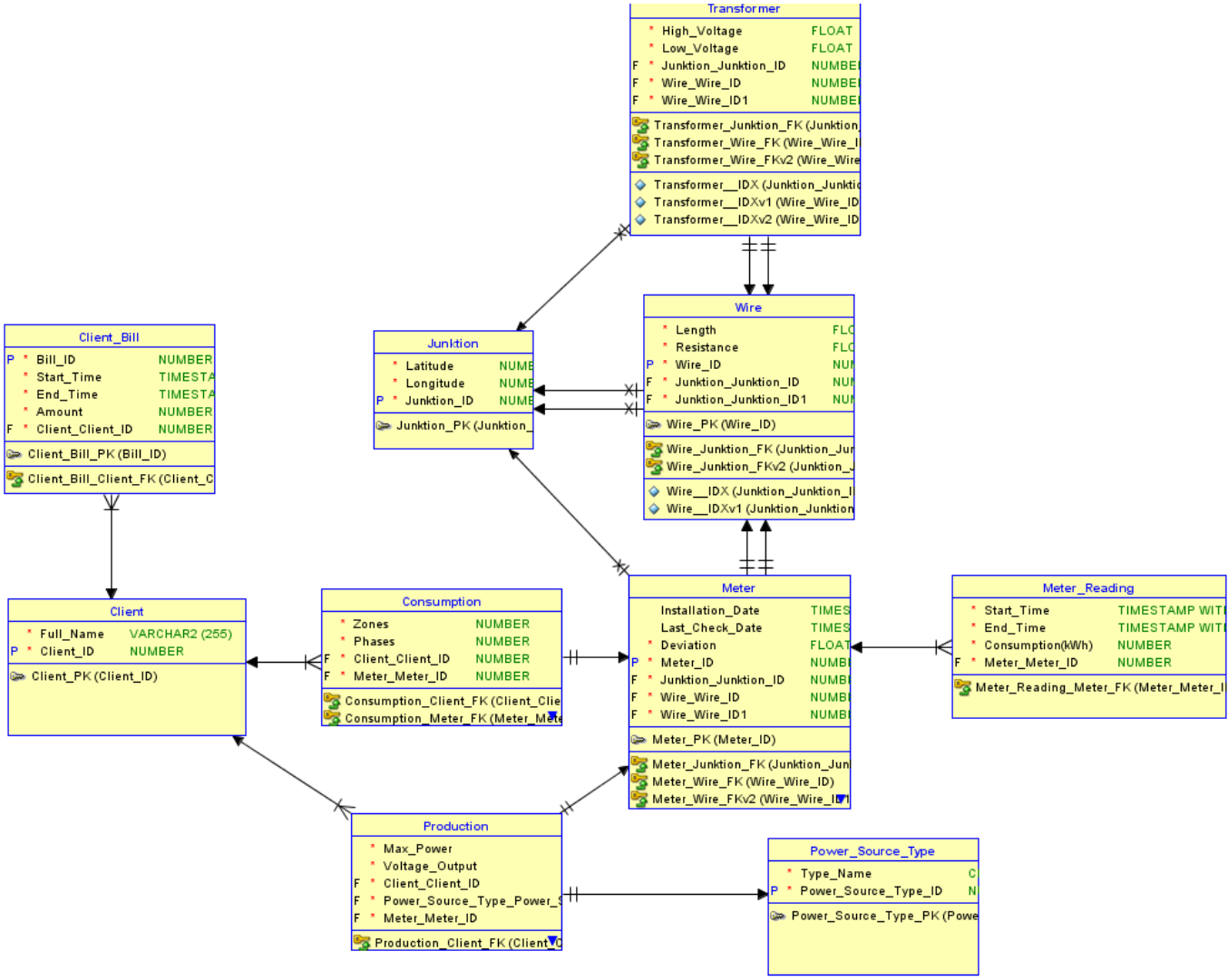


Рис. 2.4.1 - Отримана фізична модель системи моніторингу

У результаті були створені таблиці, визначені типи даних та додані поля для однозначної ідентифікації об’єктів (Див. Таблиця 2.4.1).

Таблиця 2.4.1 – **Таблиці, атрибути та їх типи даних**

Назва таблиці	**Стовпець**	**Тип даних**
1	**2**	**3**

Продовження табл. 2.4.1

Junction	Id(Primary key)	Integer
	Latitude	Binary float
	Longitude	Binary float
Meter	Id(Primary key, foreign key)	Integer
	installation_date	Timestamp
	last_check_data	Timestamp
	deviation	Binary float
	wire1_id(Foreign key)	Integer
	wire2_id(Foreign key)	Integer
meter_reading	Id(primary key)	Integer
	start_time	Timestamp
	end_time	Timestamp
	meter_id(Foreign key)	Integer
Client	id(primary key)	Integer
	full_name	Varchar(255)
Consumption	id(primary key)	Integer
	zones	Integer
	phases	Integer
	meter_id(Foreign key)	Integer

Продовження табл. 2.4.1

1	2	3
	client_id(Foreign key)	Integer
Client_bill	id(primary key)	Integer
	start_time	Timestamp
	end_time	Timestamp
	amount	Binary float
	client_id(Foreign key)	Integer
Power_source_type	id(primary key)	Integer
	type_name	Varchar(20)
Production	id(primary key)	Integer
	max_power	Binary float
	voltage_output	Binary float
	client_id(Foreign key)	Integer
	power_type_id	Integer
Transformer	Id(Primary key, foreign key)	Integer
	high_voltage	Binary float
	low_voltage	Binary float
	wire1_id(Foreign key)	Integer
	wire2_id(Foreign key)	Integer

2.5 Опис структури рішення

У результаті було спроектовано рішення для системи моніторингу розумних лічильників. Система була розбита на окремі компоненти та було визначено зв’язки між цими компонентами (рис. 2.5.1). Збір даних буде відбуватись використовуючи протокол OpenADR, де VEN - це лічильник, а VTN - сервер, що буде обробляти значення та зберігати їх у базі даних. У якості СУБД було обрано PostgreSQL. Друга частина системи - це графічний додаток, який дозволить додавати та взаємодіяти з лічильниками, переглядати покази. Для створення додатку обрано фреймворк Qt та технологію QML. Для роботи з мапою обрано OpenStreetMap.

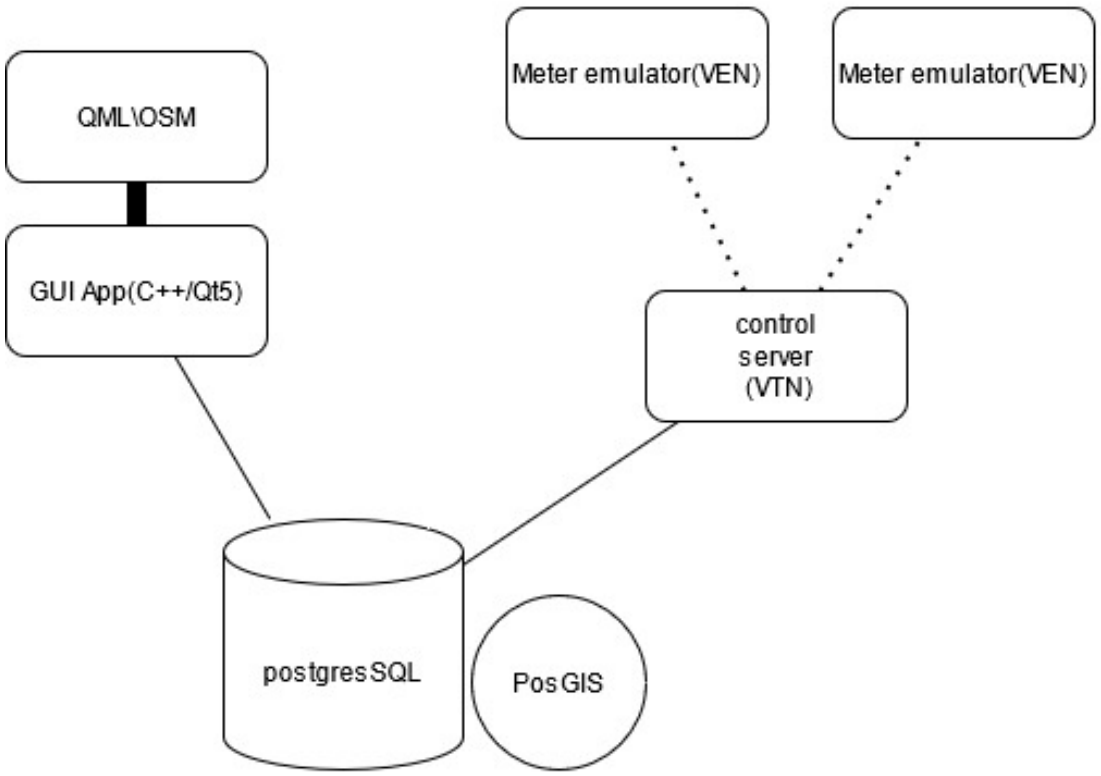


Рис. 2.5.1 - Фінальна структура системи моніторингу

2.6 Висновки до розділу

Отже було розглянуто технології, що будуть використовуватись. За допомогою Oracle Data Modeler були створені логічна та фізична моделі бази

даних. Ці моделі будуть використані при створені додатку. Для цих таблиць буде створено відповінні структури даних, якими буде оперувати розроблене програмне забезпечення. Спроектовано структуру системи для моніторингу.

РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1. Розгортання БД

Для легкості розгортання та подальшим керуванням оточення було вирішено використовувати Docker. Docker - це платформа для контейнеризації з відкритим кодом. Docker дозволяє розробникам пакувати програми в контейнери - стандартизовані виконувані компоненти, що поєднують код програми з всіма бібліотеками операційної системи та залежностями, необхідними для запуску коду в будь-якому середовищі [14].

Було встановлено Docker та використано його компонент docker-compose. Для цього створено файл конфігурації, у якому вказано необхідний образ для postgres, налаштовані змінні оточення та вказано відображення портів для підключення до БД (рис. 3.1.1).

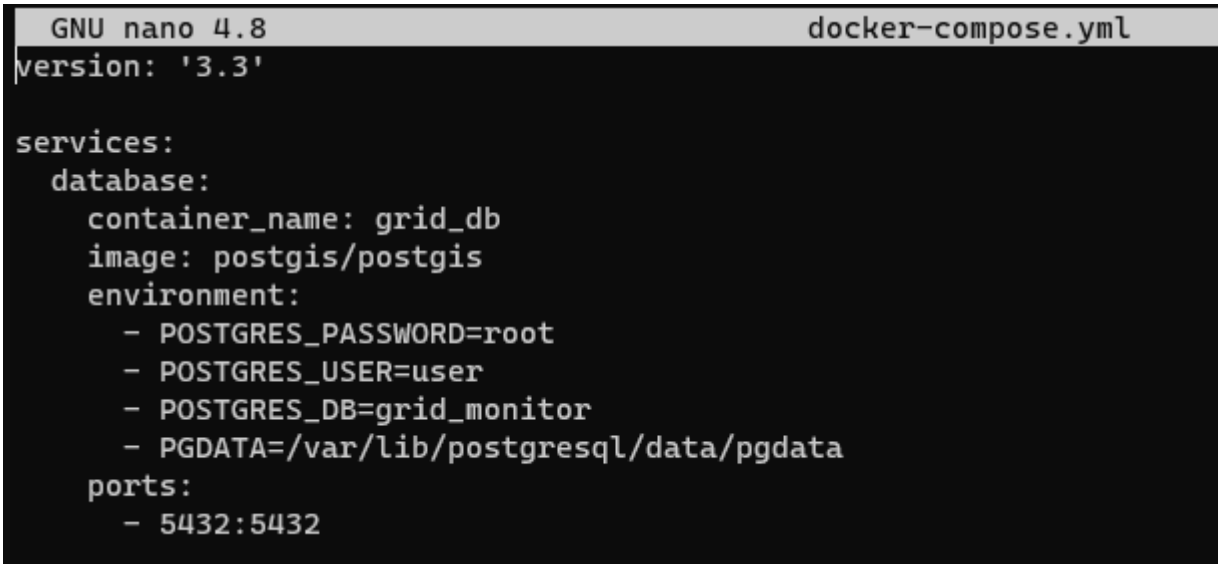


Рисунок 3.1.1 - зміст docker-compose.yml

Для запуску контейнера на основі цієї конфігурації треба виконати наступний рядок:

docker-compose -f docker-compose.yml up -d

Після цього при необхідності вручну взаємодіяти з базою даних можна приєднатись до неї у цьому контейнері за допомогою наступного рядка:

docker exec -it grid_db psql -U user -W grid_monitor

Після цього було створено таблиці для зберігання зареєстрованих VEN та звітів, що вони надіслали.

