МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Мікросервіс підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі з відновлюваними джерелами енергії»

Здобувача групи ІТ.м-23 Рикуна Владислава Андрійовича
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ Владислав РИКУН
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник ст. викладач кафедри ІТ, к.т.н., Ольга БОЙКО _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Рикун Владислав Андрійович
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи «Мікросервіс підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі з відновлюваними джерелами енергії»

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «11» грудня 2023 р.
3 Вхідні дані до кваліфікаційної роботи критерії для розробки мікросервісу

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, проектування мікросервісу підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі з відновлюваними джерелами енергії, програмна реалізація мікросервіса

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) контекстна діаграма IDEF0, діаграма декомпозиції першого рівня, діаграма варіантів використання Use Case, архітектура web-додатку

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	07.11 – 07.11	
2	Визначення вимог	07.11 – 08.11	
3	Визначити засоби реалізації	10.11 – 13.11	
4	Планування WBS	14.11 – 14.11	
5	Планування ODS	15.11 – 15.11	
6	Складання календарного плану	16.11 – 16.11	
7	Моделювання роботи мікросервісу	17.11 – 20.11	
8	Розробка функціоналу мікросервісу	21.11 – 27.11	
9	Тестування	28.11 – 30.11	
10	Написання автотестів	01.12 – 01.12	
11	Тестування розробником	04.12 – 04.12	

Магістрант _____ Владислав РИКУН

Керівник роботи _____ к.т.н., Ольга БОЙКО

АННОТАЦІЯ

Тема кваліфікаційної роботи магістра «Мікросервіс підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі з відновлюваними джерелами енергії».

Пояснювальна записка складається з вступу, 4 розділів, висновку, списку використаних джерел із 42 найменувань, додатків А-Б. Загальний обсяг роботи − 75 сторінок, у тому числі 44 сторінок основного тексту, 4 сторінки списку використаних джерел, 25 сторінок додатків.

Актуальність роботи: Збільшення використання відновлюваних джерел енергії призводить до складніших інфраструктурних викликів для енергетичних систем. Інтеграція відновлюваних джерел енергії вимагає ефективної системи прийняття рішень. Мікросервісна архітектура може забезпечити ефективний спосіб реалізації систем підтримки прийняття рішень.

Мета: створення мікросервісу підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі з використанням відновлювальних джерел енергії.

Практичне значення: Створення мікросервісу для управління режимами роботи енергетичної мікромережі покращує гнучкість, легкість інтеграції та розширення системи. Крім того, воно відкриває перспективи для інших аспектів, таких як можливість інтеграції мікросервісу з іншими системами, надання програмного інтерфейсу (API) для створення власних додатків чи інтеграції з іншими платформами. Це робить систему більш адаптивною до змін і розширює її функціональність для впровадження різноманітних інновацій.

Ключові слова: мікросервіс, мікрогрід, FastAPI, автотести, Docker.

ЗМІСТ

ВСТУП

Сучасний світ стоїть перед екологічною кризою, яка становить загрозу для всього людства. Збільшення витрат енергії, зростання цін на неї та необхідність обмеження використання викопних видів палива вимагають переходу до використання відновлюваних джерел енергії (ВДЕ). У сфері електроенергетики спостерігається прискорений розвиток технічних, економічних та ринкових трансформацій. Розвинуті країни впроваджують нові підходи до виробництва та розподілу електроенергії, спрямовані на задоволення попиту та зменшення використання вуглеводневих пальників, що призводять до викидів парникових газів.

Цифрова трансформація має суттєвий вплив на сферу електроенергетики. Навіть у консервативному секторі електроенергетики спостерігається активне впровадження передових технологій. Це не дивно, оскільки в умовах зростання автоматизації у всіх сферах життя традиційне "ручне управління" стає непотрібним. Також змінюється характер генерації та споживання електроенергії, переходячи від централізованого до розподіленого, та впровадженню мікромереж.

Заходи до цифрової трансформації в енергетичному секторі передбачають впровадження сучасних технологій для підвищення ефективності та автоматизації процесів управління енергією. В контексті зазначених тенденцій розробка інформаційної системи, яка інтегрує дані з усіх джерел і забезпечує ухвалення управлінських рішень для оптимізації використання енергії, стає необхідною складовою сучасного енергетичного середовища.

Отже, використання систем підтримки прийняття рішень щодо управління режимами роботи мікромережі є досить актуальним. Однак, вибір правильної архітектури для таких систем є ключовим етапом у їхньому успішному впровадженні.

Об'єкт дослідження – підтримка прийняття рішень щодо управління режимами роботи енергетичної мікромережі.

Предмет дослідження – процес реалізації мікросервісу для підтримки прийняття рішень щодо управління режимами роботи енергетичної мікромережі.

Мета роботи полягає у розробці та імплементації мікросервісу, інтегрованого в систему підтримки прийняття рішень щодо режимів функціонування енергетичної мікромережі.

Практична цінність полягає у створенні мікросервісу, інтегрованого в систему підтримки прийняття рішень щодо режимів функціонування енергетичної мікромережі, що покращує гнучкість, легкість інтеграції та розширення системи. Крім того, воно відкриває перспективи для інших аспектів, таких як можливість інтеграції мікросервісу з іншими системами, надання програмного інтерфейсу (API) для створення власних додатків чи інтеграції з іншими платформами. Це робить систему більш адаптивною до змін і розширює її функціональність для впровадження різноманітних інновацій.

Для досягнення цієї мети необхідно вирішити наступні задачі:

1. Дослідження стану проблеми та перспектив розвитку енергетичних мікромереж, ознайомлення з вимогами та підходами до проектування мікросервісів, формування функціональних вимог.
2. Структурно-функціональне моделювання процесу створення та використання мікросервісу.
3. Програмна реалізація мікросервісу, відповідно до розробленої архітектури та функціональних вимог, його інтеграція у інформаційну систему.
4. Тестування та перевірка функціональності, продуктивності та стійкості мікросервісу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1Сучасний стан та перспективи розвитку енергетичних мікромереж

У звітах агентства з відновлювальною енергетики України зазначено, енергетична безпека є одним із найважливіших елементів національної безпеки України. Це забезпечує надійне та доступне енергопостачання населення, підприємств та інфраструктури країни. При цьому енергетична безпека не обмежується енергопостачанням, а включає економічну, екологічну та соціальну складові [1].

Стале енергетичне майбутнє України передбачає розвиток відновлюваних джерел енергії, підвищення енергоефективності та забезпечення енергетичної безпеки на місцевому рівні.

Актуальність наукової та практично-прикладної теми, активність досліджень щодо підвищення енергоефективності дуже висока. Одним із засобів досягнення розглядають впровадження систем мікрогрід та використання систем прийняття рішень щодо різних аспектів їх роботи. Це пов’язано зі збільшенням інтересу до мікромереж як до способу підвищення надійності та стійкості електричної мережі, а також до зменшення викидів парникових газів. Мікрогрід— це невеликі локалізовані електромережі, які можуть працювати незалежно від основної мереж [2]. Це робить їх більш стійкими до відключень та інших збоїв. Мікромережі також можна використовувати для виробництва відновлюваної енергії, такої як сонячна та вітрова, що може допомогти зменшити викиди парникових газів.

Згідно бази даних Scopus з 2019 по 2023 рр. Цій темі присвячено більше ніж 600 досліджень (таблиця 1.1).

Таблиця 1.1 – Дослідження актуальності в науковій літературі

Джерело: побудовано автором

База даних	Пошуковий рядок	Кількість джерел	Роки
Scopus	"microgrid" AND "decision-making"	638	2019-2023
	"microgrid" AND "control" AND "expert"	46	
	"microgrid" OR "smart grid" AND "software architecture"	36	

На рисунку 1.1. представлено карту найбільш актуальних тем, що вивчається.

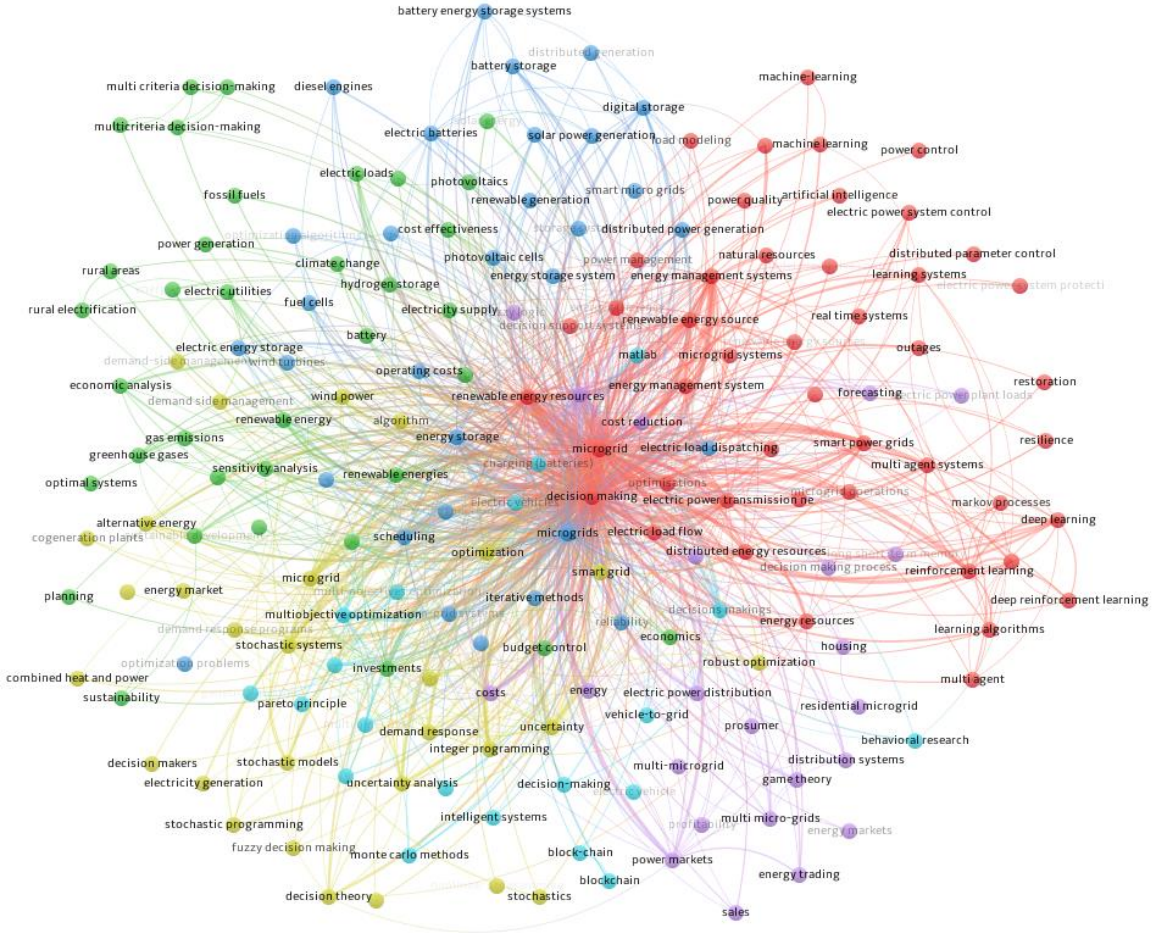


Рисунок 1.1 – Карта досліджень щодо «microgrid» AND «decision-making»

Джерело: побудовано автором

Прийняття рішень є важливою частиною планування та експлуатації мікромережі. Особи, які приймають рішення, повинні враховувати низку факторів, таких як тип мікромережі, розмір і розташування мікромережі, джерела енергії, які будуть використовуватися, і спосіб управління мікромережею. Існує низка різних інструментів прийняття рішень і структур, які можна використовувати, щоб допомогти планувальникам і операторам мікромереж приймати обґрунтовані рішення.

Експертні знання часто використовуються для прийняття рішень щодо мікромереж, оскільки вони можуть допомогти забезпечити прийняття найкращих можливих рішень. Експерти мають глибоке розуміння технічних та економічних питань, пов’язаних з плануванням та експлуатацією мікромережі. Вони також можуть надати цінну інформацію про конкретні потреби громади чи організації, яка планує розгорнути мікромережу.

Існує кілька різних способів збору експертних знань. Одним із способів є проведення інтерв’ю з експертами. Інший спосіб – проведення опитувань експертів. Також можна отримати консультації експертів під час семінарів і конференцій.

Іншим способом є використання інтерфейсів систем підтримки прийняття рішень, через які експерти надають інформацію, згідно якої приймається рішення щодо різних аспектів роботи мікромережі.

Після збору експертних знань їх можна використовувати для інформування процесу прийняття рішень. Експерти можуть надати інформацію щодо вибору технологій мікромережі, проектування мікромережі та реєимів роботи мікромережі.

Використання експертних знань може допомогти гарантувати, що мікромережі плануються, проектуються та експлуатуються безпечним, надійним та економічно ефективним способом. Це також може допомогти переконатися, що мікромережі відповідають конкретним потребам громади чи організації, яка їх розгортає.

При пошуці в базі даних Scopus за ключовими словами («microgrid» AND «control» AND «expert») в задачах контролю мікромереж можна знайти дослідження та найбільш застосовані методи вирішення проблем в них (рис. 1.2). До них можна віднести засоби штучного інтелекту та машинного навчання, нечітких множин, які застосовуються до прогнозного контролю мікромережі.