3.2 Реалізація VTN

Для налаштування підключення до бази даних необхідну інформацію було винесено в окремий файл config.ini (рис. 3.2.1). Це є гарною практикою відокремлювати логіку від даних.

```
 GNU nano 4.8                                                    config.ini
[postgresql]
host=localhost
database=grid_monitor
user=user
password=root
```

Рисунок 3.2.1 - Зміст файлу config.ini

Для зчитування та парсингу було використано бібліотеку *configparser*

Створено функцію, що використовує цю бібліотеку для зчитування конфігураційного файлу та повертає словник для зручності подальшого використання (рис.3.2.2).

```
def get_config(filename, section):
    parser = ConfigParser()
    parser.read(filename)

    dict = {}
    if parser.has_section(section):
        params = parser.items(section)
        for key, val in params:
            db[key] = val
    else:S
        raise Exception(f'Section {section} not found in {filename}')
    return db
```

Рисунок 3.2.2 - Функція для отримання словника з конфігураційного файла

VTN обробляє звіти від VEN та зберігає записи у базі даних. Для роботи з базою даних у Python було обрано бібліотеку *psycopg2.* Використовуючи зчитані конфіги відкривається з’єднання з PostgreSQL (рис.3.2.3).

```
db_config = get_config(filename='config.ini', section='postgresql')
conn = psycopg2.connect(**db_config)
```

Рисунок 3.2.3 - Встановлення з’єднання з базою даних

Для подальшої взаємодії з таблицями створено необхідні функції, що виконують SQL запити (рис.3.2.4).

```
def find_ven(name):
    cur = conn.cursor()
    cur.execute(f'SELECT id, name FROM ven WHERE name=\'{name}\';')
    row = cur.fetchone()
    cur.close()
    return row

def insert_ven(name):
    cur = conn.cursor()
    cur.execute('INSERT INTO ven(name) VALUES(%s) RETURNING id;', (name,))
    id = cur.fetchone()[0]
    conn.commit()
    cur.close()
    return id

def find_meter(ven_id, name):
    cur = conn.cursor()
    cur.execute(f'SELECT id, ven_id, name FROM smart_meter WHERE ven_id=\'{ven_id}\' AND name=\'{name}\';')
    row = cur.fetchone()
    cur.close()
    return row

def insert_meter(ven_id, name):
    cur = conn.cursor()
    cur.execute('INSERT INTO smart_meter(ven_id, name) VALUES(%s, %s) RETURNING id;', (ven_id, name,))
    id = cur.fetchone()[0]
    conn.commit()
    cur.close()
    return id

def insert_report(meter_id, datetime, total_energy=None, power=None, voltage=None):
    cur = conn.cursor()
    cur.execute('INSERT INTO meter_report(meter_id, datetime, voltage, consumed_energy, current_power) VALUES(%s, %s, %s, %s, %s) RETURNING id;', (meter_id, datetime, vol
    id = cur.fetchone()[0]
    conn.commit()
    cur.close()
    return id
```

Рисунок 3.2.4 - Частина функцій для роботи з таблицями БД

Для роботи з OpenLEADR необхідно створити декілька функцій. Перша функція - *on_register_client* (рис. 3.2.5). Вона буде викликатись коли певний VEN буде намагатись встановити з’єднання з VTN.

```
async def register_client(registration_info):
    print(registration_info)
    ven_name = registration_info['ven_name']
    result = find_ven(ven_name)

    if result:
        id, name = result
        return id, 'group'
    else:
        id = insert_ven(ven_name)
        return id, 'group'
```

Рисунок 3.2.5 - Реалізація функції для реєстрації VEN

Наступна функція - on_register_report (рис. 3.2.6). Вона буде викликатись коли VEN надасть перелік доступних звітів. Для кожного з видів буде створено окремий виклик з певними параметрами. Якщо сервер зацікавлений в звіті

необхідно повернути колбек, що буде обробляти звіт та інтервал з яким необхідно надсилати звіти.

```
async def on_register_report(ven_id, resource_id, measurement, unit, scale,
                             min_sampling_interval, max_sampling_interval, bundling=1, futures=None, receive_futures=None):
    result = find_meter(ven_id, resource_id)
    if result:
        id, ven_id, name = result
    else:
        id = insert_meter(ven_id, resource_id)

    print('resource_id', id)
    if measurement == 'Power':
        return partial(store_power, id=id), min_sampling_interval
    if measurement == 'Voltage':
        return partial(store_voltage, id=id), min_sampling_interval
    if measurement == 'Total Energy':
        return partial(store_energy, id=id), min_sampling_interval
```

Рисунок 3.2.6 - Реалізація функції для реєстрації звітів

Реалізувавши ці функції, за допомогою OpenLEADR створено та запущено сервер (рис.3.2.7). OpenLEADR побудований на асинхронній моделі та використовує модуль asyncio. Зареєстровані функції викликаються асинхронно за потреби. Після всіх налаштувань контроль передається циклу подій, що буде обробляти події сервера.

```
server = OpenADRServer(vtn_id='myvtn')
server.add_handler('on_create_party_registration', on_register_client)
server.add_handler('on_register_report', on_register_report)


loop = asyncio.get_event_loop()
loop.create_task(server.run_async())
loop.run_forever()
```

Рисунок 3.2.7 - Створення та запуск VTN

OpenADR також дозволяє створювати події на стороні VTN. Ці події надсилаються певним VEN, клієнт вирішує чи треба відреагувати на події та надсилає відповідь OptIn або OptOut.

Створено функцію, що надішле подію вимкнення або ввімкнення постачання певному VEN та колбек що обробить відповідь клієнта (рис. 3.2.8.).

```python
async def event_callback(ven_id, event_id, opt_status):
    print(f"VEN {ven_id} responded {opt_status} to event {event_id}")

def send_enabled_event(server, ven, status):
    server.add_event(ven_id=ven,
                     signal_name='enabled',
                     signal_type='level',
                     intervals=[{'dtstart': now,
                                 'signal_payload': status}],
                     callback=event_response_callback)
```

Рисунок 3.2.8 - Створення події

3.3 Реалізація VEN

Створення VEN значно легше за VTN. Для цього необхідно створити методи, що будуть викликатись для отримання значень для звітів. Оскільки розробка відбувається тільки на прикладному рівні, ці значення емулюються. Навантаження генеруються як косинус від часу (рис. 3.3.1).

```python
import multiprocessing
manager = multiprocessing.Manager()

async def read_voltage():
    value = 220 + math.sin(time.time()/20) * 5
    return value

async def read_power(namespace, period, phase):
    value = 1000 + math.cos(time.time()/period + phase) * 1000
    if not enabled:
        value = 0
    namespace.total_energy += convert_to_hours(sampling_rate) * value
    print('total energy', namespace.total_energy)
    return value

async def read_energy(namespace):
    return namespace.total_energy
```

Рисунок 3.3.1 - Функції для генерування значень

Для створення клієнту(VEN) необхідно вказати адресу сервера. Після цього треба лише зареєструвати види звітів. Для цього ми вказуємо функції створені вище, та інформацію про звіт (рис. 3.3.2.).

```
client = OpenADRClient(ven_name='myven',
                       vtn_url='http://127.0.0.1:8080/OpenADR2/Simple/2.0b')

client.add_report(callback=read_voltage,
                  report_specifier_id='VoltageReport',
                  resource_id='smart_meter1',
                  measurement='Voltage',
                  sampling_rate=sampling_rate,
                  unit='V')

client.add_report(callback=read_power,
                  report_specifier_id='PowerReport',
                  resource_id='smart_meter1',
                  measurement='Power',
                  sampling_rate=sampling_rate,
                  unit='W')

client.add_report(callback=read_energy,
                  report_specifier_id='EnergyReport',
                  resource_id='smart_meter1',
                  measurement='Total Energy',
                  sampling_rate=sampling_rate,
                  unit='Wh')


loop = asyncio.get_event_loop()
loop.create_task(client.run())
loop.run_forever()
```

Рисунок 3.3.2 - Створення VEN та реєстрація звітів

Для обробки подій необхідно створити та зареєструвати відповідний колбек (рис 3.3.3). Оскільки ми лише емулюємо пристрій, відбувається простий перезапис глобальної змінної

```
async def on_event(event):
    first_signal = event['event_signals'][0]
    intervals = first_signal['intervals']
    enabled = intervals['signal_payload']
    return 'optIn'

client.add_handler('on_event', on_event)
```

Рисунок 3.3.3 - Створення та реєстрація обробника подій

Після реалізації клієнтської і серверної частини необхідно перевірити їх роботу. Запуск обох частин видає на сервері результат (рис. 3.3.4).

Рисунок 3.3.4 - Результат роботи VTP

Вище видно успішний запуск сервера, після чого було оброблено нове з’єднання від клієнта, та зареєстровано 3 типа звітів.

3.4 Налаштування системи збирання

Для автоматизації збору проекту використано CMake. CMake є вільним міжплатформеним програмним забезпеченням із відкритим кодом для автоматизації збирання, тестування, упаковки та встановлення програмного забезпечення за допомогою незалежного від компілятора методу. CMake не є системою збірки, а генерує файли збірки іншої системи. Він підтримує ієрархію каталогів та програми, які залежать від декількох бібліотек. Він має мінімальні залежності, вимагаючи лише компілятора C ++ у власній системі збирання. Qt Creator підтримує роботу з CMake та синхронізується зі структурою проектів, що зконфігуровані у цій системі.

Для створення CMake конфігурації необхідно у корні директорії створити файл під назвою “CMakeLists.txt”. Для створення нового проекту було в корені нової директорії створено файл конфігурації (рис. 3.4.1).

```
cmake_minimum_required(VERSION 3.14)

set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)

project(GridMonitor LANGUAGES CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_subdirectory(src)
```

Рисунок 3.4.1 Файл CMake конфігурації

В цьому файлі вказана версія CMake, що використовується, конфігуруються директорії, де будуть створюватись та зберігатись артефакти компіляції, вказано назву проекту, мову, що використовується, та версію стандарту. Для розробки буде використовуватись останній доступний стандарт C++17. Інша частина конфігурації винесена у піддиректорії.