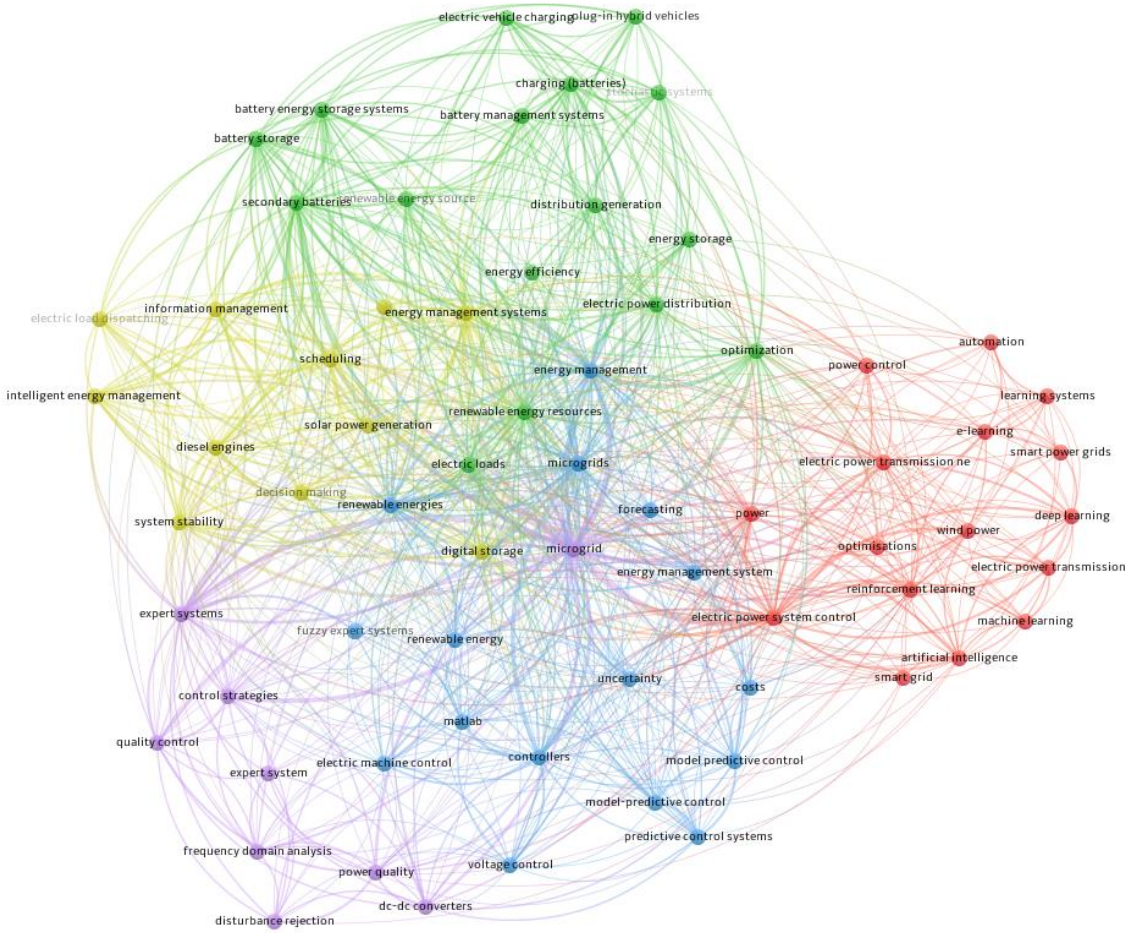


Рисунок 1.2 – Карта досліджень щодо («microgrid» AND «control» AND «expert»

Джерело: побудовано автором

1.2 Методи підтримки прийняття рішень щодо режимів роботи енергетичних мікромереж

Енергетичні мікромережі являють собою автономні системи електропостачання, що складаються з декількох джерел енергії, розподільчих мереж та споживачів [3]. У таких системах прийняття рішень щодо режимів роботи є складним завданням, оскільки необхідно враховувати безліч факторів, таких як:

- Доступність та потужність джерел енергії;
- Попит на електроенергію від споживачів;
- Рівень напруги та частоти в мережі;
- Кошти на виробництво та передачу електроенергії;
- Екологічні вимоги.

Для підтримки прийняття рішень щодо режимів роботи енергетичних мікромереж використовуються різні методи [3]. Ось деякі з основних методів:

Метод аналізу запасів. У цьому методі прийняття рішень ґрунтується на аналізі запасів енергії в мікромережі. Режим роботи мікромережі визначається таким чином, щоб забезпечити достатній запас енергії для задоволення попиту на електроенергію від споживачів.

Метод аналізу запасів працює шляхом оцінки очікуваного попиту на електроенергію протягом певного періоду часу [4], [5]. Це можна зробити за допомогою історичних даних, прогнозу погоди чи інших факторів. Коли буде відомий очікуваний попит, оператор мікромережі може розрахувати кількість запасів, необхідних для задоволення цього попиту.

Обсяг необхідних запасів залежить від ряду факторів, зокрема [6]: мінливість попиту, надійність джерел електроенергії, вартість запасів.

Метод аналізу запасів є відмінним вибором для мікромереж з стабільним попитом та надійними джерелами електроенергії. Він простий у використанні та ефективний для прийняття рішень у реальному часі, особливо в умовах обмежених ресурсів. Однак важливо враховувати, що метод не враховує

мінливість попиту і може бути неоптимальним для мікромереж із змінним попитом або ненадійними джерелами електроенергії..

Метод оптимізації. Метод оптимізації є одним із найдосконаліших методів прийняття рішень щодо режимів роботи енергетичних мікромереж. Це дозволяє врахувати всі фактори, що впливають на роботу мікромережі, і знайти оптимальний режим роботи, що відповідає поставленим цілям.

У цьому методі прийняття рішень здійснюється шляхом вирішення математичної задачі оптимізації. Задача оптимізації може бути сформульована таким чином, щоб мінімізувати витрати на виробництво та передачу електроенергії або максимізувати надійність мікромережі.

Задачу оптимізації для прийняття рішень про режими роботи енергетичних мікромереж можна сформулювати так:

$$min\ f(x)$$
$$g(x)\ \leq\ 0$$
$$h(x)\ =\ 0$$

Джерело: побудовано автором

де: f(x) - цільова функція, яка визначає мету прийняття рішення (наприклад, мінімізація витрат або максимізація надійності); x - вектор змінних, що визначають режим роботи мікромережі; g(x) - функції обмеження, що визначають допустимі значення змінних; h(x) - функції рівності, які також визначають допустимі значення змінних.

Функції обмежень g(x) і h(x) можуть визначати такі обмеження: наявність джерел енергії; попит на електроенергію з боку споживачів; рівні напруги і частоти в мережі; кошти для виробництва та передачі електроенергії; екологічні вимоги.

Задача оптимізації може бути вирішена різними методами, такими як:

- Метод дихотомії;
- Метод диференціальних рівнянь;
- Симплексний метод;

- Метод динамічного програмування;
- Метод генетичного алгоритму.

Вибір методу вирішення задачі оптимізації залежить від конкретних властивостей задачі, таких як: кількість змінних; кількість обмежень; форма цільової функції; форма функції обмежень.

Метод оптимізації дозволяє знаходити оптимальний режим мікромережі, враховуючи всі фактори, але має високі вимоги до обчислювальних ресурсів та може бути складним у формулюванні задачі, призводячи до нестабільних та неочікуваних результатів.

Метод нечіткого логічного програмування [7], [8]. У цьому методі прийняття рішень ґрунтується на нечітких міркуваннях. Режим роботи мікромережі визначається таким чином, щоб забезпечити задоволення всіх вимог до мікромережі, навіть якщо ці вимоги неможливо точно кількісно визначити.

Програмування нечіткої логіки можна використовувати для прийняття рішень щодо режимів роботи енергетичних мікромереж. Наприклад, його можна використовувати для:

- Визначити оптимальний режим роботи мікромережі за наявності невизначеної потреби в електроенергії;
- Визначити оптимальний режим роботи мікромережі за наявності невизначеної доступності відновлюваних джерел енергії;
- Визначити оптимальний режим роботи мікромережі за наявності невизначених умов зовнішнього середовища.

Визначення оптимального режиму роботи мікромережі за наявності невизначених умов середовища. У цьому випадку програмі з нечіткою логікою необхідно враховувати наступні фактори:

- Очікувані умови навколишнього середовища.
- Невизначеність очікуваних умов навколишнього середовища.
- Вплив умов зовнішнього середовища на роботу мікромережі.

Програма нечіткої логіки потім використовуватиме ці фактори для визначення оптимального режиму роботи мікромережі, наприклад кількість електроенергії, яку потрібно виробити з кожного джерела, кількість електроенергії, яку потрібно зберігати, і кількість електроенергії, яку потрібно імпортувати або експортувати з мережі. .

Цей метод забезпечує ефективні режими роботи мікромережі, навіть при невизначених вимогах, але має високі вимоги до обчислювальних ресурсів та може виявитися складним у формулюванні нечітких міркувань для оператора мікромережі.

Метод машинного навчання [9]. Машинне навчання (ML) можна використовувати для підтримки прийняття рішень щодо роботи енергетичної мікромережі. Алгоритми ML можна навчити на історичних даних, щоб навчитися прогнозувати майбутнє навантаження, виробництво та погодні умови. Потім ця інформація може бути використана для прийняття рішень про те, як експлуатувати мікромережу таким чином, щоб відповідати її цілям.

Існує ряд різних методів ML, які можна використовувати для роботи енергетичної мікромережі. Одним із поширених підходів є використання регресійних моделей для прогнозування майбутнього навантаження. Регресійні моделі навчаються на наборі історичних даних про навантаження, щоб дізнатися про зв’язок між навантаженням та іншими факторами, такими як час доби, день тижня та погодні умови.

Іншим поширеним підходом є використання моделей класифікації для прогнозування оптимальної суміші генерацій. Моделі класифікації навчаються на наборі історичних даних про виробництво, щоб дізнатися про зв’язок між виробництвом та іншими факторами, такими як навантаження, погода та вартість різних джерел виробництва.

Методи ML також можна використовувати для оптимізації роботи енергетичних мікромереж. Алгоритми оптимізації можна використовувати

для пошуку набору рішень щодо генерації та диспетчеризації навантаження, які мінімізують вартість експлуатації або максимізують надійність системи.

Методи ML є багатообіцяючим підходом до підтримки прийняття рішень щодо роботи енергетичної мікромережі. Вони пропонують потенціал для підвищення ефективності, надійності та економічної ефективності роботи мікромережі.

Цей метод може забезпечити ефективні режими роботи мікромережі при невизначених вимогах, але вимагає наявності достатньої кількості історичних даних та може бути викликаний складністю навчання машинної моделі для точних прогнозів попиту на електроенергію.

1.3 Використання інформаційних продуктів кінцевим користувачем

Інформаційні продукти, призначені для підтримки прийняття рішень щодо режимів роботи мікромережі, повинні бути простими у використанні та зрозумілими для стейкхолдерів.

Залежно від конкретних потреб системи підтримки прийняття рішень (СППР) щодо режимів роботи мікрогрід до її стейкхолдерів можуть входити:

- *Оператори мікромережі.* Це особи, відповідальні за повсякденну роботу мікромережі, включаючи моніторинг її продуктивності, прийняття рішень про те, як виробляти та розподіляти електроенергію, а також підтримувати її надійність. СППР може надати операторам мікромережі цінну інформацію про поточний і майбутній стан мікромережі, допомагаючи їм приймати кращі рішення та оптимізувати її продуктивність.
- *Власники мікромережі.* Це особи чи організації, які володіють мікромережею та відповідають за її фінансову життєздатність. СППР може допомогти власникам мікромережі оптимізувати роботу мікромережі таким чином, щоб максимізувати її прибутковість або мінімізувати її експлуатаційні витрати.

- *Регуляторні органи.* Це державні установи, відповідальні за регулювання мікромережевої галузі. СППР може допомогти регуляторним органам забезпечити безпечну, надійну та ефективну роботу мікромереж.
- *Споживачі.* Це особи або підприємства, які використовують електроенергію з мікромережі. СППР може допомогти забезпечити споживачам доступ до надійної та доступної електроенергії.
- *Науково-дослідні інститути.* Ці стейкхолдери займаються розробкою нових технологій та методів управління мікрогрідами. Вони використовують систему підтримки прийняття рішень для тестування нових технологій та методів.
- *Інженерні компанії.* Ці стейкхолдери проектують та реалізують мікрогрід. Вони використовують систему підтримки прийняття рішень для аналізу ефективності мікрогрід та прийняття рішень щодо їхнього розвитку.
- *Фінансові установи.* Ці стейкхолдери інвестують у мікрогрід. Вони використовують систему підтримки прийняття рішень для оцінки ризиків та доходів від інвестицій у мікрогрід.

Окрім цих загальних зацікавлених сторін, залежно від конкретного застосування СППР можуть бути інші конкретні зацікавлені сторони. Наприклад, якщо СППР використовується для оптимізації використання відновлюваних джерел енергії, тоді зацікавлені сторони можуть включати виробників відновлюваної енергії та екологічні групи.

Конкретні зацікавлені сторони, залучені до розробки та використання СППР для режимів роботи мікромережі, залежатимуть від конкретних потреб СППР та конкретного контексту, в якому розробляється СППР. Однак при проектуванні та розробці СППР слід враховувати всіх перелічених вище зацікавлених сторін.

Інформація в СППР щодо режимів роботи мікросітки може включати:

- Дані про стан мікромережі, такі як поточний режим роботи, прогнози виробництва і споживання енергії, граничні потужності мережі.

- Дані про обмеження мікромережі, такі як вимоги до якості електроенергії, екологічні обмеження.
- Дані про альтернативні сценарії роботи мікромережі.

Інформаційні системи, що використовуються для підтримки прийняття рішень щодо режимів роботи мікромережі, повинні бути масштабованими і гнучкими. Це означає, що вони повинні бути здатні обробляти великі обсяги даних і адаптуватися до змін в конфігурації мікромережі.

1.4 Підходи до проектуваняння архітектури інформаційної системи

Архітектура інформаційної системи повинна забезпечувати ефективну взаємодію між різними компонентами системи, включаючи:

- Моделі роботи різних режимів мікромережі [10].
- Експертні системи [11], [12].
- Алгоритми штучного інтелекту [13].

З цієї точки зору існує декілька варіантів побудови архітектури інформаційної системи, а саме: мікросервісна, монолітна, хмарна, гібридна.

Мікросервісна архітектура є найбільш поширеним варіантом для інформаційних систем, що підтримують прийняття рішень щодо режимів роботи енергетичних мікромереж. Ця архітектура передбачає поділ системи на невеликі, самостійні модулі, які називаються мікросервісами. Мікросервіси можуть бути розроблені і підтримані різними командами, що дозволяє значно спростити розробку і масштабування системи.

Монолітна архітектура є більш традиційним варіантом. Ця архітектура передбачає, що вся система реалізована як єдиний модуль. Монолітні системи прості у розробці та підтримці, але вони є менш масштабованими і гнучкими, ніж мікросервісні системи.

Хмарна архітектура передбачає розміщення системи в хмарі. Цей варіант дозволяє істотно знизити витрати на обслуговування системи і забезпечити її доступність з будь-якої точки світу.

Гібридна архітектура є поєднанням мікросервісної, хмарної та/або крайової архітектур. Цей варіант дозволяє забезпечити найкращі характеристики з точки зору масштабованості, гнучкості та доступності.

Для вибору конкретного варіанту архітектури необхідно враховувати такі фактори, як: тип системи, обсяг даних, вимоги до масштабованості, вимоги до гнучкості, витрати на розробку і обслуговування.

Для інформаційних систем, що підтримують прийняття рішень щодо режимів роботи енергетичних мікромереж, найбільш оптимальним варіантом є мікросервісна архітектура. Ця архітектура дозволяє забезпечити необхідну масштабованість, гнучкість і безпеку системи.

Порівняння архітектурних рішень наведено у таблиці 1.1.

Таблиця 1.1 - Порівняння архітектурних рішень.

Джерело: побудовано автором

Характеристика	Мікросервісна архітектура	Монолітна архітектура	Хмарна архітектура	Гібридна архітектура	MVC
Централізоване управління	Ні	Так	Так	Так	Так
Розподіленість	Так	Ні	Так	Так	Ні
Розширюваність	Дуже висока	Середня	Дуже висока	Дуже висока	Середня
Модульність	Дуже висока	Середня	Середня	Середня	Середня
Здатність до масштабування	Дуже висока	Середня	Дуже висока	Дуже висока	Середня
Мобільність	Дуже висока	Середня	Дуже висока	Дуже висока	Середня
Безпека	Середня	Середня	Середня	Висока	Середня
Складність	Висока	Середня	Середня	Середня	Середня
Вартість	Середня	Низька	Висока	Середня	Середня
Де реалізовано	[14], [15]	[16]	[17]	[18]	[19]

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Об'єкт дослідження – підтримка прийняття рішень щодо управління режимами роботи енергетичної мікромережі.

Предмет дослідження – процес реалізації мікросервісу для підтримки прийняття рішень щодо управління режимами роботи енергетичної мікромережі.

Мета роботи полягає у розробці та імплементації мікросервісу, інтегрованого в систему підтримки прийняття рішень щодо режимів функціонування енергетичної мікромережі.

Практична цінність полягає у створенні мікросервісу, інтегрованого в систему підтримки прийняття рішень щодо режимів функціонування енергетичної мікромережі, що покращує гнучкість, легкість інтеграції та розширення системи. Крім того, воно відкриває перспективи для інших аспектів, таких як можливість інтеграції мікросервісу з іншими системами, надання програмного інтерфейсу (API) для створення власних додатків чи інтеграції з іншими платформами. Це робить систему більш адаптивною до змін і розширює її функціональність для впровадження різноманітних інновацій.

Для досягнення цієї мети необхідно вирішити наступні задачі:

1. Дослідження стану проблеми та перспектив розвитку систем підтримки прийняття рішень щодо режимів роботи енергетичних мікромереж. Ознайомлення з вимогами та підходами до проектування мікросервісів .Формування функціональних вимог.
2. Структурно-функціональне моделювання процесу створення та використання мікросервісу. Визначення ключових етапів та

взаємозв'язків між компонентами мікросервісу та системи в яку він інтегрується.

3. Програмна реалізація мікросервісу, відповідно до розробленої архітектури та функціональних вимог, а також забезпечення інтеграції мікросервісу у існуючу інформаційну систему.
4. Тестування та перевірка функціональності, продуктивності та стійкості мікросервісу.

2.2 Методи дослідження

Для вирішення поставлених задач використано наступні методи:

1. Аналітичний метод для вивчення питання актуальності, а саме пошук та аналіз релевантоної наукової літератури, виачення існуючих систем підтримки прийняття підтримки прийняття рішень, а також визначення функціональних вимог до мікросервісу.

2. Метод структурно-функціонального моделювання для визначення системи у вигляді взаємоповязаних функцій, як перетворюються вхідні та вихідні дані між функціями.

3. UML проектування для розробки діаграми варіантів використання для визначення основних акторів та ідентифікації їх взаємодії з системою. Діаграма послідовності для для відображення взаємодії між обєктами системи.

4. Методи та технології практичної реалізації мікросервісу та вебсистеми:

- Архітектура Мікросервісу. Мікросервіс побудовано на архітектурному принципі "багато малих сервісів". Кожен функціональний модуль відповідає конкретній відповідальності, що сприяє легкості розширення та підтримки коду.

- Робота з API. API реалізовано за допомогою FastAPI, використовуючи Pydantic для валідації та опису даних. Завдяки цьому, API є не тільки потужним, але й документованим, що спрощує його використання.
- Взаємодія з базою даних. SQLAlchemy [25] використовується для зручної реалізації взаємодії з базою даних. Всі дані про стан мікромережі, правила та інші необхідні дані зберігаються в реляційній базі, забезпечуючи надійність та стабільність системи.

5. Тестування та забезпечення безпеки. Тестування виконується з використанням Pytest [26]. Тестові сценарії охоплюють різні аспекти функціоналу, включаючи валідацію даних, правильність роботи API та збереження інформації в базі даних.

Для забезпечення безпеки використовуються стандартні практики, такі як захист від SQL-ін'єкцій, обмеження доступу до ресурсів та шифрування конфіденційної інформації.

3 ПРОЕКТУВАННЯ МІКРОСЕРВІСУ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ ЩОДО РЕЖИМІВ РОБОТИ ЕНЕРГЕТИЧНОЇ МІКРОМЕРЕЖІ З ВІДНОВЛЮВАНИМИ ДЖЕРЕЛАМИ ЕНЕРГІЇ

3.1Структурно-функціональне моделювання об’єкта дослідження

Нотація IDEF0 призначена для опису бізнес-процесів. Для кращого представлення останніх проект представляється у вигляді прямокутника [20]. Керуючі стрілки під’єднуються до його сторін. Кожна стрілка повинна відповідати за різний наступний тип даних:

– ліва стрілка – вхідні дані системи;

– права стрілка – вихідні дані системи;

– верхня стрілка – дані керування, тобто документ, що фіксує, як система повинна працювати та реалізовуватися;

– нижня стрілка – дані механізму, тобто персонал або програмне забезпечення, які залучаються для реалізації системи.

На рисунку 3.1 представлена контекстна діаграма процесу взаємодії користувача (експерта, що надає задає правила щодо режимів роботи мікромережі) з інформаційною системою, в яку інтегровано мікросервіс.

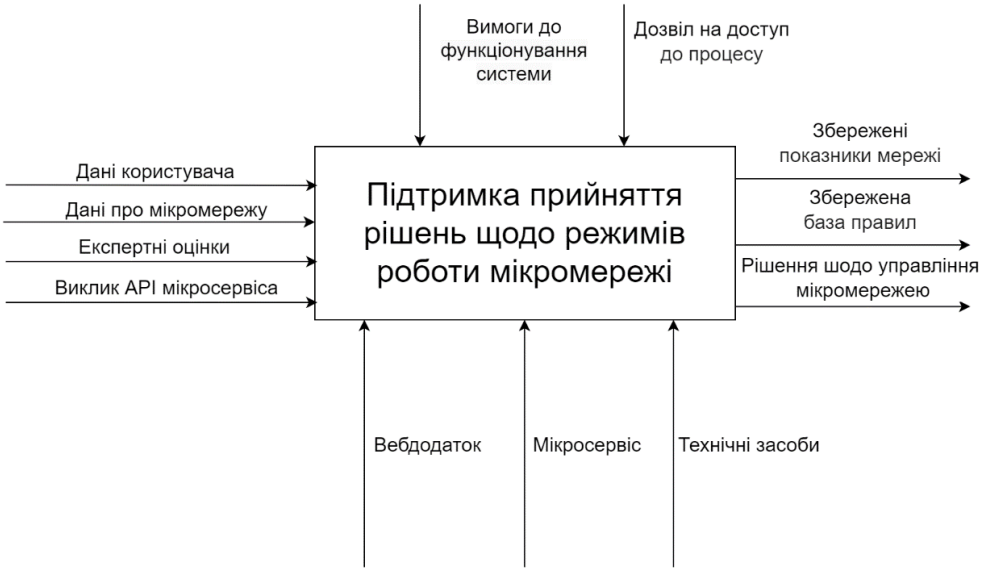


Рисунок 3.1 Контекстна діаграма IDEF0

Джерело: побудовано автором

Наступний крок – декомпозиція головної функції на підпроцеси. Основні дії, які можна виділити:

– збереження внесених експертом значень показників;

– визначення стану вимикача.

Після декомпозиції, розроблено діаграму першого рівня. Дані для діаграми наступні:

– вхідні дані – значення показників;

– вихідні дані – дані стану вимикачів на схемі;

– управління – користувач отримує інформацію для прийняття рішень;

– механізми – web-додаток підтримки управління мікромережею.

На рисунку 3.2 представлена діаграма першого рівня.

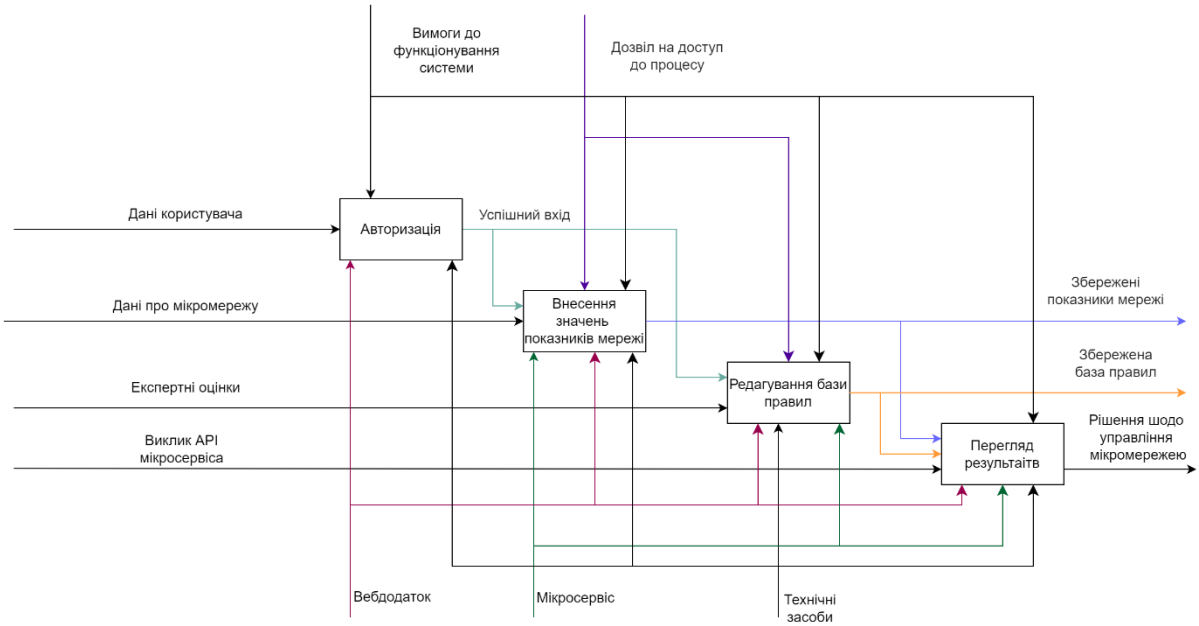


Рисунок 3.2 Діаграма декомпозиції першого рівня

Джерело: побудовано автором

3.2Діаграма варіантів використання

У діаграмі Use Case проводиться опис взаємодії користувача з системою [21]. Було визначено одного акторами – користувача (кінцевий користувач мікромережі) та експерта (особа, що надає знання). Також як актори виділені зовнішня база даних та мікросервіс. Нижче відображено список сценаріїв використання ним системи :

– уведення данних показників;

– визначення стану вимикача;

– отримання даних показників;

Розроблена діаграма Use Case на основі даних сценаріїв представлена на рисунку 3.3

Рисунок 3.3 Діаграма Use Case

Джерело: побудовано автором

3.3Взаємодія веб-додатка з мікросервісом

На рисунку 3.4 описана послідовність дій та процесів у мікросервісі, показує кроки та умови, які визначають, яка дія повинна бути виконана. Вказує на процеси та взаємодію зовнішніх факторів.