Проект має структуру, зображену на Рис. 3.4.2

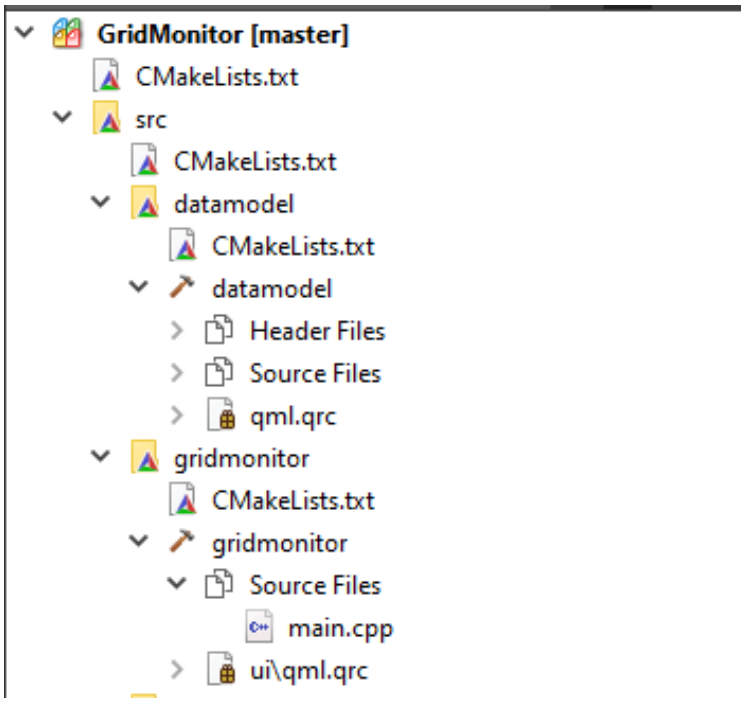


Рис. 3.4.2. Структура проекту

Гарною практикою є відокремлення даних від логіки роботи застосунку. Це дозволяє легше змінювати окремі модулі та уникає створення зворотних залежностей. Було створено окремий проект “datamodel” в якому буде знаходитись необхідний функціонал для роботи з базою даних. Цей проект буде збиратись як окрема статична бібліотека, що чітко розділяє відповідальність модуля та ускладнює створення зайвих залежностей. Другий проект під назвою “gridmonitor” буде містить у собі реалізацію графічного інтерфейсу користувача. До цього проекта буде лінкуватись статична бібліотека зібрана на основі першого проекту, що дозволить легко працювати з БД.

Для спрощення процесу лінкування “datamodel” засобами CMake генерується хедер, що містить макрос для експорту класів або функції з бібліотеки (рис. 3.4.3).

```
5   #ifdef DATAMODEL_STATIC_DEFINE
6   #  define DATAMODEL_EXPORT
7   #  define DATAMODEL_NO_EXPORT
8   #else
9   #  ifndef DATAMODEL_EXPORT
10  #    ifdef datamodel_EXPORTS
11          /* We are building this library */
12  #      define DATAMODEL_EXPORT __declspec(dllexport)
13  #    else
14          /* We are using this library */
15  #      define DATAMODEL_EXPORT __declspec(dllimport)
16  #    endif
17  #  endif
```

Рисунок 3.4.3. Макрос для експорту статичної бібліотеки

Цей макрос необхідно використати у файлах заголовку бібліотеки перед об’явленням необхідного класу чи функції. Він розкриється по-різному в залежності від того в якому проекті буде включено цей файл заголовку.

3.5 Реалізація моделей даних

Для роботи з об’єктами, що не відстежуються фреймворком Qt використано C++, оскільки QML дозволяє працювати лише з експонованими

об'єктами. Для початку роботи з базою даних в додатку необхідно встановити з’єднання. Для цього було створено класс Database, що використовує QSqlDatabase для цих цілей. Встановлення підключення реалізовано у методі void connectToDataBase() (рис. 3.5.1).

```
void Database::connectToDataBase()
{
    db = QSqlDatabase::addDatabase("QPSQL");
    db.setHostName("127.0.0.1");
    db.setDatabaseName("grid_monitor");
    db.setUserName("user");
    db.setPassword("root");
    bool ok = db.open();

    if (ok)
    {
        QFile test(":/create.sql");
        test.open(QIODevice::ReadOnly);
        QString s(test.readAll());
        QSqlQuery query(s);
        while (query.next())
        {
            qWarning() << query.value(0).toString();
            qWarning() << query.value(1).toString();
        }

        qDebug() << query.lastError().text();
    }
}
```

Рисунок 3.5.1 - Підключення до БД

При створенні екземпляру QSqlDatabase необхідно вказати плагін, який містить необхідну реалізацію для роботи з СУБД. В нашому випадку це QPSQL. Цей плагін поставляється разом з Qt5 як вже зібрана динамічна бібліотека. Для його використання необхідно помістити відповідні файли динамічних бібліотек поряд з виконуваним файлом. Це відбувається засобами CMake після збору проекту.

Програмам потрібно формувати дані та відображати дані. Qt Quick має поняття моделей, представлень та делегатів для відображення даних (рис. 3.5.2).

Вони модулюють візуалізацію даних, щоб дати розробнику або дизайнеру контроль над різними аспектами даних. Розробник може поміняти місцями різні види представлень з невеликими змінами даних. Аналогічним чином, інкапсуляція екземпляру даних у делегат дозволяє розробнику диктувати спосіб подання даних або обробки даних.

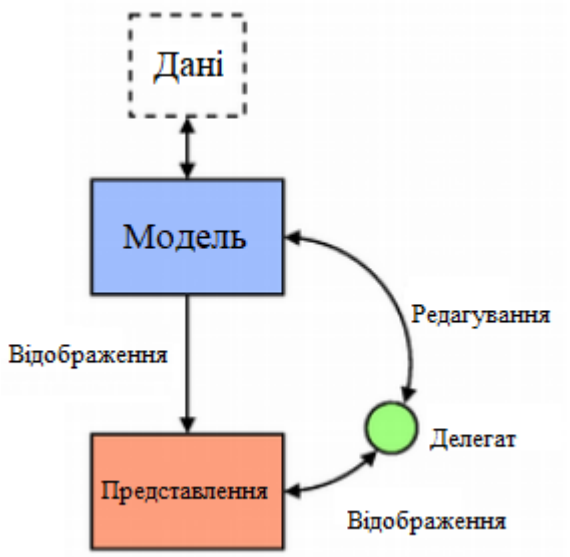


Рисунок 3.5.2 - Model-View шаблон

- Модель - містить дані та їх структуру. Існує кілька типів QML для створення моделей.
- Представлення - контейнер, що відображає дані. У поданні можуть відображатися дані у списку або сітці.
- Делегат - диктує, як дані повинні відображатися у представленні . Делегат бере всі дані в моделі та інкапсулює їх. Дані доступні через делегата. Делегат може також записати дані назад у редаговані моделі [15].

Для роботи з цим фреймворком в Qt необхідно реалізовувати відповідні інтерфейси. Для моделей даних це QAbstractItemModel. Модель даних, що лежить в основі, виставляється для представлення та делегування як ієрархія таблиць. Якщо не використовується ієрархія, то модель являє собою просту таблицю рядків і стовпців. Кожен елемент має унікальний індекс, визначений QModelIndex.

Для цілей роботи з базою даних найбільш підходить QSqlQueryModel, що наслідується від QAbstractItemModel. QSqlQueryModel - це інтерфейс високого рівня для виконання запитів SQL та обходу набору результатів. Він побудований поверх QSqlQuery нижчого рівня і може використовуватися для надання даних для перегляду класів.

Наслідуючись від QSqlQueryModel, було створено необхідні моделі, що будуть надавати інтерфейс для роботи з базою даних (рис. 3.5.3).

```
#pragma once

#include "datamodel_export.h"
#include <QSqlQueryModel>
#include <QGeoCoordinate>

class DATAMODEL_EXPORT MeterModel : public QSqlQueryModel
{
    Q_OBJECT

public:
    explicit MeterModel(QObject *parent = nullptr);

    enum Roles {
        IdRole = Qt::UserRole + 1,      // id
        LocationRole,
        ClientIdRole = LocationRole + 2,
        MeterIdRole,
        ConnectionTypeRole
    };

    QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const override;
    Q_INVOKABLE QVariantMap get(int row);
protected:
    QHash<int, QByteArray> roleNames() const override;

public slots:
    void addMeter(QGeoCoordinate i_coordinates, int client_id, int meter_id, int connection_type);
    void remove(int i_index);
    void updateModel();
};
```

Рисунок 3.5.3 - Об’явлення класу моделі та його методів

Було визначено ролі, що відповідають полям таблиці. Перевизначено методи *data* та *roleNames*. Це дозволить отримувати необхідні значення з моделі. Також було визначено методи, для редагування даних у таблиці. Для цього використано два механізма для виставлення методів у мета-фреймворк Qt5. Першим механізмом були “слоти”. За їх задумкою вони не повертають значення, тому методи, які мають повернути певний об’єкт, оголошені з використанням макросу Q_INVOKABLE, який теж виставляє методи у мета-фреймворк. Після цього ці методи будуть доступні з QML.

Важливо було перевизначити *roleNames* (рис. 3.5.4).

```
46  QHash<int, QByteArray> MeterModel::roleNames() const {
47      QHash<int, QByteArray> roles;
48      roles[IdRole] = "id";
49      roles[LocationRole] = "location";
50      roles[ClientIdRole] = "client_id";
51      roles[MeterIdRole] = "meter_id";
52      roles[ConnectionTypeRole] = "connection_type";
53
54      return roles;
55  }
```

Рисунок 3.5.4 - Визначення ролей

Цей метод повертає словник, де визначена відповідність ролей до їх текстового представлення. Це дозволить мати простий та прямий доступ до полей моделі засобами QML.

QSqlQueryModel інкапсулює у собі виконання запиту до бази даних та обробку відповіді. Для отримання значень з таблиці достатньо вказати SQL запит.

this->setQuery("SELECT id, ST_X(location), ST_Y(location), client_id, meter_id, connection_type FROM junction NATURAL JOIN meter WHERE subtype='meter';");

Для отримання необхідних даних необхідно перевизначити метод data (рис. 3.5.5). Цей метод повертає QVariant оскільки тип даних залежить від індекса та ролі. Індекс визначає рядок у таблиці, де знаходиться необхідна інформація. Роль вказує на атрибут. Модель зберігає дані у вигляді таблиці, тому нам необхідно вирахувати необхідний стовпчик на основі ролі. Для географічних координат знадобилось реалізувати окрему обробку, оскільки була необхідність два поля таблиці перетворити у одне значення типу QGeoCoordinate.

```
QVariant MeterModel::data(const QModelIndex &index, int role) const
{
    if (!index.isValid())
        return QVariant();

    if(role == LocationRole)
    {
        QModelIndex x_index = this->index(index.row(), 1);
        QModelIndex y_index = this->index(index.row(), 2);
        double x = QSqlQueryModel::data(x_index).toDouble();
        double y = QSqlQueryModel::data(y_index).toDouble();
        return QVariant::fromValue(QGeoCoordinate(x, y));
    }

    int columnId = role - Qt::UserRole - 1;
    QModelIndex modelIndex = this->index(index.row(), columnId);

    return QSqlQueryModel::data(modelIndex, Qt::DisplayRole);
}
```

Рисунок 3.5.5 Реалізація отримання даних з моделі

При роботі з моделями, що зберігають час, вказуємо необхідний часовий пояс. Оскільки OpenLEADR передає час для 0 часового поясу було вирішено також працювати з ним і надалі. Для цього перед поверненням значення явно вказується часовий пояс (рис. 3.5.6).

```
auto val = QSqlQueryModel::data(index, role);
auto dt = val.toDateTime();
if(!dt.isNull())
{
    dt.setTimeSpec(Qt::UTC);
    return dt;
}
return val;
```

Рисунок 3.5.6 - Вказування часового поясу

Оскільки записів у таблицях показів лічильників дуже багато, запити для таких таблиць повертають обмежену кількість запитів. Крім цього при звітуванні з маленьким періодом кількість записів дуже велика. Для вирішення цієї проблеми було згруповано запити за певним інтервалом та взято середнє значення.

SELECT t.meter_id, t.datetime, avg FROM meter_energy_reading t INNER JOIN (SELECT MAX(datetime) as datetime, AVG(consumed_energy) FROM meter_energy_reading

GROUP BY DIV(CAST(EXTRACT(EPOCH FROM datetime) AS Integer), %1)) m ON t.datetime = m.datetime ORDER BY(t.datetime) DESC LIMIT 300;

3.6 Створення додатку з графічним інтерфейсом

Для створення графічного додатку було обрано QtQuick. Більша частина додатку буде визначатись QML файлами. Для запуску самого додатку необхідно засобами C++ створити екземпляр QApplication та передати йому екземпляр QQmlApplicationEngine (рис.3.6.1).

```
104 int main(int argc, char **argv)
105 {
106
107 #if QT_VERSION < QT_VERSION_CHECK(6, 0, 0)
108     QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
109 #endif
110     QApplication app(argc, argv);
111
112     QQmlApplicationEngine engine;
113     const QUrl url(QStringLiteral("qrc:///main.qml"));
114     QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
115                      &app, [url](QObject *obj, const QUrl &objUrl) {
116         if (!obj && url == objUrl)
117             QCoreApplication::exit(-1);
118     }, Qt::QueuedConnection);
119
120     Database database;
121     database.connectToDataBase();
122
123     addModels(engine);
124
125     engine.load(url);
126
127     return app.exec();
128 }
```

Рисунок 3.6.1 - Точка входу у програму

QQmlApplicationEngine проаналізує зміст QML файлів та створить на основі цього необхідні об’єкти та графічний інтерфейс. Движок також дозволяє зв’язати об’єкти визначені в QML файлах з об’єктами, що створені у C++. Це використано у функції addModels, де створюються моделі даних, реалізовані у попередньому підрозділі, та зв’язуються з відповідними псевдонімами, доступними з QML (рис. 3.6.2).

```
void addModels(QQmlApplicationEngine& engine)
{
    static ClientModel clientModel;
    static ClientTypeModel clientTypeModel;
    static ConnectionTypeModel connectionTypeModel;
    static JunctionModel junctionModel;
    static LorawanStationModel lorawanStationModel;
    static MeterModel meterModel;

    static MeterEnergyReadingModel meterEnergyReadingModel;
    static MeterPowerReadingModel meterPowerReadingModel;
    static MeterVoltageReadingModel meterVoltageReadingModel;

    static QSortFilterProxyModel meterProxyModel;
    meterProxyModel.setSourceModel(&meterModel);
    meterProxyModel.setFilterRole(MeterModel::Roles::ClientIdRole);

    engine.rootContext()->setContextProperty("clientModel", &clientModel);
    engine.rootContext()->setContextProperty("clientTypeModel", &clientTypeModel);
    engine.rootContext()->setContextProperty("connectionTypeModel", &connectionTypeModel);
    engine.rootContext()->setContextProperty("junctionModel", &junctionModel);
    engine.rootContext()->setContextProperty("lorawanStationModel", &lorawanStationModel);
    engine.rootContext()->setContextProperty("meterModel", &meterModel);
    engine.rootContext()->setContextProperty("meterProxyModel", &meterProxyModel);

    engine.rootContext()->setContextProperty("meterEnergyReadingModel", &meterEnergyReadingModel);
    engine.rootContext()->setContextProperty("meterPowerReadingModel", &meterPowerReadingModel);
    engine.rootContext()->setContextProperty("meterVoltageReadingModel", &meterVoltageReadingModel);
}
```

Рисунок 3.6.2 - Зв’язування об’єктів

Аналіз QML файлів починається з main.qml (рис. 3.6.3). У цьому файлі описується зміст головного вікна програми. Незалежні частини було винесено у відповідні компоненти, такі як GridMap, ClientView, GlobalStatisticsView, та винесено у окремі файли.

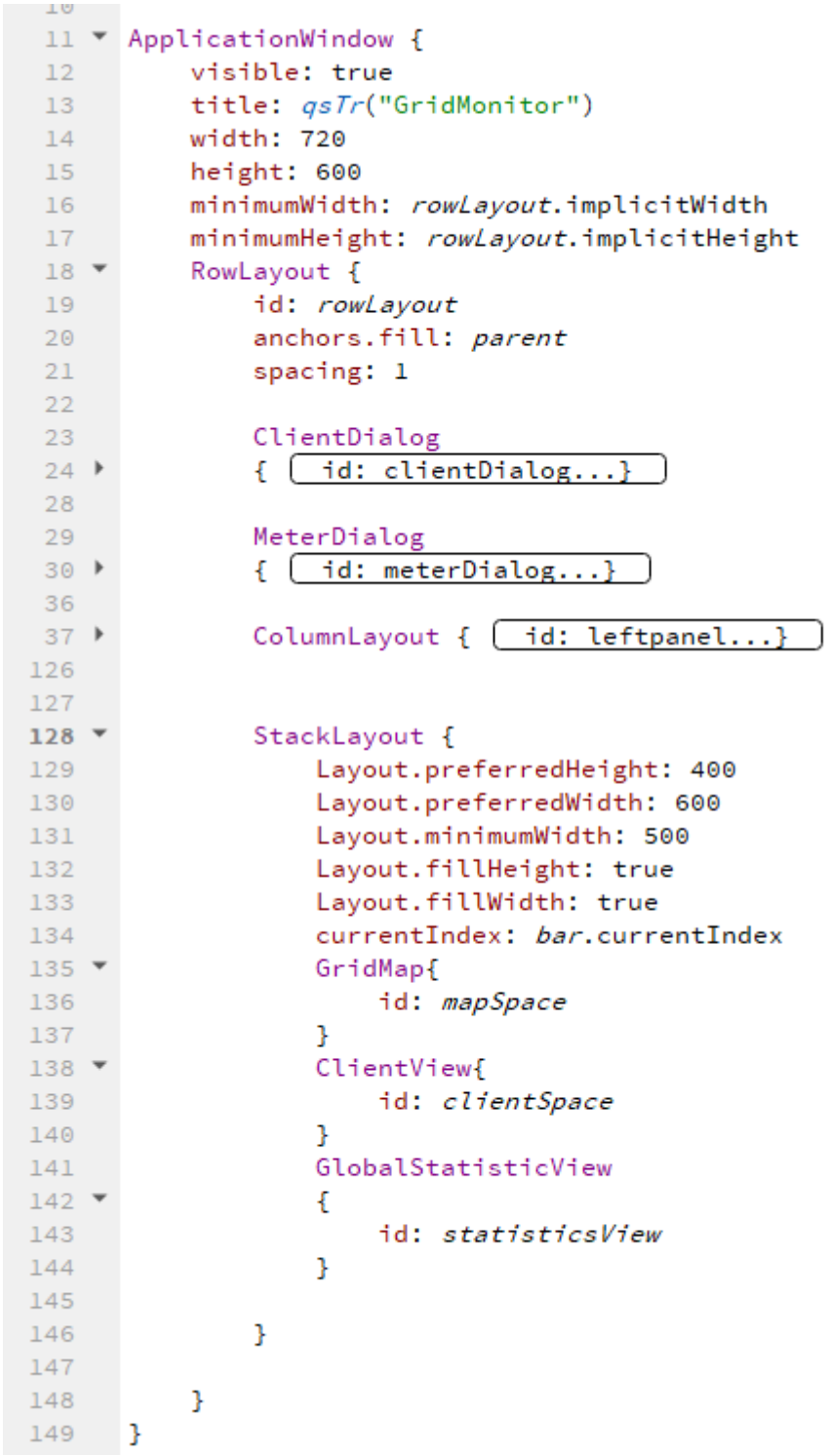


Рисунок 3.6.3 - Структура головного вікна додатку

Середа розробки QtCreator також надає графічний редактор для роботи з QML файлами (рис. 3.6.4).

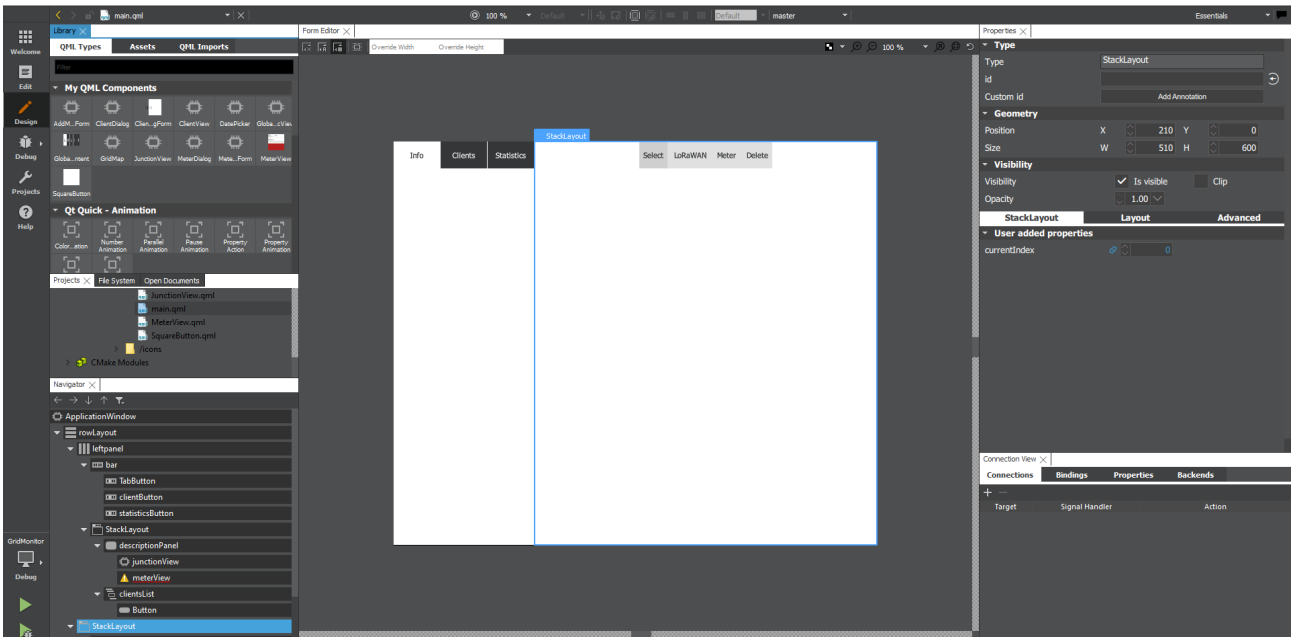


Рисунок 3.6.4 - Графічний редактор QML

У нижній лівій частині редактора зображена ієрархія компонентів. Вище знаходяться структура файлів та доступні для використання компоненти. У центральній частині зображено попередній вигляд редагованого компоненту. Правіше знаходиться редактор властивостей компонента та компонування.

Інтерфейс додатку було розділено на 2 основні частини:

- ліва панель, де буде виконуватись навігація по додатку;
- робоча зона, де буде знаходитись основний контент.

За замовчуванням в робочій зоні буде відображатись мапа з лічильниками та базовими станціями. Для цього використано компонент Map, що використовується для відображення карти або зображення Землі, з можливістю також відображати інтерактивні об'єкти, прив'язані до поверхні карти (рис. 3.6.5). Для вказання мапи необхідно створити відповідний об’єкт плагіну. Було використано “osm” плагін, що постачається з Qt5 та дозволяє використати OpenStreetMap.