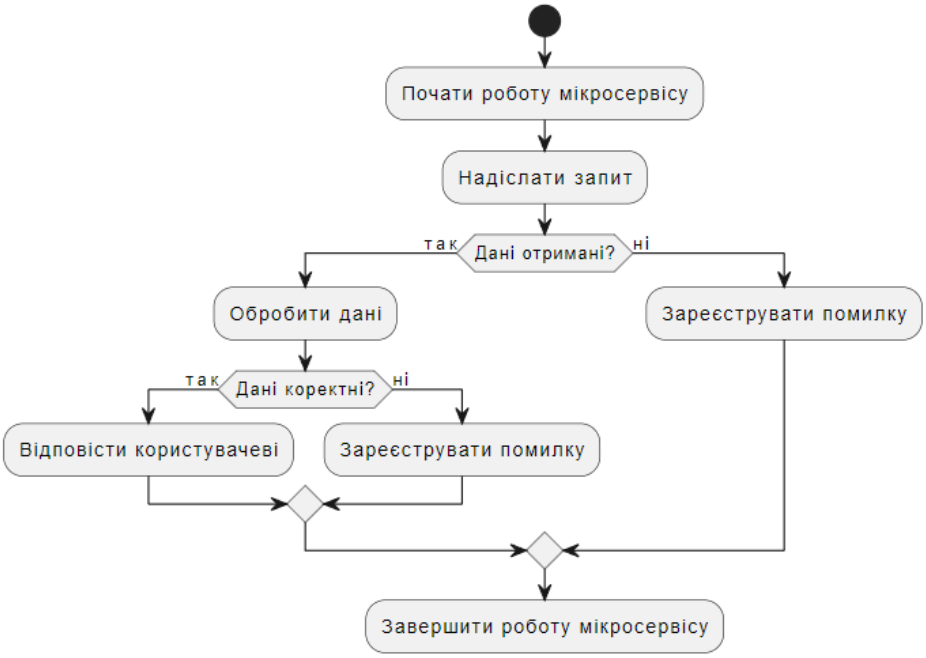


Рисунок 3.4 Діаграма послідовності

Джерело: побудовано автором

Діаграма послідовності [42] є одним із типів діаграм UML (Unified Modeling Language) і використовується для візуалізації взаємодії між різними об'єктами в системі в певний момент часу. Вона дозволяє показати послідовність обміну повідомленнями між об'єктами в рамках конкретного виконання сценарію або функції.

Основні елементи діаграми послідовності включають:

- Об'єкти: Представлені прямокутниками з ім'ям об'єкта, який вони представляють.

- Лінії життя (Lifelines): Відображають "життя" об'єкта та представляють часовий проміжок, протягом якого об'єкт існує чи виконує конкретну дію.
- Повідомлення: Стрілки, які вказують на обмін повідомленнями між об'єктами. Вони можуть бути напрямленими та мати асоційований з ними текст, що описує тип повідомлення.
- Фрейми: Використовуються для визначення контексту або межі виконання певної частини діаграми.

Діаграми послідовності допомагають розуміти порядок виконання різних елементів системи та взаємодію між ними. Це корисний інструмент для аналізу та проектування систем, які взаємодіють в реальному часі.

На рисунку 3.5 описана взаємодія веб-додатка з мікросервісом, показує кроки та умови, які визначають, яка дія повинна бути виконана.

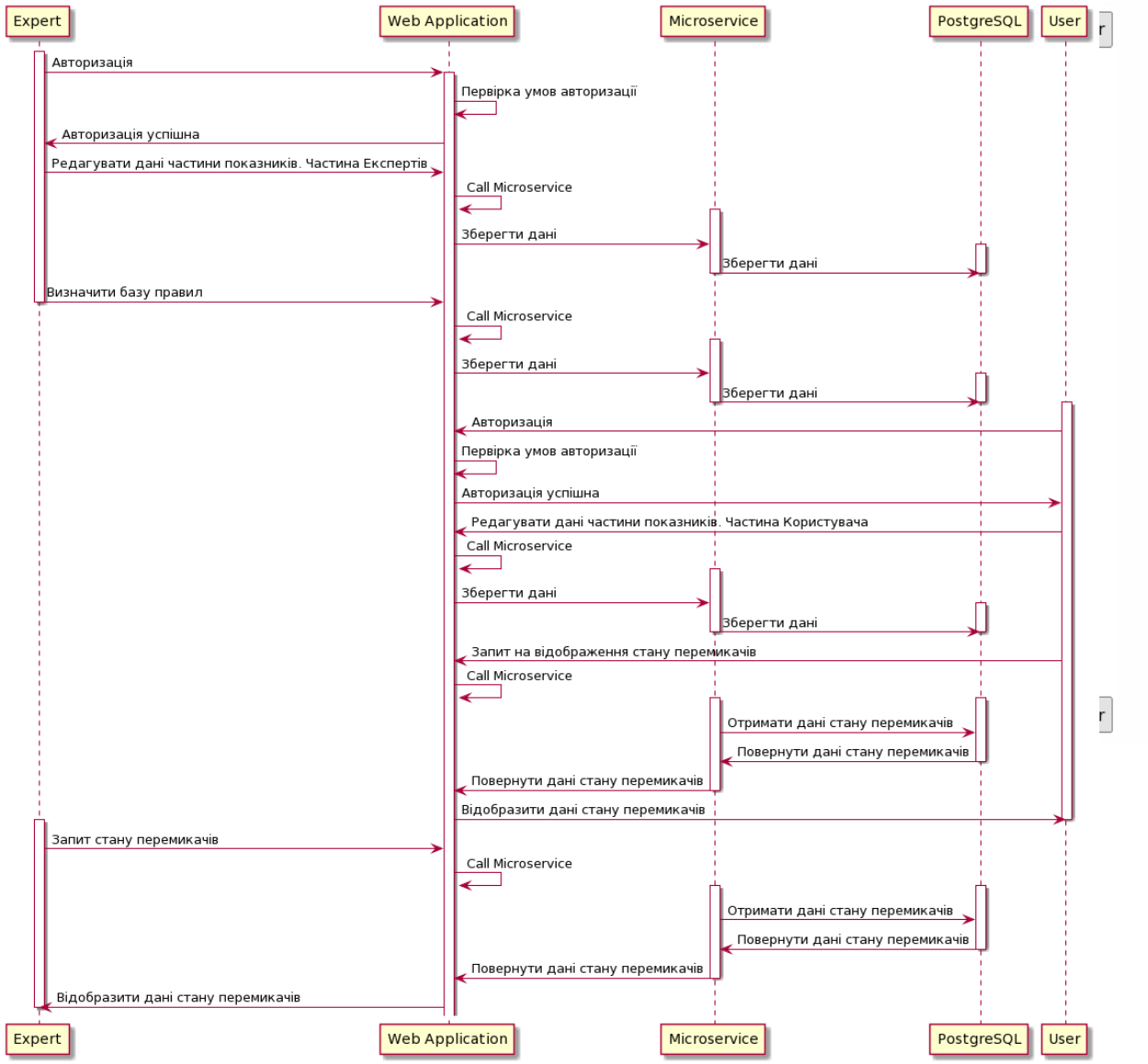


Рисунок 3.5 Діаграма послідовності

Джерело: побудовано автором

3.4Проектування бази даних

ER-діаграма (сутнісно-реляційна діаграма [41]) — це графічний інструмент, який використовується для відображення сутностей та їх взаємозв'язків у базі даних. Вона допомагає моделювати структуру даних та відносини між різними сутностями. Основною метою ER-діаграм є візуалізація, проектування та розуміння логічної структури бази даних перед її реалізацією.

У ER-діаграмі сутності зображуються у вигляді прямокутників, а взаємозв'язки — у вигляді ліній, що з'єднують ці сутності. Сутності представляють об'єкти, інформацію про які слід зберігати в базі даних, тоді як взаємозв'язки вказують на способи, якими ці об'єкти пов'язані між собою.

ER-діаграми є важливим інструментом для аналізу, проектування та реалізації баз даних, а також для взаємодії з різними зацікавленими сторонами у процесі розробки інформаційних систем.

ER-діаграма була розроблена для візуалізації структури та взаємозв'язків між сутностями у базі даних Рисунок 3.6.

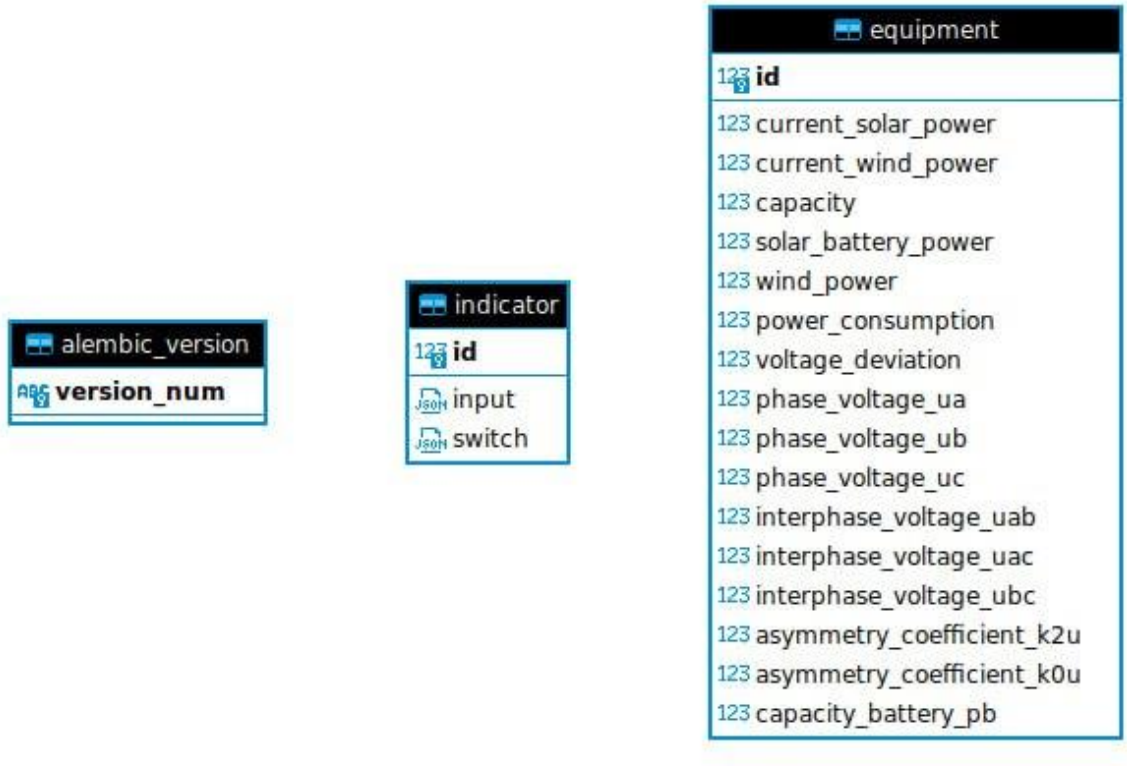


Рисунок 3.6 ER – діаграма бази даних
Джерело: побудовано автором

4 ПРОГРАМНА РОЗРОБКА МІКРОСЕРВІСУ

4.1Архітектура мікросервісу

Вибір мікросервісної архітектури для проекту обумовлено багатьма перевагами та можливостями, які вона пропонує. Мікросервіси дозволяють розділити систему на невеликі та автономні компоненти, що спрощує розробку, тестування та розгортання. Цей підхід полегшує масштабування окремих складових системи та забезпечує гнучкість у виборі технологій для кожного мікросервісу. Більш того, мікросервісна архітектура сприяє незалежному вдосконаленню та розгортанню окремих компонентів, що робить систему більш модульною та легко збереженною.

Архітектура додатку використовує сучасний та ефективний підхід, в якому API мікросервісу реалізовано з використанням фреймворку FastAPI [24, 40], а доступ до бази даних забезпечено за допомогою PostgreSQL [33].

FastAPI - це високопродуктивний фреймворк для створення веб-додатків з підтримкою автоматичної документації на основі стандарту OpenAPI. Він дозволяє швидко та ефективно розробляти API, використовуючи сучасні стандарти та механізми автоматизації.

PostgreSQL виступає в ролі бази даних для зберігання і управління даними. Це потужна та надійна реляційна база даних, яка підтримує розширені можливості та забезпечує ефективність обробки запитів.

Застосування фреймворку FastAPI дозволяє швидко побудувати надійне API, забезпечити автоматичну документацію та високий рівень продуктивності. Використання PostgreSQL гарантує надійність та ефективність роботи з базою даних, роблячи додаток готовим до масштабування та забезпечуючи стабільну роботу в умовах реального світу.

На рисунку 4.1 показано архітектуру розроблюваного додатку з урахуванням інтеграції мікросервісу.

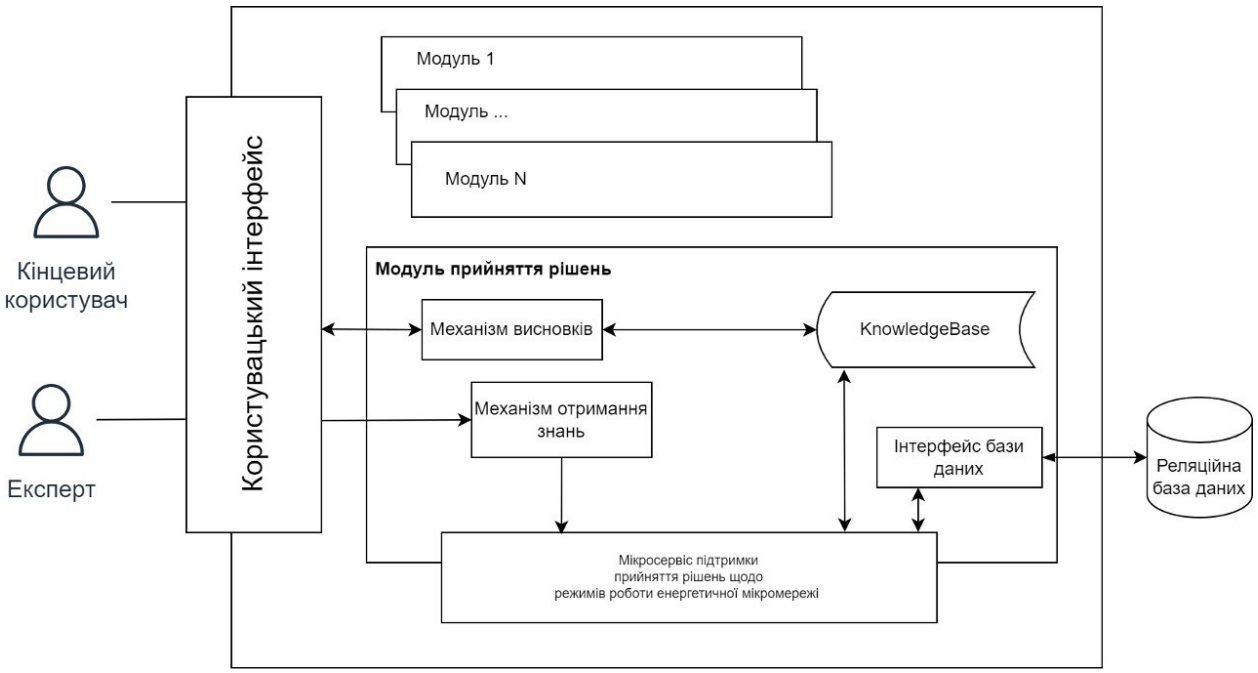


Рисунок 4.1 Архітектура додатку
Джерело: побудовано автором

4.2Розгортання віртуального середовища для локальної розробки

На початковому етапі робіт над мікросервісом, основний акцент робиться на налаштуванні середовища для локальної розробки за допомогою Docker [28]. Це включає встановлення Docker на робочу машину та створення конфігураційного файлу Dockerfile, що визначає, як буде побудований контейнер.

Далі проводиться конфігурація середовища, включаючи параметри та змінні середовища, які необхідні для функціонування мікросервісу. Після цього виконується запуск Docker-контейнера для локальної розробки, використовуючи створений образ, що дозволяє розробникам тестувати та вдосконалювати мікросервіс прямо на своєму комп'ютері.

Останній етап полягає в перевірці роботи мікросервісу у локальному Docker-середовищі для підтвердження правильності його розгортання та взаємодії з іншими компонентами системи. Це дозволяє ефективно розпочати роботу над функціональністю мікросервісу та його подальшим вдосконаленням.

На початковому етапі побудовано два контейнери для забезпечення локальної розробки мікросервісу. Перший контейнер включає в себе середовище Python та сервер uvicorn [37], необхідні для виконання мікросервісу. У цьому контейнері налаштовані всі необхідні залежності для коректного виконання коду мікросервісу на платформі Python [34].

Другий контейнер призначений для бази даних PostgreSQL. В ньому розгорнута та налаштована система управління базою даних, що дозволяє мікросервісу взаємодіяти з базою данних для збереження та отримання необхідної інформації.

Ці два контейнери працюють разом, створюючи ізольоване середовище для розробки та тестування мікросервісу, що дозволяє зручно виконувати роботу та вдосконалювати функціонал у локальному режимі.

```
services:
  itenergy-service:
    build: .
    container_name: itenergy
    depends_on:
      - itenergy-service-postgres
    volumes:
      - ${PWD}:/app
    command: ["uvicorn", "itenergy.app:app", "--host", "0.0.0.0", "--port", "8000"]

  itenergy-service-postgres:
    container_name: itenergy-service-postgres
    image: postgres:14.2
    environment:
      PGHOST: itenergy-service-postgres
      PGPORT: 5432
      POSTGRES_DB: itenergy
      POSTGRES_USER: itenergy
      POSTGRES_PASSWORD: password
    ports:
      - 5433:5432
```

4.3Розробка структури мікросервісу

Структура мікросервісу була розроблена та відображена на відповідному рисунку. У рамках цього проекту визначено компоненти, їх взаємозв'язки та ролі в системі, що допомагає зрозуміти організацію та взаємодію елементів мікросервісної архітектури. Представлена на Рисунку 4.2

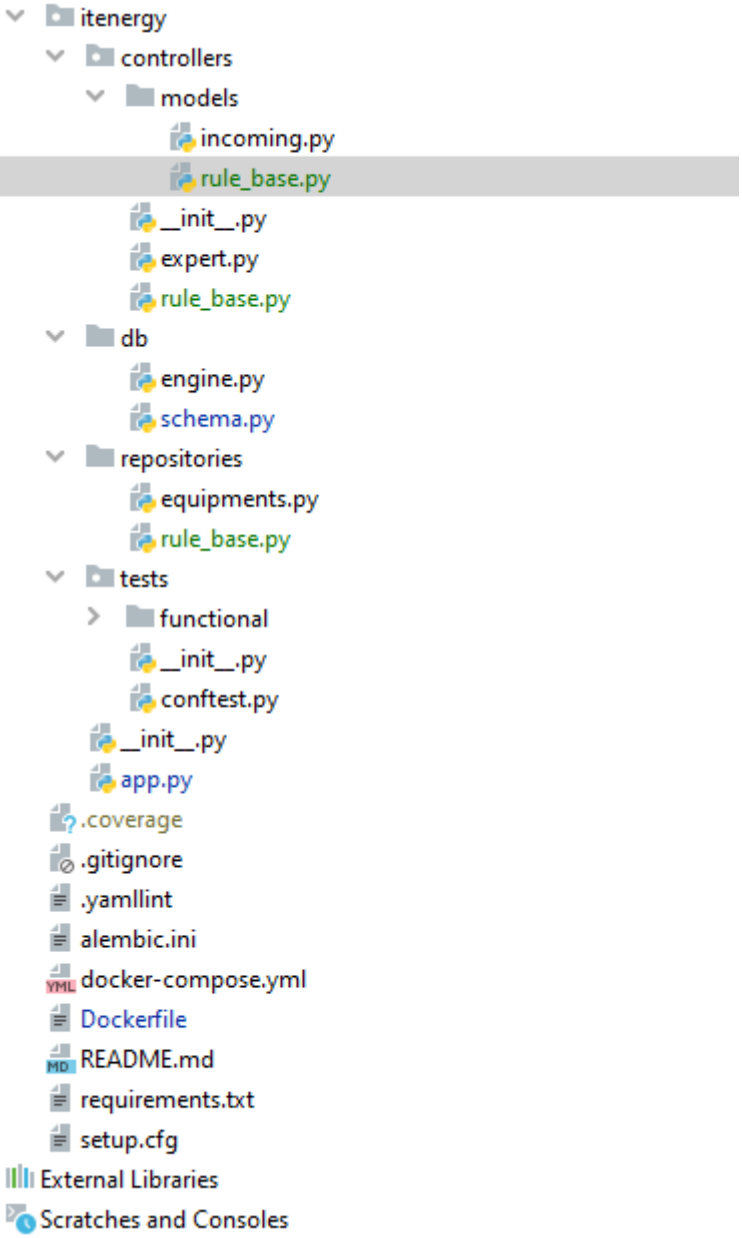


Рисунок 4.2 Архітектура додатку
Джерело: побудовано автором

В проекті було використано валідацію моделей Pydantic для перевірки та забезпечення відповідності даних в запитах визначеним умовам. Використання Pydantic для валідації моделей дозволяє переконатися, що дані, які надходять у ваш додаток, відповідають очікуваному формату та типам даних. Це сприяє підвищенню надійності та безпеки програми, допомагаючи уникнути можливих помилок у введених даних.

Основні переваги використання валідації моделей Pydantic включають:

Визначення Типів та Правил Валідації: Pydantic дозволяє явно вказувати типи даних для кожного поля моделі, а також встановлювати правила валідації, такі як мінімальна/максимальна довжина рядка, мінімальне/максимальне значення числових полів та інше.

Автоматична Генерація Документації: Pydantic може автоматично генерувати документацію, що вказує на правила та обмеження для кожного поля моделі, що полегшує розуміння, як користувачі повинні будувати свої запити.

Перевірка Валідності Даних: Pydantic автоматично перевіряє валідність даних при їх передачі у модель, і видає виняток у разі невідповідності заданим умовам. Це допомагає виявляти помилки вводу даних на ранніх етапах розробки.

Підтримка Автоматичної Серіалізації та Десеріалізації: Pydantic надає функціонал для автоматичної серіалізації об'єктів у формат JSON та інші формати, а також для їх автоматичної десеріалізації.

Код використовує модуль BaseModel з бібліотеки Pydantic для опису схеми запиту (request schema) EquipmentRequest. Ця схема визначає структуру даних, яку можна використовувати для валідації та десеріалізації даних, що надходять у запиті.

Основні риси цього коду: Наслідування від BaseModel: Клас EquipmentRequest успадковує властивості та методи з BaseModel, що дозволяє

використовувати можливості валідації та генерації документації, які надає Pydantic.

Опис полів: Кожне поле визначене як атрибут класу EquipmentRequest представляє параметр, який може бути переданий у запиті. Кожне поле є числовим типом (int або float) або None, що дозволяє йому бути необов'язковим.

Типи данних з Pydantic: Використання оператора | для визначення альтернативних типів даних (у цьому випадку, float або None). Це дозволяє встановити, що значення полів може бути або числовим, або None.

Значення за замовчуванням: Вказані значення за замовчуванням для кожного поля (None), що вказує, що це поле є необов'язковим і може бути відсутнім в запиті. Нижче наведено код відповідний за валідацію.

```
class RuleBaseRequest(BaseModel):
    id: int
    input: Any
    switch: Any
class EquipmentRequest(BaseModel):
    id: int
    voltage_deviation: float | None = None
    phase_voltage_ua: float | None = None
    phase_voltage_ub: float | None = None
    phase_voltage_uc: float | None = None
    interphase_voltage_uab: float | None = None
    interphase_voltage_uac: float | None = None
    interphase_voltage_ubc: float | None = None
    asymmetry_coefficient_k2u: float | None = None
    asymmetry_coefficient_k0u: float | None = None
    capacity_battery_pb: float | None = None
    current_solar_power: float | None = None
    current_wind_power: float | None = None
    capacity: float | None = None
    solar_battery_power: float | None = None
    wind_power: float | None = None
    power_consumption: float | None = None
```

4.4Демонстрація роботи мікросервісу

На наступному етапі для розробки мікросервісу був обраний фреймворк FastAPI. Використання FastAPI забезпечує високу швидкодію та простоту розробки API. Основні переваги FastAPI включають автоматичну генерацію

документації, підтримку стандарту OpenAPI (Swagger), автоматичну валідацію даних, асинхронність та інші. Цей фреймворк дозволяє ефективно реалізовувати мікросервіс та працювати з HTTP-запитами, надаючи швидкий та надійний інструмент для створення API.

Окрім цього, FastAPI автоматично генерує Swagger UI для API, що спрощує взаємодію з мікросервісом. Swagger UI — це інтерактивна документація, яка дозволяє розробникам та іншим зацікавленим сторонам взаємодіяти з API, тестувати його та отримувати інформацію щодо доступних ендпойнтів, параметрів, типів даних тощо, всі ці можливості забезпечують зручність та прозорість у використанні мікросервісу.

Представлена на Рисунку 4.3

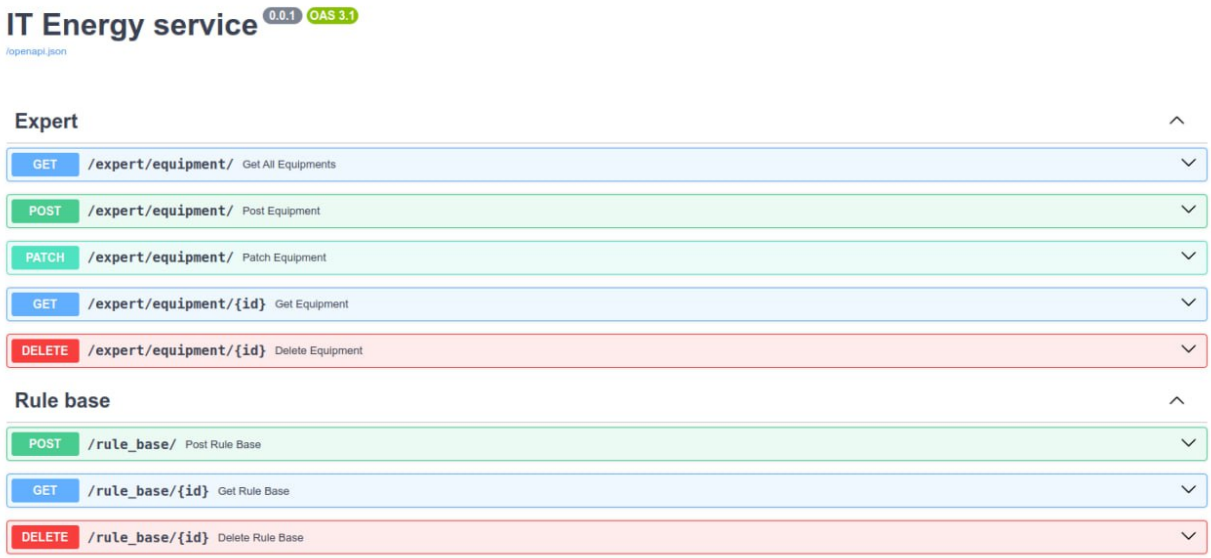


Рисунок 4.3 Тестовий інтерфейс мікросервіса
Джерело: побудовано автором

Наступним єтапом був розроблений стандартний набір ендпойнтів для здійснення операцій CRUD (створення, читання, оновлення, видалення) у мікросервісі. Ці ендпойнти дозволяють взаємодіяти з базою даних та виконувати різні операції з ресурсами, представленими в мікросервісі.

Створено (Create) ендпоінт для створення нових записів про обладнання. Це означає, що користувач може відправити дані про нове обладнання до серверу, і ці дані будуть збережені в базі даних. Створює новий ресурс у базі даних на основі переданих даних.