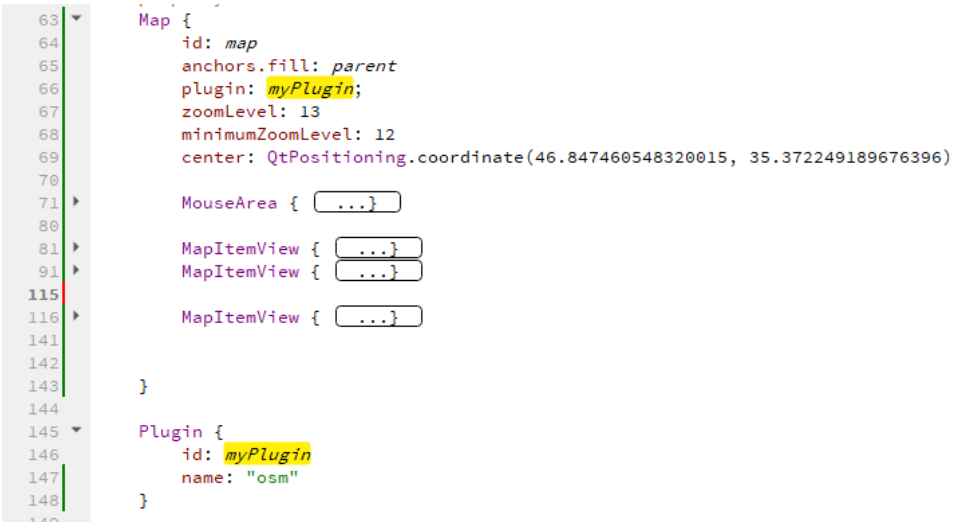


Рисунок 3.6.5 - Компонент карти

Географічний регіон, що відображається в елементі Map, позначається як його область перегляду, і це визначається властивостями center та zoomLevel. Властивість center містить координату, яка вказує центр області перегляду, тоді як zoomLevel контролює масштаб карти [16].

Для відображення певних елементів на карті було використано компронент MapItemView (рис.3.6.6).

Як джерело інформації у відображення передається модель. Було використані моделі реалізовані у попередньому підрозділі. Далі необхідно визначити делегат для візуалізації кожного елемента. Для позначення пристроїв було використано Image. Розміри такого делегата не залежать від масштабу карти. Для відображення зони покриття базової станції було використано MapCircle. Цей елемент масштабується разом з картою, що дозволяє коректно відображати радіус покриття.

У делегата є доступ до полей об’єкта за іменами ролей, що були визначені у моделі. Таким чином у делегат передається необхідні географічні координати з моделі.

```
81          MapItemView {
82              model: lorawanStationModel
83              delegate: MapCircle {
84                  center: location
85                  radius: 3000.0
86                  color: 'green'
87                  opacity: 0.05
88                  border.width: 1
89              }
90          }
91          MapItemView {
92              model: lorawanStationModel
93              delegate: MapQuickItem {
94                  coordinate: location
95
96                  anchorPoint.x: baseStationImage.width * 0.5
97                  anchorPoint.y: baseStationImage.height * 0.5
98                  sourceItem: Image {
99                      id: baseStationImage
100                     width: 30
101                     fillMode: Image.PreserveAspectFit
102                     source: "/icons/base_station.png"
103                     MouseArea {
104                         anchors.fill: parent
105                         hoverEnabled: false
106                         acceptedButtons: Qt.LeftButton
107                         onClicked: {
108                         activeMouse.processStationClick(lorawanStationModel, index)
109                         lorawanStationModel.updateModel()
110                         }
111                     }
112                 }
113             }
114         }
```

Рисунок 3.6.6 - Використання відображень та делегатів

Щоб елементи карти були інтерактивними необхідно обробляти події миші. MouseArea дозволяє визначити обробку різних подій. Необхідна поведінка може відрізнятись в залежності від стану додатку. Через це визначати поведінку на місці не дуже зручно. Замість цього поведінку винесено у окремі класи, що мають визначеними методи з однаковими іменами. Завдяки динамічній типізації JavaScript у обробці події відбувається виклик методу з заздалегідь відомим ім’ям.

Результуючий компонент зображено на Рис. 3.6.7

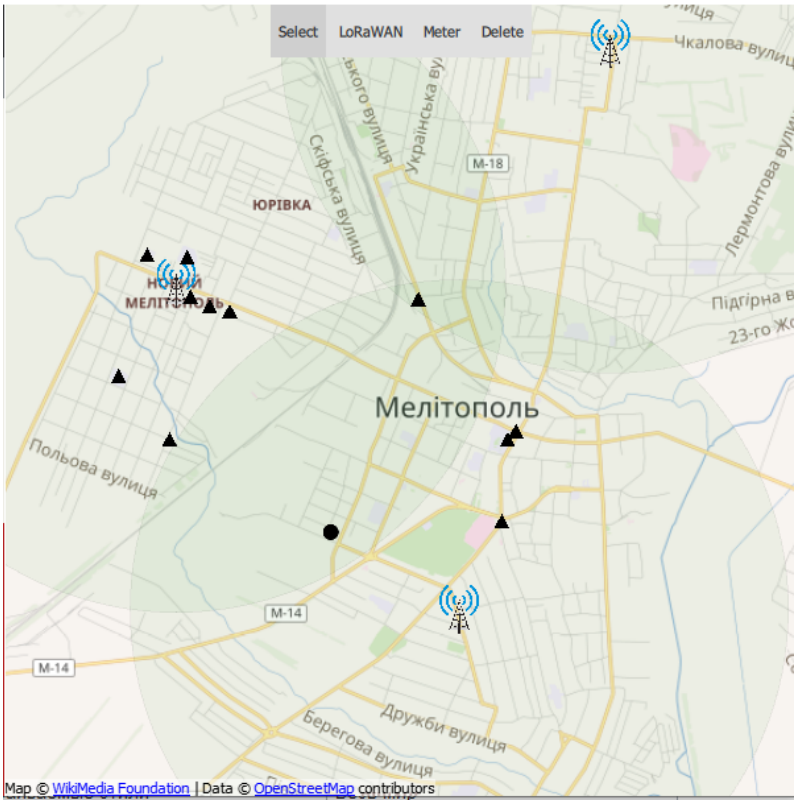


Рисунок 3.6.7 - Результуючий компонент мапи

Панель у верхній частині дозволяє вибирати інструмент, що змінює поведінку при обробці натиску кнопок миші. Були створені інструменти для:

- вибору елемента на карті;
- додавання нової базової станції;
- додавання лічильника;
- видалення елемента.

Додавання елемента на карту відбувається за допомогою подвійного кліку. При додаванні нового лічильника створюється модальний діалог, з затемненням фону (рис.3.6.8). У з’явленому діалогу необхідно вказати тип зв’язку, клієнт та ID лічильника.

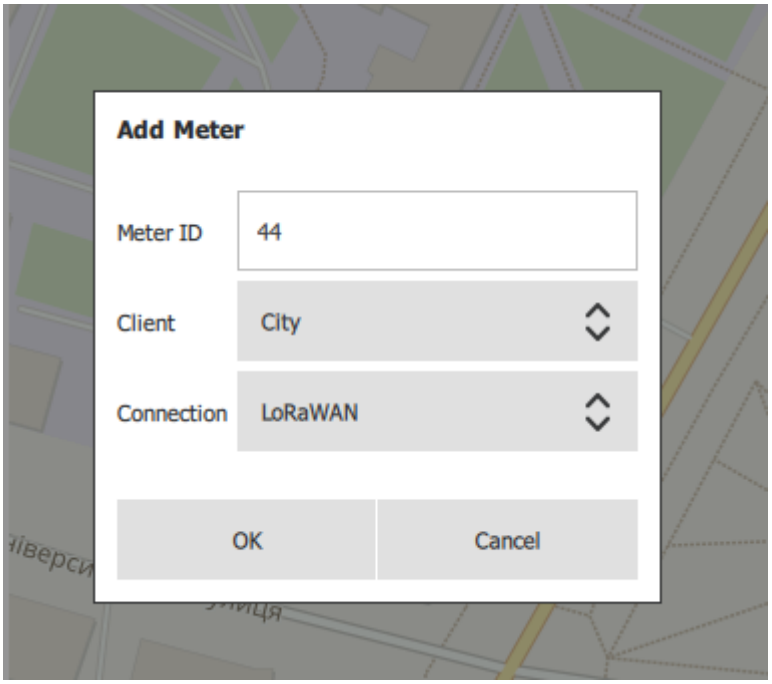


Рисунок 3.6.8 - Додавання лічильника

Для кожного лічильника збираються покази. Для візуалізації цих показів використано ChartView (рис.3.6.9).

```
ChartView
{
    id: energy_chart
    legend.visible: false
    VXYModelMapper
    {
        model: meterEnergyReadingModel
        series: energy_series
        xColumn: 1
        yColumn: 2
    }

    LineSeries
    {
        id: energy_series
        axisX: DateTimeAxis {
            min: min_time
            max: max_time
            titleText: "Time"
            format: "HH:mm:ss"
        }
        axisY: ValueAxis {
            titleText: "Energy (Wh)"
            min: 0
        }
    }
    Layout.fillHeight: true
    Layout.fillWidth: true
}
```

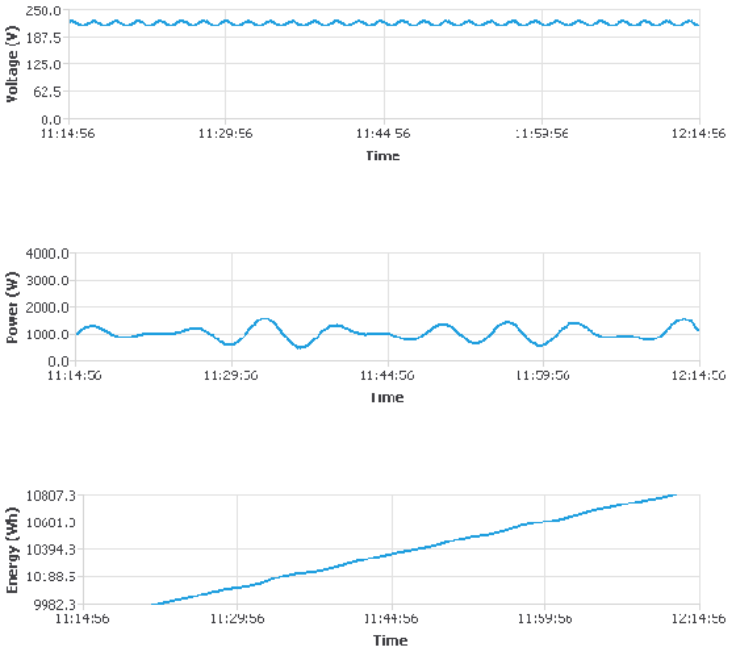


Рисунок 3.6.9 Створення графіку

Цьому компоненту необхідно отримати опис вісей та дані для візуалізації. Дані мають надаватись через інтерфейс QAbstractSeries. Для перетворення показів з моделі у серію було застосовано VXYModelMapper з

вказанням які ролі моделі використовувати для яких компонент на XY площині. Було створено графіки для кожного з зберігаємих показів.

3.7 Висновки до розділу

У даному розділі розгорнуто базу даних PostgreSQL. Використовуючи мову Python 3 та бібліотеку OpenLEADR реалізовано клієнтську та серверну частину для роботи з OpenADR, проемульовано генерацію значень з лічильників. Використавши Qt5, C++ та QML з JS було створено додаток з графічним інтерфейсом.

ВИСНОВКИ

Було проведено аналіз системи розумної електромережі, описана користь вдосконалення системи вимірювання. Засобами Oracle Data Modeler розроблено логічні та фізичні моделі бази даних для області застосування системи розумної електромережі. Розроблені моделі були використані при розробці додатку. СУБД PostgreSQL розгорнута у контейнері за допомогою Docker. Було спроектовано систему, визначено окремі її компоненти.

Проведено аналіз протоколів, що використовуються для моніторингу електромережі. Для подальшої роботи обрано OpenADR. Використавши OpenLEADR та мову Python 3, земульовано генерацію та передачу показів з лічильників на сервер та їх збереження у БД.

Для створення додатку з графічним інтерфейсом використано фреймворк Qt5. Засобами цього фреймворку та мови C++17 створено моделі даних для роботи з таблицями PostgreSQL. Створено Графічний інтерфейс створено за допомогою технології QtQuick. Структура та частина зв’язків визначена декларативною мовою QML, інтерактивна поведінка реалізована вставками JavaScript коду.

Отримана система складається з окремих компонентів. Таке відокремлення полегшує додавання нових змін: редагування одного з компонентів не впливає на інші.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smart grids and meters [Електроннний ресурс] / Режим доступу: https://ec.europa.eu/energy/topics/markets-and-consumers/smart-grids-and-meters/overview_en (дата звертання 14.12.2020)
2. Smart Metering Enabling The Smart Grid [Електроннний ресурс] / Режим доступу: https://www.eprmagazine.com/interview/smart-metering-enabling-the-smart-grid/ (дата звертання 30.05.2021)
3. ПОЛИТАРИФ-А. [Електроннний ресурс] / Режим доступу: https://spbzip.ru/tekhnologii/politarif-a (дата звертання 30.05.2021)
4. WebNMS - Enterprise Internet of Things (IoT) Solution. [Електроннний ресурс] / Режим доступу: https://www.webnms.com (дата звертання 30.05.2021)
5. WinPM.Netpower monitoring. [Електроннний ресурс] / Режим доступу: https://assets.new.siemens.com/siemens/assets/api/uuid:902ab559-bfb5-46d9-95bd-2b63ead96bb6/ca-si-lv-en-winpm-net-app-guide-si-ep-1722.pdf (дата звертання 30.05.2021)
6. LoRaWAN and NB-IoT : competitors or complementary. [Електроннний ресурс] / Режим доступу: https://lora-alliance.org/wp-content/uploads/2020/11/cr-lora-102_lorawanr_and_nb-iot.pdf (дата звертання 30.05.2021)
7. LoRaWAN vs NB-IoT: A Comparison Between IoT Trend-Setters. [Електроннний ресурс] / Режим доступу: https://ubidots.com/blog/lorawan-vs-nb-iot/ (дата звертання 30.05.2021)
8. Understanding OpenADR. [Електроннний ресурс] / Режим доступу: https://www.automatedbuildings.com/news/jul12/articles/openadr/120626102101openadr.html (дата звертання 30.05.2021)
9. Welcome to OpenLEADR. [Електроннний ресурс] / Режим доступу: https://openleadr.org/docs/ (дата звертання 30.05.2021)

10. LF Energy launches openLEADR to streamline integration of green energy for demand side management. [Електроннний ресурс] / Режим доступу: https://energynorthern.com/2020/10/28/lf-energy-launches-openleadr-to-streamline-integration-of-green-energy-for-demand-side-management/ (дата звертання 30.05.2021)
11. PLLUG C++/Qt Roadmap Book [Електроннний ресурс] / Режим доступу: https://pllug-community.gitbook.io/pllug-c-qt-roadmap-book/ (дата звертання 14.12.2020)
12. How to Write a Minimal Qt QML application with CMake. [Електроннний ресурс] / Режим доступу: https://www.apriorit.com/dev-blog/475-qt-qml-with-cmake (дата звертання 14.12.2020)
13. Столяров А.И. Построение концептуальной схемы баз данных // Портал научно-практических публикаций [Електроннний ресурс] / Режим доступу: http://portalnp.ru/2014/06/2064 (дата звертання 13.12.2020)
14. What is Docker? [Електроннний ресурс] / Режим доступу: https://www.ibm.com/cloud/learn/docker (дата звертання 14.12.2020)
15. Models and Views in Qt Quick. [Електроннний ресурс] / Режим доступу: https://doc.qt.io/qt-5/qtquick-modelviewsdata-modelview.html (дата звертання 30.05.2021)
16. Maps and Navigation (QML). [Електроннний ресурс] / Режим доступу: https://doc.qt.io/qt-5/location-maps-qml.html (дата звертання 30.05.2021)
17. IEA (2011) Technology Road Map on Smart Grids. International Energy Agency.
18. Книга: Організація баз даних та знань [Електроннний ресурс] / Режим доступу: https://cinref.ru/razdel/02200informatika/19/377919.htm (дата звертання 13.12.2020)

19. Smart Grid: A Beginner's Guide [Електроннний ресурс] / Режим доступу: https://www.nist.gov/el/smart-grid/about-smart-grid/smart-grid-beginners-guide (дата звертання 14.12.2020)

20. Grid Modernization and the Smart Grid [Електроннний ресурс] / Режим доступу: https://www.energy.gov/oe/activities/technology-development/grid-modernization-and-smart-grid (дата звертання 14.12.2020)

21. Data Modeling with Oracle SQL Developer [Електроннний ресурс] / Режим доступу: https://www.oracle.com/database/technologies/appdev/datamodeler.html (дата звертання 14.12.2020)

22. Overview of Docker Compose. [Електроннний ресурс] / Режим доступу: https://docs.docker.com/compose/ (дата звертання 14.12.2020)

23. An introduction to WSL 2. [Електроннний ресурс] / Режим доступу: https://www.infoworld.com/article/3412063/an-introduction-to-wsl-2.html (дата звертання 14.12.2020)

24. Using CMake with Qt 5 [Електроннний ресурс] / Режим доступу: https://www.kdab.com/using-cmake-with-qt-5 (дата звертання 10.05.2021)

25. Introduction to Graph Theory [Електроннний ресурс] / Режим доступу: https://www.maths.ed.ac.uk/~v1ranick/papers/wilsongraph.pdf (дата звертання 14.12.2020)

26. Мейерс С. Эффективный и современный C++. / С. Мейерс. – М.:Вильямс, 2016.

27. Smart Grid Market Trends, Companies, Driver, Segmentation, Forecast to 2024 [Електроннний ресурс] / Режим доступу: https://www.marketwatch.com/press-release/smart-grid-market-trends-companies-driver-segmentation-forecast-to-2024-2020-12-09?tesla=y (дата звертання 14.12.2020)

28. The four stages to a truly smart grid [Електроннний ресурс] / Режим доступу: https://www.energynetworks.com.au/news/energy-insider/2020-energy-insider/the-four-stages-to-a-truly-smart-grid/ (дата звертання 14.12.2020)

29. Smart Grid, Smart City: the future of Australia's electricity networks [Електроннний ресурс] / Режим доступу: https://utilitymagazine.com.au/smart-grid-smart-city-the-future-of-australias-electricity-networks/ (дата звертання 30.05.2021)

30. Networking architectures and protocols for smart city systems [Електроннний ресурс] / Режим доступу: https://jisajournal.springeropen.com/articles/10.1186/s13174-018-0097-0 (дата звертання 30.05.2021)

31. Advantages of Implementing an Advanced Metering Infrastructure [Електроннний ресурс] / Режим доступу: https://electricenergyonline.com/energy/magazine/256/article/Advantages-of-Implementing-an-Advanced-Metering-Infrastructure.htm

32. Australian Standards for Smart Grids – Standards Roadmap [Електроннний ресурс] / Режим доступу: https://www.standards.org.au/StandardAU/Media/SA-Archive/Documents/120904-Smart-Grids-Standards-Road-Map-Report.pdf (дата звертання 30.05.2021)

33. Smart Grid Communication Protocols [Електроннний ресурс] / Режим доступу: https://www.ijtsrd.com/papers/ijtsrd21344.pdf (дата звертання 30.05.2021)

34. Створення інформаційної системи моніторингу забруднення атмосферного повітря міста на основі технології «Інтернет речей» [Текст] / В. Б. Мокін, Б. Ю. Собко, Є. М. Крижановський [та ін.] // Вісник Вінницького політехнічного інституту. — 2017. — № 3. — С. 49-58.

35. Smart grid monitoring: new opportunities and challenges with digitalization and 5G integration. [Електроннний ресурс] / Режим доступу: https://blog.sintef.com/sintefenergy/energy-systems/smart-grid-monitoring-ict-5g/ (дата звертання 30.05.2021)

36. Build with CMake. Build with Confidence. [Електроннний ресурс] / Режим доступу: https://cmake.org/ (дата звертання 30.05.2021)

37. Grid and Form Layouts in Qt5 Python. [Електроннний ресурс] / Режим доступу: https://www.tutorialkart.com/blog/grid-and-form-layouts-in-qt5-python/ (дата звертання 30.05.2021)

38. Design Patterns. [Електроннний ресурс] / Режим доступу: https://sourcemaking.com/design_patterns (дата звертання 30.05.2021)

39. What is PostgreSQL?. [Електроннний ресурс] / Режим доступу: https://www.ibm.com/cloud/learn/postgresql (дата звертання 30.05.2021)

40. PostGIS — Spatial and Geographic Objects for PostgreSQL. [Електроннний ресурс] / Режим доступу: https://postgis.net (дата звертання 30.05.2021)

41. Modeling Polymorphic Associations in a Relational Database. [Електроннний ресурс] / Режим доступу: https://hashrocket.com/blog/posts/modeling-polymorphic-associations-in-a-relational-database (дата звертання 30.05.2021)

42. Efficient Evenly Distributed Sampling of Time Series Records in PostgreSQL. [Електроннний ресурс] / Режим доступу: https://blog.joshsoftware.com/2020/10/14/efficient-evenly-distributed-sampling-of-time-series-records-in-postgresql/ (дата звертання 30.05.2021)

43. Connecting to Databases. [Електроннний ресурс] / Режим доступу: https://doc.qt.io/qt-5/sql-connecting.html (дата звертання 30.05.2021)

44. Python multiprocessing - process-based parallelism in Python. [Електроннний ресурс] / Режим доступу: https://zetcode.com/python/multiprocessing/ (дата звертання 30.05.2021)

45. An Introduction to Asynchronous Programming in Python. [Електроннний ресурс] / Режим доступу: https://medium.com/velotio-perspectives/an-introduction-to-asynchronous-programming-in-python-af0189a88bbb (дата звертання 30.05.2021)

46. Flexibility Services Based on OpenADR Protocol for DSO Level. [Електроннний ресурс] / Режим доступу: https://www.mdpi.com/1424-8220/20/21/6266/htm (дата звертання 30.05.2021)

47. Smart Grid Communication Protocols –Impact on Smart Grid Applications [Електроннний ресурс] / Режим доступу: https://emmos.org/prevconf/2014/Smart%20Grid%20Communication%20Protocols-Application-Impact-Ralph%20Mackiewicz.pdf (дата звертання 30.05.2021)

48. Time Zones with Qt [Електроннний ресурс] / Режим доступу: http://silmor.de/qtstuff.tzone.php (дата звертання 30.05.2021)