Також розроблено (Read) ендпоінт для отримання інформації про обладнання. За допомогою цього ендпоінта користувач може отримати всі наявні дані про конкретне обладнання з бази даних. Отримує дані конкретного ресурсу за його унікальним ідентифікатором (id).

Для оновлення існуючих даних про обладнання створено відповідний (Update) ендпоінт. Користувач може відправити нові дані для конкретного обладнання, і вони будуть оновлені в базі даних. Отримує дані у вигляді списку всіх ресурсів

Також доступний (Delete) ендпоінт для видалення записів про обладнання. Користувач може надіслати запит на видалення конкретного обладнання, і відповідний запис буде вилучено з бази даних. Видаляє ресурс з бази даних за його унікальним ідентифікатором (id).

Цей стандартний набір ендпойнтів забезпечує базовий функціонал для керування ресурсами у мікросервісі та взаємодії з базою даних.

На рисунку 4.4 показаний приклад роботи ендпоінта GET для отримання списку показників всіх установок.

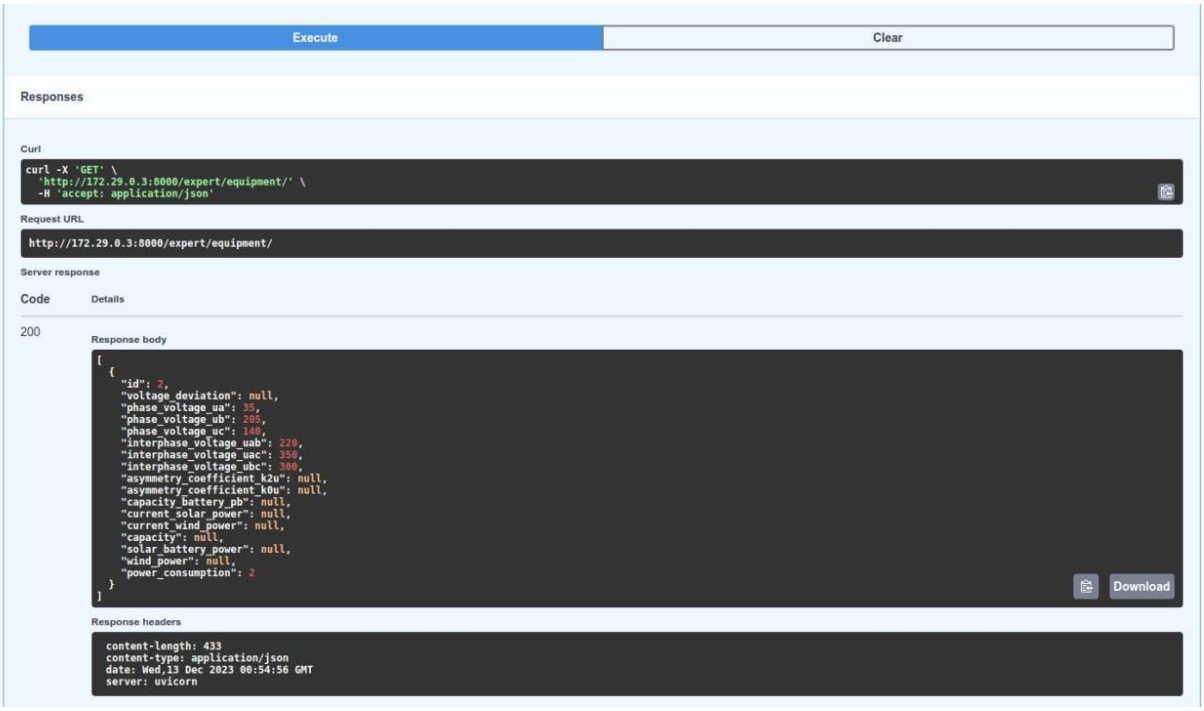


Рисунок 4.4 Приклад роботи ендпоінта GET

Джерело: побудовано автором

У результаті модифікацій коду веб-додатка були внесені незначні зміни у відображення форм опитування експертів (рис. 4.5), та додані API виклики мікросервісу що дозволяють взаємодіяти з мікросервісом.

Рисунок 4.5 Інтерфейс веб-додатку

Джерело: побудовано автором

На рисунку 4.6 показаний приклад роботи ендпоінта PATCH для оновлення даних ресурсу по id.

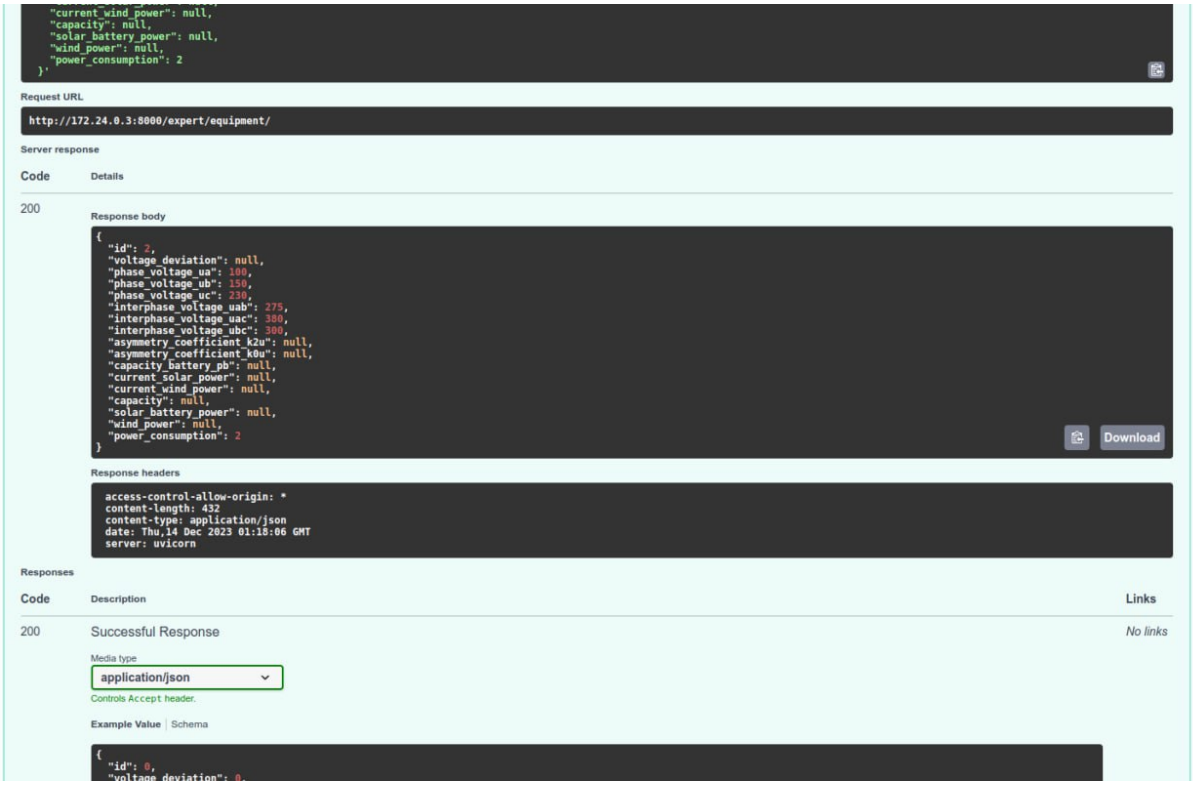


Рисунок 4.6 Приклад роботи ендпоінта PATCH

Джерело: побудовано автором

На рисунку 4.7 показаний приклад роботи ендпоінта DELETE для видалення ресурсу по id.

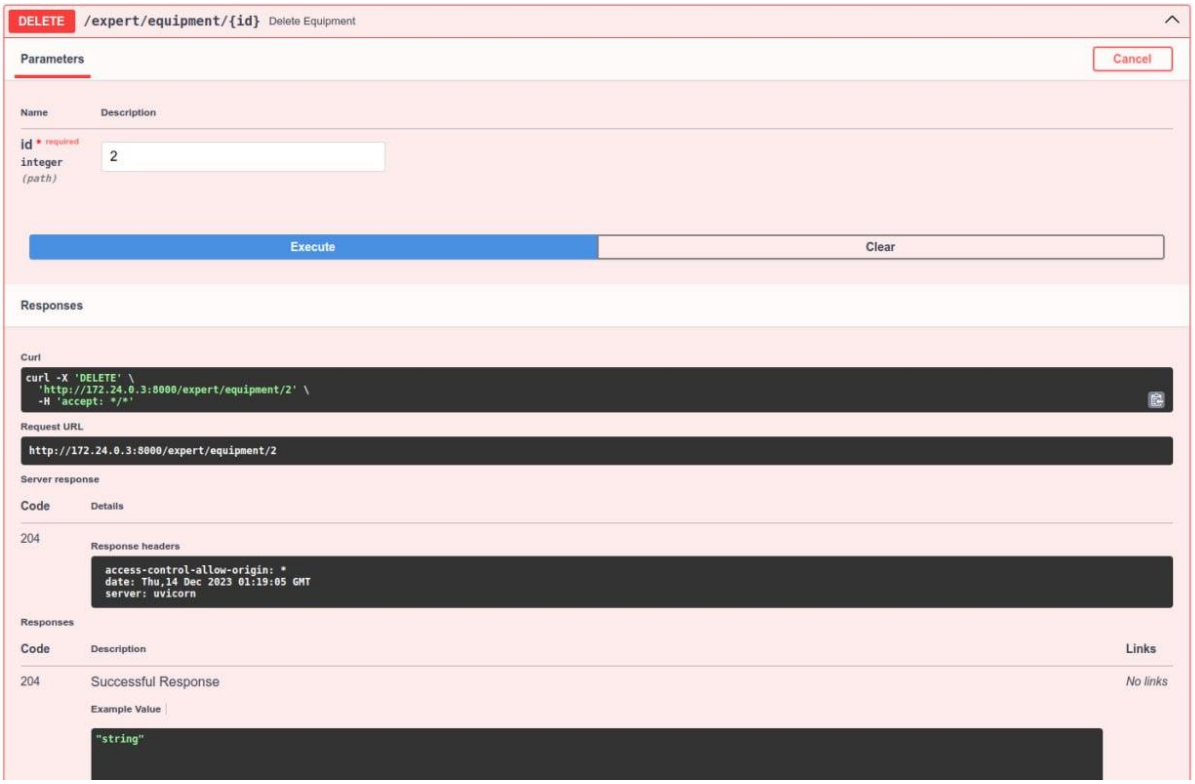


Рисунок 4.7 Приклад роботи ендпоінта GET
Джерело: побудовано автором

4.5Тестування мікросервісу за допогою pytest

На наступному етапі розробки був використаний інструмент Pytest для тестування API мікросервісу. Pytest - це потужний фреймворк для тестування у мові програмування Python, який надає зручний та зрозумілий синтаксис для написання тестів. Він підтримує автоматичне виявлення тестів та їх виконання, а також надає різноманітні можливості для організації тестових наборів та валідації результатів.

Pytest є відмінним вибором для тестування API через його простий синтаксис, широкі можливості для параметризації тестів та легкість інтеграції з іншими інструментами [35].

Параметр покриття тестами на рівні 100% означає, що кожна частина коду, яка виконує яку-небудь функціональність, була покрита тестами.

Забезпечення 100% покриття тестами покликане зменшити кількість помилок, підвищити стабільність системи та полегшити рефакторинг коду.

Налаштування такого високого рівня покриття тестами дозволяє впевнитися, що кожна зміна в коді піддається тестуванню, що в свою чергу сприяє стабільності та надійності мікросервісу. Приклад роботи pytest відображено на Рисунку 4.8

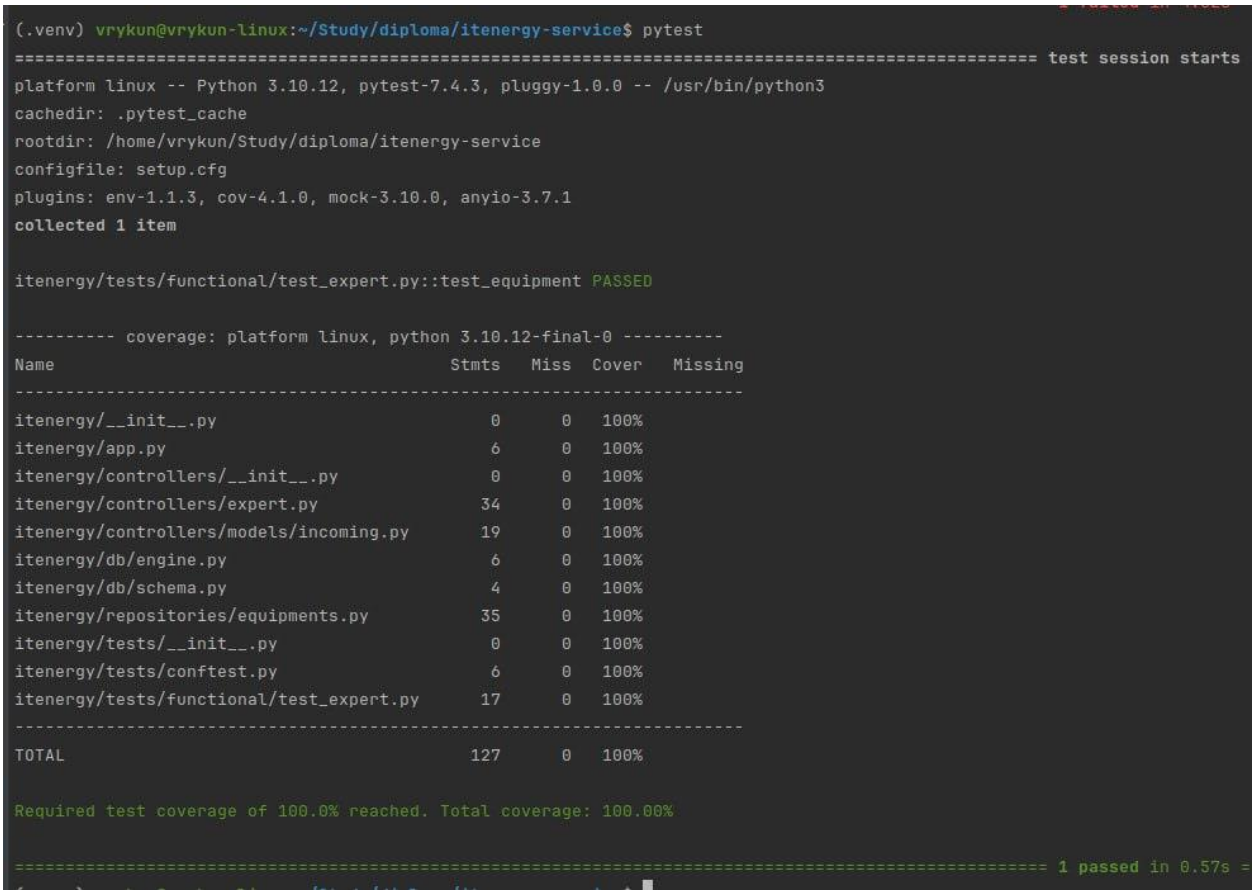


Рисунок 4.8 Результат роботи pytest

Джерело: побудовано автором

GitHub Actions - це сервіс автоматизації завдань, який надається GitHub [36]. Він дозволяє автоматизувати різноманітні робочі процеси в репозиторії, такі як тестування коду, збирання проекту, розгортання або взаємодія з іншими сервісами.

Використання GitHub Actions є важливим з кількох причин:

Автоматизація рутинних задач: GitHub Actions дозволяє автоматизувати багато рутинних завдань, звільняючи розробників від необхідності виконувати їх вручну.

Інтеграція з CI/CD: Ви можете легко налаштувати неперервну інтеграцію (CI) та неперервне розгортання (CD) для свого проекту, щоб автоматично перевіряти, тестувати та розгортати зміни коду.

Стандартизація робочих процесів: GitHub Actions дозволяє визначати та зберігати робочі процеси як код, що сприяє стандартизації та легкості управління.

Спільна робота: Використання GitHub Actions дозволяє команді однаково розуміти, як працюють різні скрипти та автоматичні задачі, сприяючи спільній роботі та зменшенню можливих конфліктів.

У GitHub Actions були додані завдання для автоматичної перевірки інсталяції всіх сторонніх модулів проекту, що дозволяє впевнитися в коректності налаштувань залежностей. Також була налаштована перевірка коду за допомогою додаткових бібліотек для визначення його відповідності стандартам та кращим практикам програмування. Це спрямовано на забезпечення якості коду, виявлення можливих проблем та вдосконалення загального стандарту програмного забезпечення.

Для перевірки коду були використані наступні інструменти:

- isort: Цей інструмент відповідає за автоматичне сортування імпортів у файлі згідно з встановленими стандартами та рекомендаціями [29].
- flake8: Flake8 використовується для забезпечення відповідності коду встановленим стандартам PEP 8 та виявлення можливих стилістичних і синтаксичних помилок [31].
- mypy: Цей інструмент відповідає за статичний аналіз коду та перевірку типів, що дозволяє виявляти можливі помилки та непослідовності в типах зазначених об'єктів [32].
- yamllint: Використовується для перевірки синтаксису та стилю YAML-файлів, забезпечуючи правильність форматування та уніфікований стиль у конфігураційних файлах [30].

Приклад роботи Github Actions відображено на Рисунку 4.9

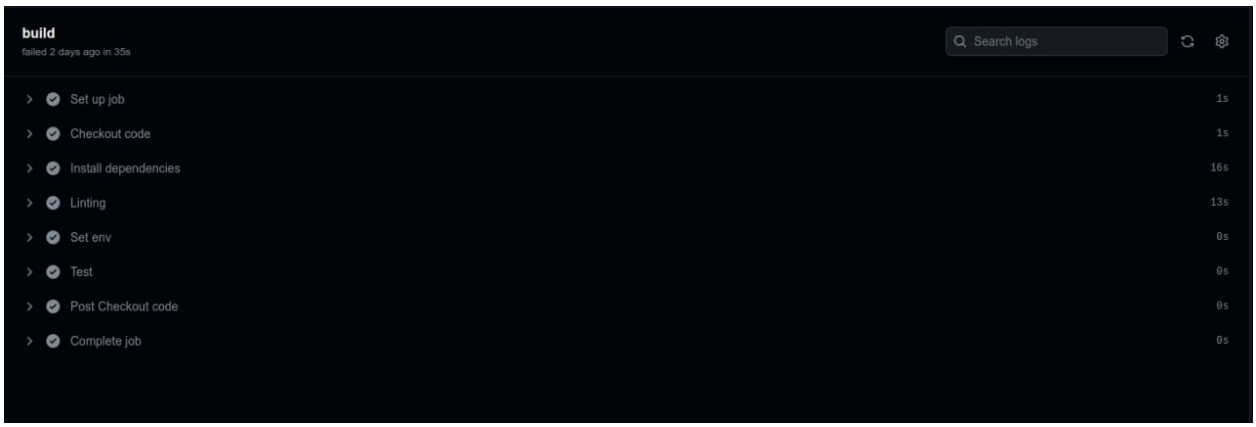


Рисунок 4.9 Результат роботи Github Actions

Джерело: побудовано автором

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра було досліджено сучасний стан та перспективи розвитку енергетичних мікромереж. Виявлено, що енергетичні мікромережі є перспективним напрямком у сфері енергетики, оскільки вони сприяють підвищенню надійності електропостачання, зменшенню викидів шкідливих речовин і сприяють більш ефективному використанню. На даному етапі виконано ряд ключових завдань, спрямованих на створення та реалізацію мікросервісу підтримки прийняття рішень для енергетичної мікромережі. Отримані результати можна узагальнити наступним чином:

Виконано аналіз актуальних проблем в енергетичних мікромережах та визначено перспективи розвитку галузі. Розроблено модель створення та функціональності мікросервісу, визначено структуру та взаємодію компонентів системи.

Створено мікросервіс відповідно до архітектурних та функціональних вимог. Успішно здійснено інтеграцію мікросервісу у вже існуючу інформаційну систему. Проведено ретельне тестування, включаючи випробування функціональності, продуктивності та стійкості.

Набуті знання та результати роботи на даному етапі становлять основу для подальшого розвитку та вдосконалення мікросервісу з метою виконання його основної функції – прийняття рішень щодо режимів роботи енергетичної мікромережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О. Трибой, “Сприяння енергетичній безпеці та сталому розвитку місцевих громад в Україні,” 2021.

2. M. A. Jirdehi, V. S. Tabar, S. Ghassemzadeh, and S. Tohidi, “Different aspects of microgrid management: A comprehensive review,” *J Energy Storage*, vol. 30, 2020, doi: 10.1016/j.est.2020.101457.

3. L. Ahmethodzic and M. Music, “Comprehensive review of trends in microgrid control,” *Renewable Energy Focus*, vol. 38. 2021. doi: 10.1016/j.ref.2021.07.003.

4. S. K. Rangu, P. R. Lolla, K. R. Dhenuvakonda, and A. R. Singh, “Recent trends in power management strategies for optimal operation of distributed energy resources in microgrids: A comprehensive review,” *International Journal of Energy Research*, vol. 44, no. 13. 2020. doi: 10.1002/er.5649.

5. M. O. Onibonoje, O. O. Alegbeleye, and A. O. Ojo, “Control Design and Management of a Distributed Energy Resources System,” *International Journal of Technology*, vol. 14, no. 2, 2023, doi: 10.14716/ijtech.v14i2.5884.

6. A. Mohammadpour Shotorbani, S. Zeinal-Kheiri, G. Chhipi-Shrestha, B. Mohammadi-Ivatloo, R. Sadiq, and K. Hewage, “Enhanced real-time scheduling algorithm for energy management in a renewable-integrated microgrid,” *Appl Energy*, vol. 304, 2021, doi: 10.1016/j.apenergy.2021.117658.

7. S. P. Bihari *et al.*, “A Comprehensive Review of Microgrid Control Mechanism and Impact Assessment for Hybrid Renewable Energy

Integration,” *IEEE Access*, vol. 9. 2021. doi: 10.1109/ACCESS.2021.3090266.

8. С. Шендрик, “Моделі та інформаційна технологія підтримки прийняття рішень при управлінні гібридними енергомережами,” 2020.

9. R. Trivedi and S. Khadem, “Implementation of artificial intelligence techniques in microgrid control environment: Current progress and future scopes,” *Energy and AI*, vol. 8. 2022. doi: 10.1016/j.egyai.2022.100147.

10. A. Cagnano, E. De Tuglie, and P. Mancarella, “Microgrids: Overview and guidelines for practical implementations and operation,” *Applied Energy*, vol. 258. 2020. doi: 10.1016/j.apenergy.2019.114039.

11. M. Hamidi, O. Bouattane, and A. Raihani, “Microgrid energy management system: Technologies and architectures review,” in *Proceedings - 2020 IEEE International Conference of Moroccan Geomatics, MORGEO 2020*, 2020. doi: 10.1109/Morgeo49228.2020.9121885.

12. D. Rodríguez, A. Angulo, D. F. Gómez, D. Álvarez, and S. Rivera, “Smart microgrids operation considering expert knowledge and ensembled based metaheuristic optimization algorithms,” *Far East Journal of Mathematical Sciences (FJMS)*, vol. 140, 2022, doi: 10.17654/0972087123001.

13. T. Wu and J. Wang, “Artificial intelligence for operation and control: The case of microgrids,” *Electricity Journal*, vol. 34, no. 1, 2021, doi: 10.1016/j.tej.2020.106890.

14. J. A. A. Silva, J. C. López, C. P. Guzman, N. B. Arias, M. J. Rider, and L. C. P. da Silva, “An IoT-based energy management system for AC microgrids with grid and security constraints,” *Appl Energy*, vol. 337, 2023, doi: 10.1016/j.apenergy.2023.120904.

15. I. Georgievski, L. Fiorini, and M. Aiello, “Towards Service-Oriented and Intelligent Microgrids,” in *ACM International Conference Proceeding Series*, 2020. doi: 10.1145/3378184.3378214.

16. R. H. Lasseter and P. Paigi, “Microgrid: A conceptual solution,” in *PESC Record - IEEE Annual Power Electronics Specialists Conference*, 2004. doi: 10.1109/PESC.2004.1354758.

17. M. Numair, D. E. A. Mansour, and G. Mokryani, “A Proposed IoT Architecture for Effective Energy Management in Smart Microgrids,” in *2nd Novel Intelligent and Leading Emerging Sciences Conference, NILES 2020*, 2020. doi: 10.1109/NILES50944.2020.9257923.

18. M. Dabbaghjamanesh, A. Moeini, A. Kavousi-Fard, and A. Jolfaei, “Real-time monitoring and operation of microgrid using distributed cloud–fog architecture,” *J Parallel Distrib Comput*, vol. 146, 2020, doi: 10.1016/j.jpdc.2020.06.006.

19. А. Сокрута, “Web-додаток підтримки управління мікромережею з відновлюваними джерелами енергії,” Сумський державний університет, Суми, 2022.

20. The Complete Guide to Understand IDEF Diagram. URL: https://www.edrawmax.com/article/the-complete-guide-to-understand-idef-diagram.html (дата звернення: 01.12.2023)

21. Use Case Diagram Tutorial (Guide with Examples). URL: https://creately.com/blog/diagrams/use-case-diagram-tutorial (дата звернення: 01.12.2023)

22. Структура розбиття робіт (Work Breakdown Structure - WBS). URL: https://www.maxzosim.com/struktura-rozbittia-robit/ (дата звернення: 01.12.2023)

23. Побудова організаційної структури (OBS). URL: https://studfile.net/preview/9721264/page:4/ (дата звернення: 01.12.2023)

24. Tutorial - User Guide. URL: https://fastapi.tiangolo.com/tutorial/ (дата звернення: 01.12.2023)

25. SQLAlchemy 2.0 Documentation. URL:https://docs.sqlalchemy.org/en/20/ (дата звернення: 01.12.2023)

26. Pytest: helps you write better programs. URL:https://docs.pytest.org/en/7.4.x/ (дата звернення: 01.12.2023)

27. The Migration Environment. URL:https://alembic.sqlalchemy.org/en/latest/ (дата звернення: 01.12.2023)

28. Docker Compose overview. URL: https://docs.docker.com/compose/ (дата звернення: 01.12.2023)

29. Read Latest Documentation. URL: https://pycqa.github.io/isort/ (дата звернення: 01.12.2023)

30. Yamllint documentation URL: https://yamllint.readthedocs.io/en/stable/ (дата звернення: 01.12.2023)

31. Flake8: Your Tool For Style Guide Enforcement URL: https://flake8.pycqa.org/en/latest/

32. Getting started URL: https://mypy.readthedocs.io/en/stable/getting_started.html (дата звернення: 01.12.2023)

33. PostgreSQL 16.1 URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 01.12.2023)

34. Python 3.11.7 documentation. URL: https://docs.python.org/3.11/ (дата звернення: 01.12.2023)

35. Full pytest documentation. URL: https://docs.pytest.org/en/7.1.x/contents.html (дата звернення: 01.12.2023)

36. GitHub Actions documentation. URL: https://docs.github.com/en/actions

37. Quickstart. URL: https://www.uvicorn.org/#quickstart (дата звернення: 01.12.2023)

38. Git -Reference Manual. URL: https://git-scm.com/docs/git (дата звернення: 01.12.2023)

39. Markdown Cheat Sheet. URL: https://www.markdownguide.org/cheat-sheet/ (дата звернення: 01.12.2023)

40. Microservice in Python using FastAPI. URL: https://dev.to/paurakhsharma/microservice-in-python-using-fastapi-24cc (дата звернення: 01.12.2023)

41. What is an Entity Relationship Diagram (ERD). URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 01.12.2023)

42. Діаграма послідовності (Sequence Diagrams). URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 01.12.2023)

ДОДАТОК А

Деталізація мети проекту методом SMART. Продуктом дипломного проекту є мікросервіс мікросервіс підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі з відновлюваними джерелами енергії.