Додаток А. Презентація захисту роботи

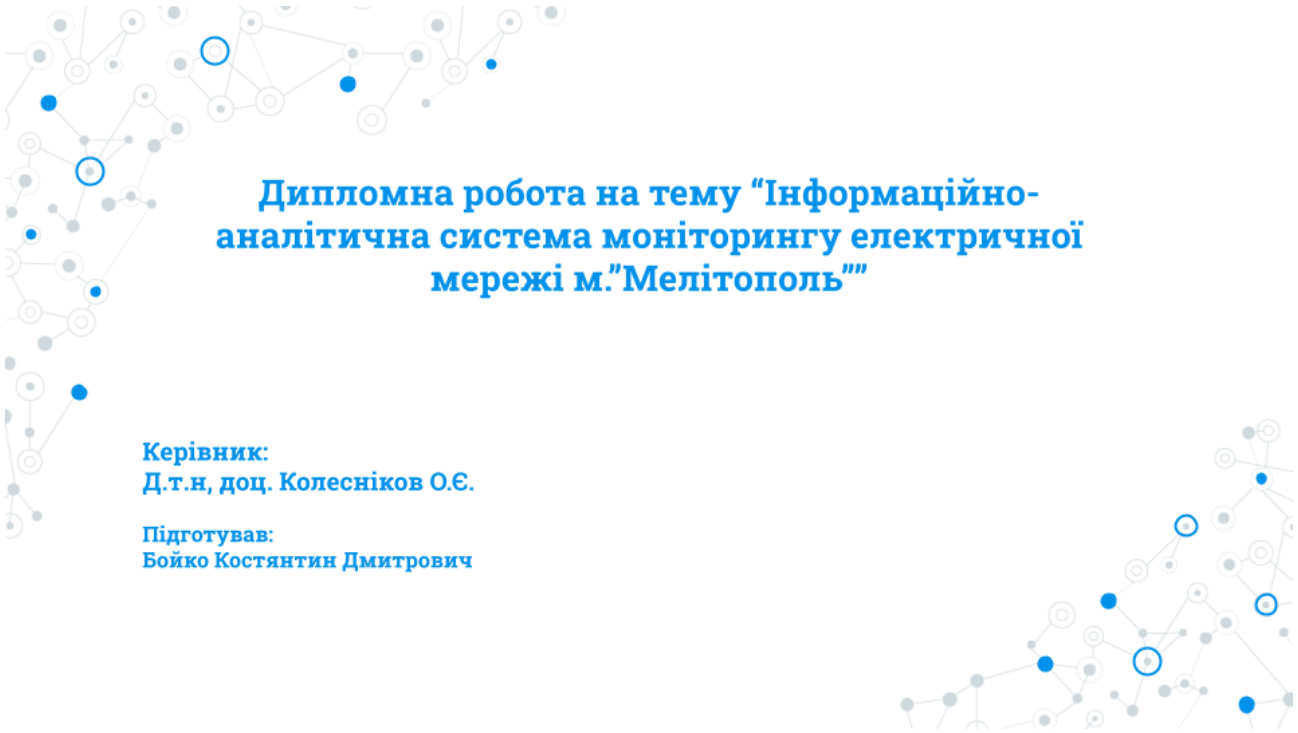


Рисунок А 1 – Слайд 1 Тема доповіді

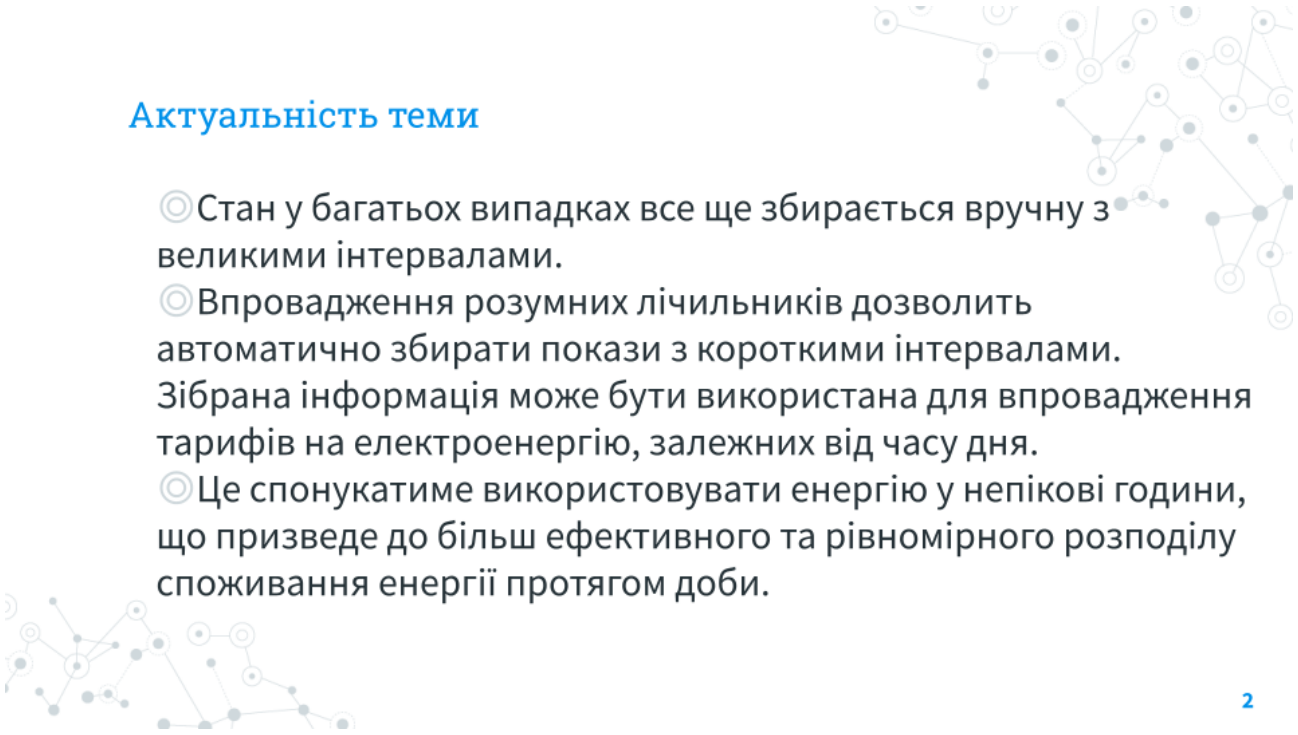


Рисунок А 2 – Слайд 2 Актуальність теми

Рисунок А 3 – Слайд 3 Аналіз технологій, що використовуються

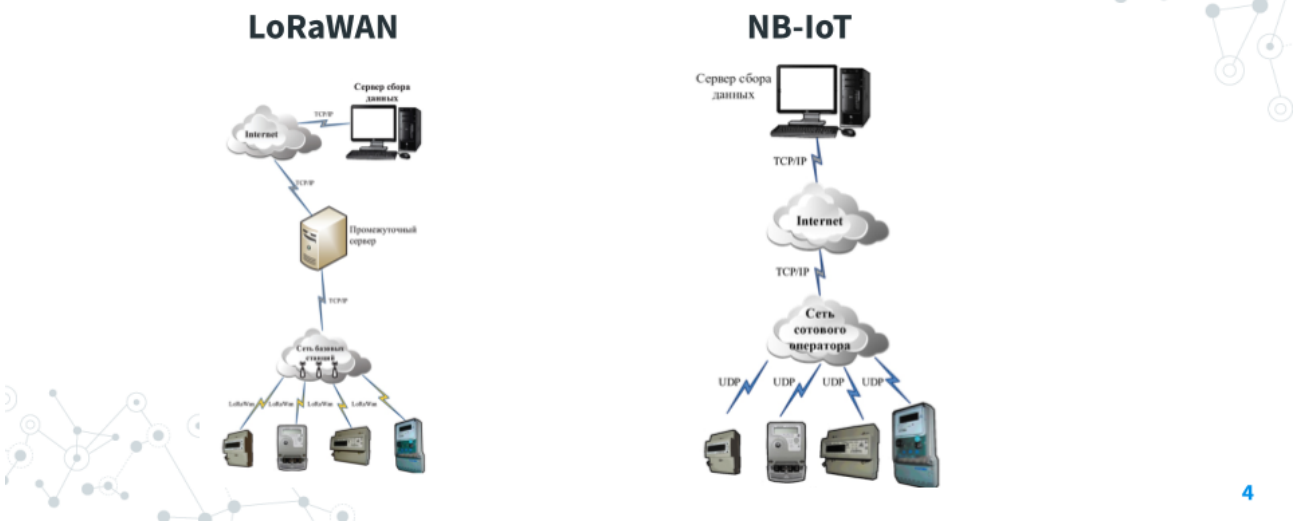


Рисунок А 4 – Слайд 4 Основні технології для широкого покриття

Рисунок А 5 – Слайд 5 Розроблена архітектура системи моніторингу

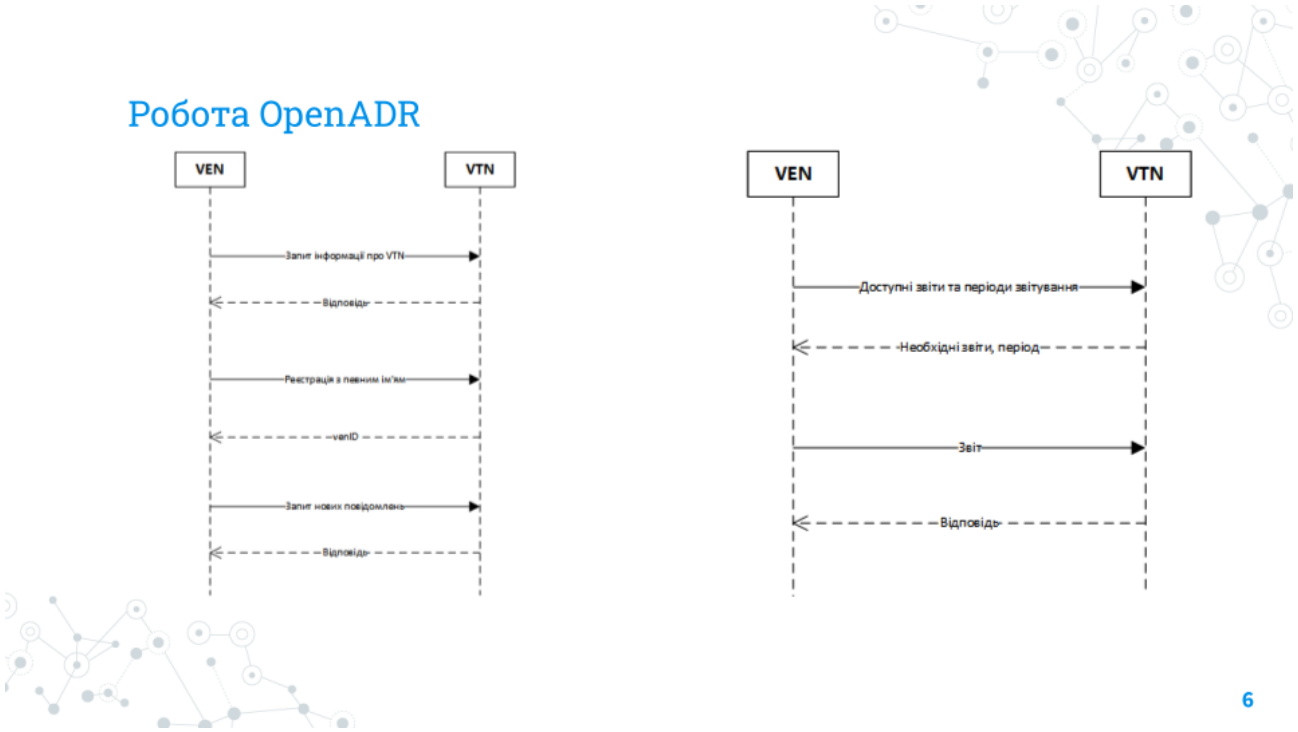


Рисунок А 6 – Слайд 6 Робота OpenADR

Генерація значень на VEN

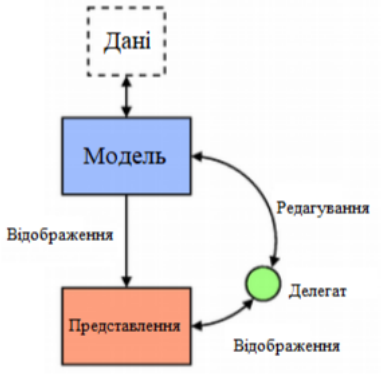
```python
async def read_voltage():
    value = 220 + math.sin(time.time()/20) * 5
    return value

async def read_power(namespace, period, phase):
    value = 1000 + math.cos(time.time()/period + phase) * 1000
    if not namespace.enabled:
        value = 0
    namespace.total_energy += convert_to_hours(sampling_rate) * value
    return value

async def read_energy(namespace):
    return namespace.total_energy
```

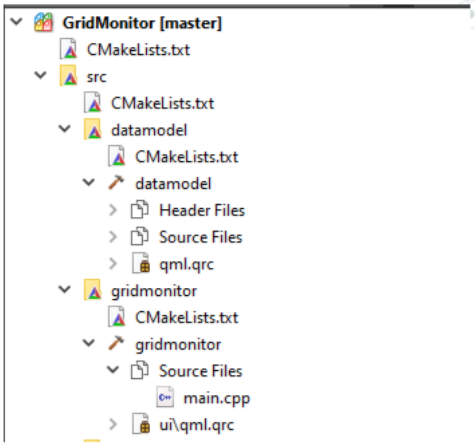
```python
client = OpenADRClient(ven_name=id,
                       vtn_url='http://127.0.0.1:8080/OpenADR2/Simple/2.0b')
client.add_report(callback=read_voltage,
                  report_specifier_id='VoltageReport',
                  resource_id='smart_meter1',
                  measurement='Voltage',
                  sampling_rate=sampling_rate,
                  unit='V')
client.add_report(callback=partial(read_power, Global, randint(20, 100), randint(0, 1000000)),
                  report_specifier_id='PowerReport',
                  resource_id='smart_meter1',
                  measurement='Power',
                  sampling_rate=sampling_rate,
                  unit='W')
client.add_report(callback=partial(read_energy, Global),
                  report_specifier_id='EnergyReport',
                  resource_id='smart_meter1',
                  measurement='Total Energy',
                  sampling_rate=sampling_rate,
                  unit='Wh')
```

7

Рисунок А 7 – Слайд 7 Генерація значень на VEN

Розділення даних та логіки



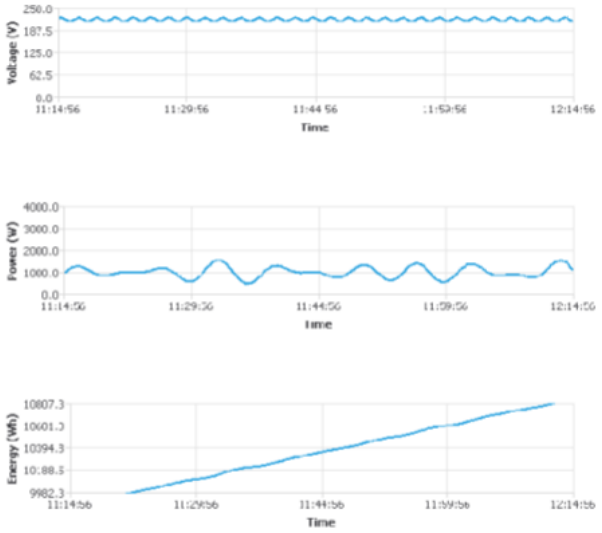


8

Рисунок А 8 – Слайд 8 Розділення даних та логіки

Створення візуалізації показів

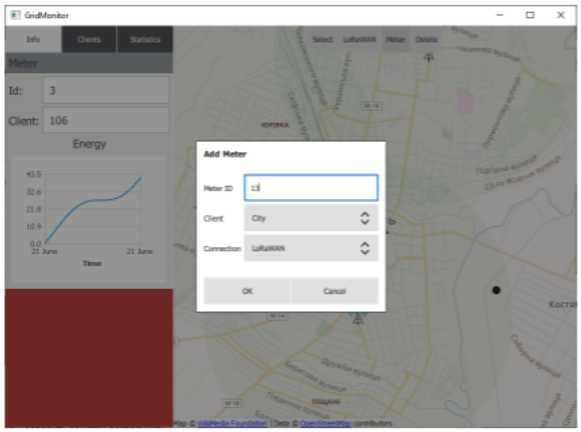
```
ChartView
{
    id: energy_chart
    legend.visible: false
    VXYModelMapper
    {
        model: meterEnergyReadingModel
        series: energy_series
        xColumn: 1
        yColumn: 2
    }

    LineSeries
    {
        id: energy_series
        axisX: DateTimeAxis {
            min: min_time
            max: max_time
            titleText: "Time"
            format: "HH:mm:ss"
        }
        axisY: ValueAxis {
            titleText: "Energy (Wh)"
            min: 0
        }
    }
    Layout.fillHeight: true
    Layout.fillWidth: true
}
```

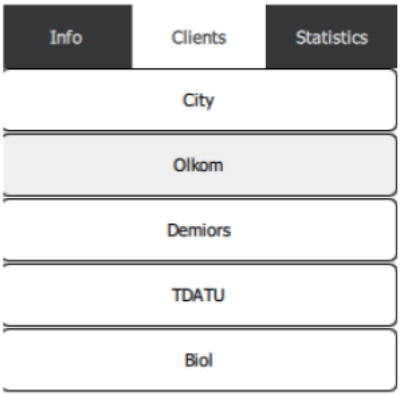


9

Рисунок А 9 – Слайд 9 Створення візуалізації показів

Частини графічного інтерфейсу





10

Рисунок А 10 – Слайд 10 Частини графічного інтерфейсу

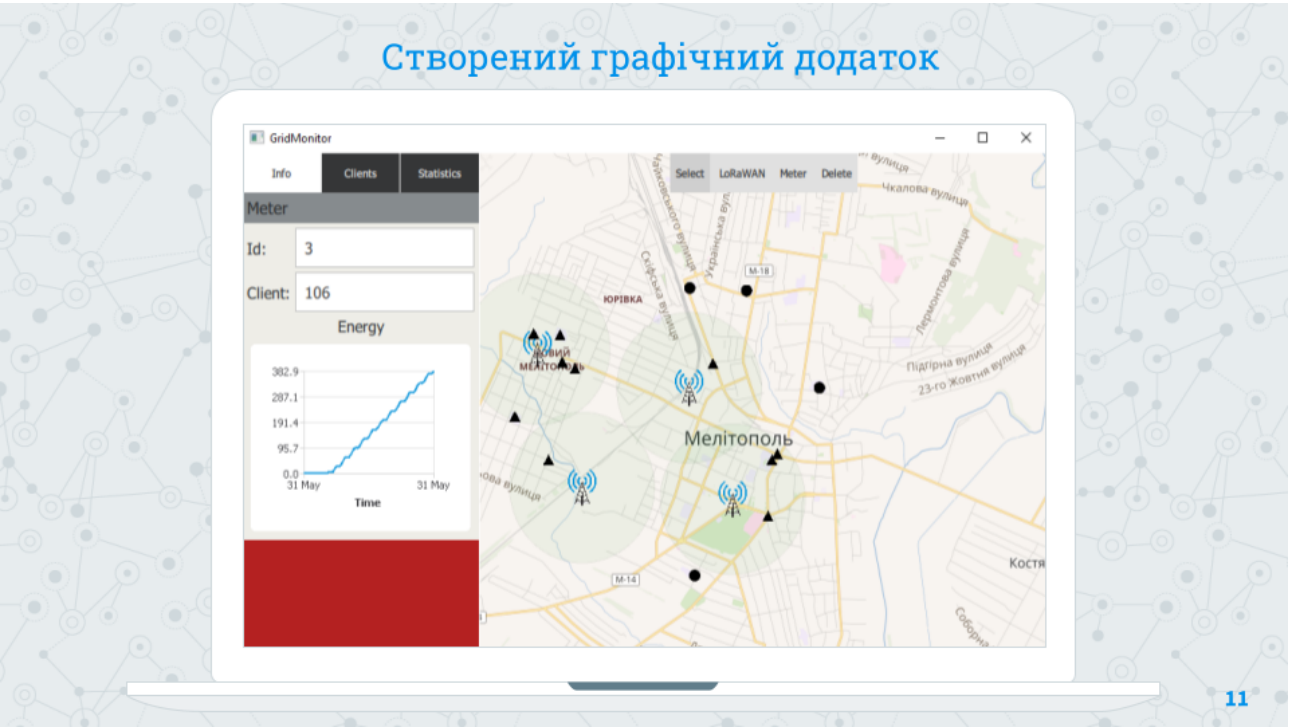


Рисунок А 11 – Слайд 11 Створений графічний додаток

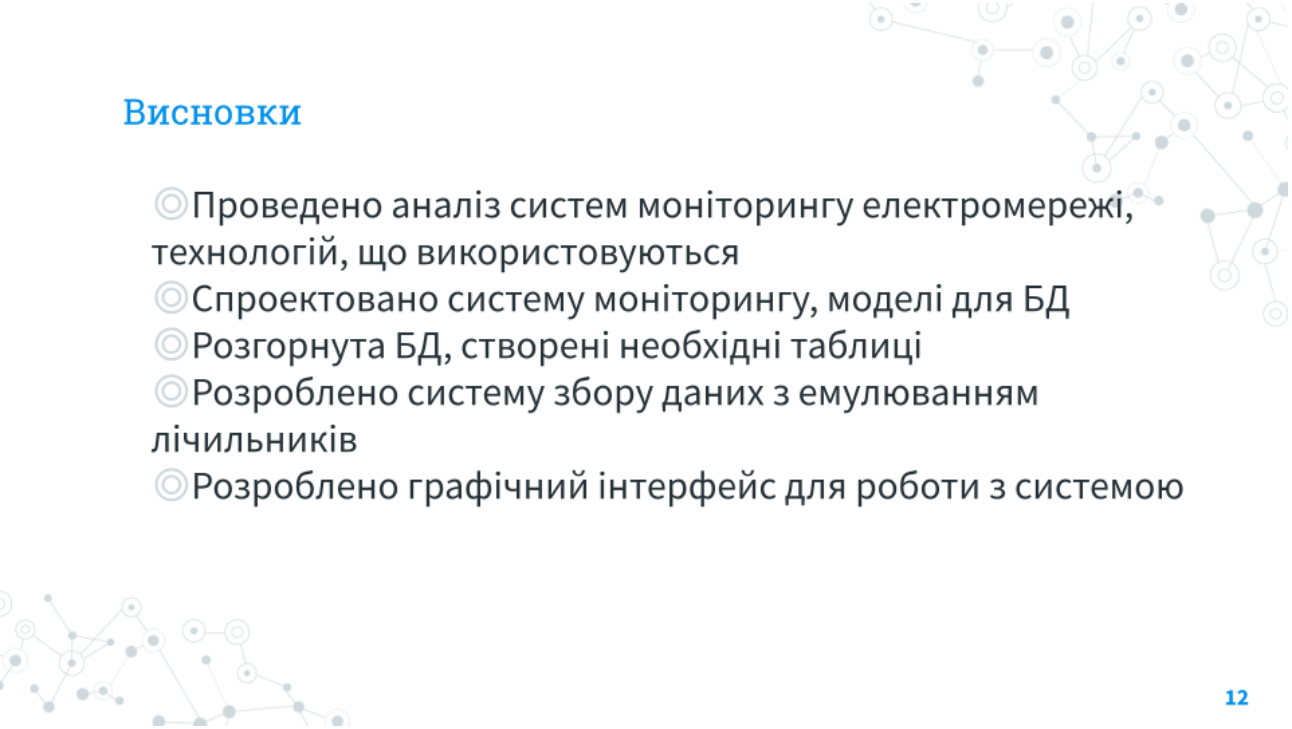


Рисунок А 12 – Слайд 12 Висновки