Джерело: побудовано автором

Таблиця А.1 − Деталізація мети методом SMART

Джерело: побудовано автором

Specific (конкретна)	Розробка мікросервісу для системи управління енергетичною мікромережею, спрямованого на забезпечення ефективного моніторингу та управління за допомогою відновлюваних джерел енергії.
Measurable (вимірювана)	Результатом проекту є створений мікросервіс, який забезпечує зниження часу, потрібного для ухвалення управлінських рішень.
Achievable (досяжна)	Реалізація мікросервісу виконується за допомогою мови програмування Python та фреймворку FastAPI, що забезпечує ефективність та досяжність поставлених завдань. Розробник має достатні кваліфікації для виконання задач проекту.
Relevant (реалістична)	Використання розробленого мікросервісу сприятиме підвищенню ефективності управління енергетичними мікромережами та зменшенню впливу на навколишнє середовище.
Time-framed (обмежена у часі)	Мета проекту має чітко визначений термін виконання. Робота повинна відбуватися в рамках обумовлених замовником термінів та відповідати календарному плану проекту.

Планування змісту структури робіт. Основним інструментом для планування змісту проекту використовує методологію WBS (Work Break Structure [22]), яка дозволяє подробно розкласти проект на ієрархічні етапи та результати. Цей інструмент спрямований на детальне планування, оцінку вартості, визначення відповідальності та інші аспекти, що визначають зміст проекту.

На верхньому рівні WBS визначено сам проект: "Мікросервіс підтримки прийняття рішень щодо режимів роботи енергетичної мікромережі". Цей проект подробно розкладено на чотири рівні: ініціалізація, планування, реалізація та завершення.

Ініціалізація додатку включає етапи ознайомлення з предметною областю, визначення потреби в додатку та ідентифікацію ідеї проекту.

На рівні планування проект розбивається на два рівні:

1. Аналіз предметної області

2. Визначення вимог

- визначити засоби реалізації
- планування WBS
- планування OBS
- складання календарного плану

3. Реалізація проекту має 3 рівні:

– Моделювання роботи мікросервісу.

– Розробка функціоналу мікросервісу.

– Тестування:

4. Завершення

- тестування розробником
- написання автотестів

На рисунку А.1 представлено WBS з розробки мікросервісу підтримки управління мікромережею з відновлюваними джерелами енергії.

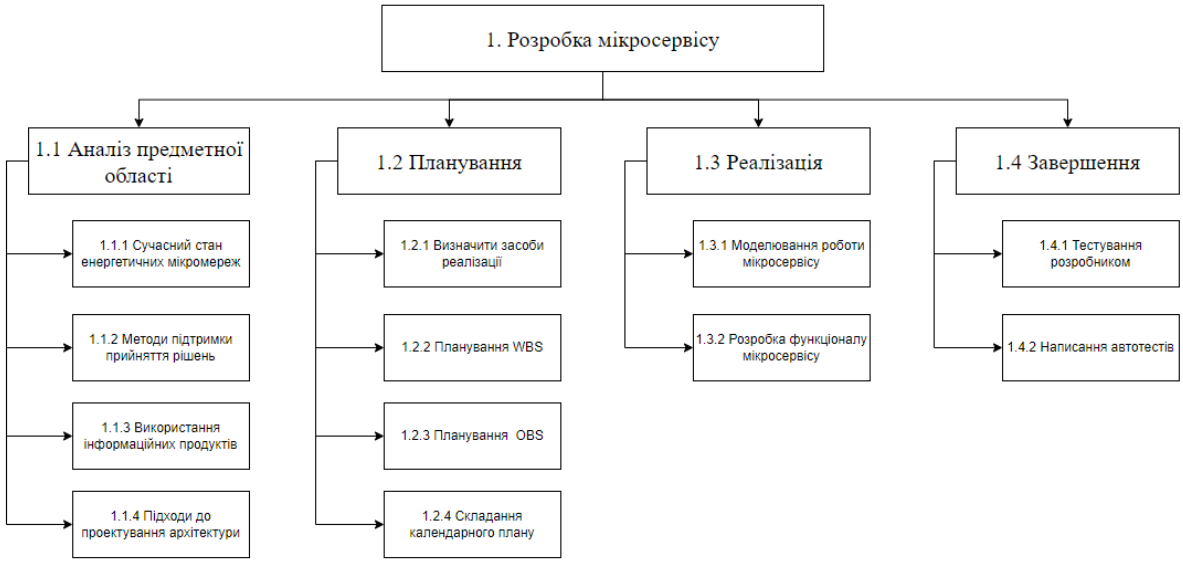


Рисунок А.1 – WBS-структура проекту

Джерело: побудовано автором

Створення структури організації для впровадження готового проекту (OBS) – це наступний крок після побудови WBS проекту. На цьому етапі розробляється OBS (Organization Breakdown Structure [23]) – це система, що визначає структуру, підпорядкування, взаємодію та розподіл обов'язків між підрозділами та органами управління. У цій системі встановлюються відносини з питань владних повноважень, командних потоків та обміну інформацією. Організаційна структура проекту визначає внутрішню організацію проекту і не включає відносин між проектними групами або учасниками та батьківськими організаціями. Список виконавців, що беруть участь у проекті, представлений у таблиці А.2. Організаційну структуру проекту можна побачити на рисунку А.2.

Таблиця А.2 – Виконавці проекту

Джерело: побудовано автором

Роль	Ім’я	Проектна роль
Розробник	Рикун В.А.	Виконує розробку основного функціоналу проекту.
Менеджер проекту	Боїко О.В.	Відповідає за виконання термінів, виконує збір та аналіз даних.
Тестувальник	Рикун В.А.	Відповідає за тестування функціоналу проекту.

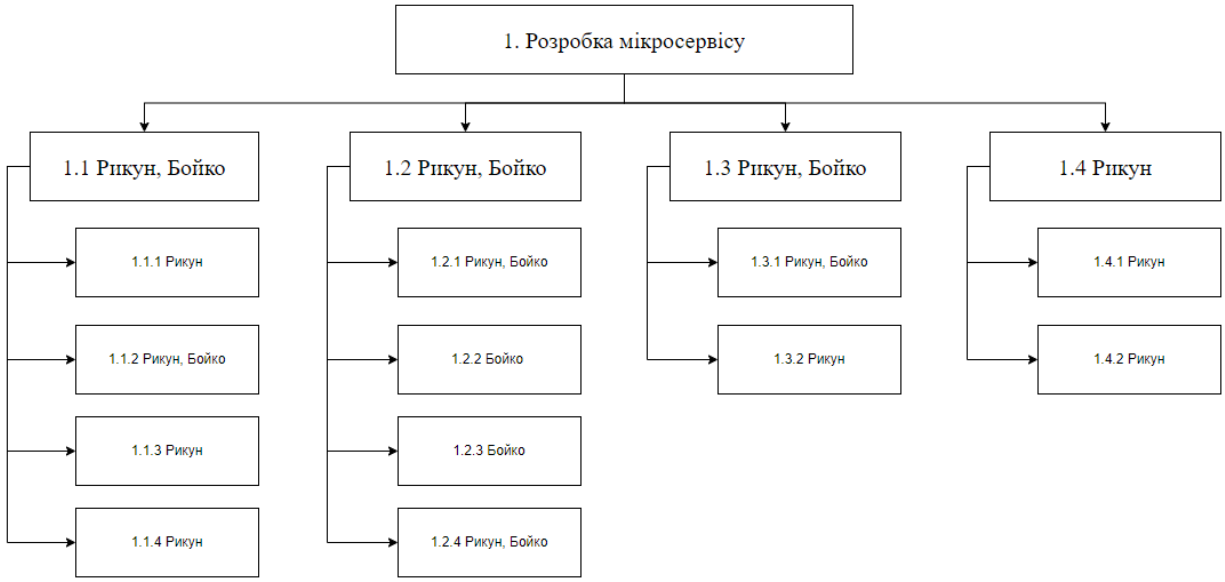


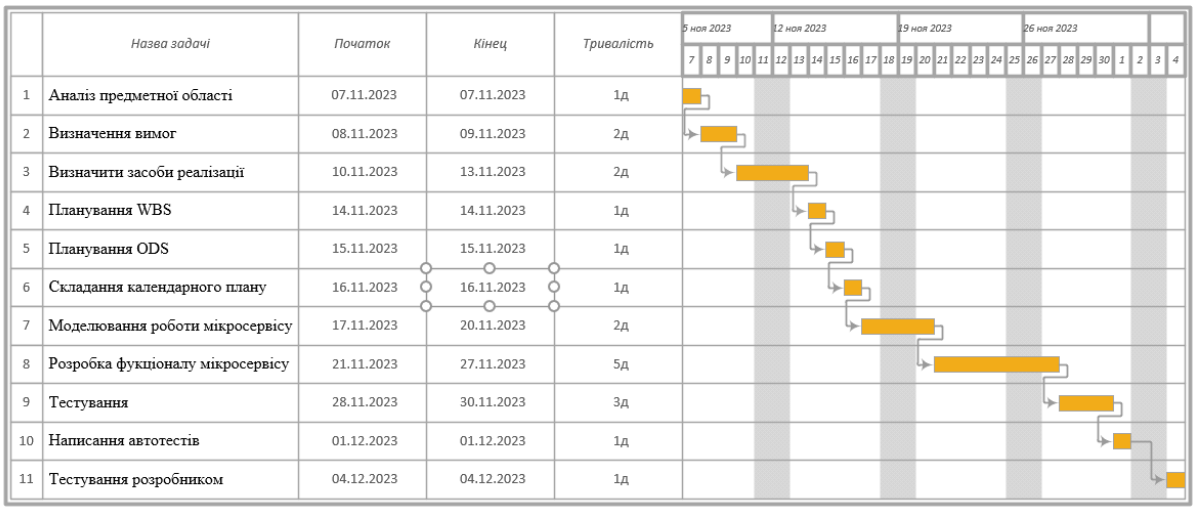
Рисунок А.2 – OBS-структура проекту

Джерело: побудовано автором

Діаграма Ганта. Створюємо календарний план для виконання дипломного проекту за допомогою діаграми Ганта. Цей графік є широко використовуваним інструментом управління проектами, а його формат гістограми сприяє відстеженню відсотка завершення кожного завдання. Для керівників проектів важливо розподілити завдання так, щоб мати впевненість у своєчасному завершенні проекту. Діаграми Ганта зорієнтовані на відсотковий прогрес виконання кожного завдання та особливо ефективні для

проектів із невеликою кількістю взаємопов'язаних завдань. З використанням програмного забезпечення MS Project була побудована діаграма Ганта, яка візуалізує тривалість кожного процесу, визначеного на етапі формування WBS. На рисунку А.3 представлено цю діаграму Ганта.

Діаграма Ганта



	Назва задачі	Початок	Кінець	Тривалість
1	Аналіз предметної області	07.11.2023	07.11.2023	1д
2	Визначення вимог	08.11.2023	09.11.2023	2д
3	Визначити засоби реалізації	10.11.2023	13.11.2023	2д
4	Планування WBS	14.11.2023	14.11.2023	1д
5	Планування ODS	15.11.2023	15.11.2023	1д
6	Складання календарного плану	16.11.2023	16.11.2023	1д
7	Моделювання роботи мікросервісу	17.11.2023	20.11.2023	2д
8	Розробка фукціоналу мікросервісу	21.11.2023	27.11.2023	5д
9	Тестування	28.11.2023	30.11.2023	3д
10	Написання автотестів	01.12.2023	01.12.2023	1д
11	Тестування розробником	04.12.2023	04.12.2023	1д

Рисунок А.3 – Діаграма Ганта проекту

Джерело: побудовано автором

Аналіз ризиків.

Ризик – це можлива подія, яка може мати як позитивний, так і негативний вплив на проект. Виникнення ризиків зумовлене невизначеністю, яка завжди присутня в кожному проекті. Ризики можна поділити на «відомі» – ті, які ідентифіковані, оцінені, і до яких можливе планування, та «невідомі» – ті, які не ідентифіковані і не піддаються передбаченню. Навіть якщо конкретні ризики і умови їх виникнення не визначені, більшість ризиків можна передбачити.

Ідентифікація ризиків – це процес визначення можливих ризиків, які можуть вплинути на проект, та фіксація їх ключових характеристик. Цей процес визначає, які саме ризики можуть виникнути та фіксує їх основні

атрибути. Ефективність ідентифікації ризиків залежить від її регулярності протягом всього життєвого циклу проекту.

Участь в ідентифікації ризиків повинна бути максимальною та включати різноманітних учасників, таких як керівники проекту, користувачі та незалежні фахівці.

Класифікація ризиків є наступна:

1. За ймовірністю виникнення:

– слабо ймовірнісні;
– мало ймовірнісні;
– імовірні;
– досить імовірні;
– майже імовірні.

2. За величиною втрат:

– мінімальна;
– низька;
– середня;
– висока;
– максимальна.

На основі цих даних була проведена класифікація ризиків для даного проекту, що наведена в таблиці А.3.

Таблиця А.3 – Класифікація ризиків

Джерело: побудовано автором

№	Назва ризику	Ймовірність	Величина втрат
1	Недотримання календарного плану	1	3
2	Некоректна робота програмного забезпечення	4	4
3	Некоректна робота апаратного забезпечення	4	4
4	Хвороба розробника	2	2
5	Некоректне тестування	2	1

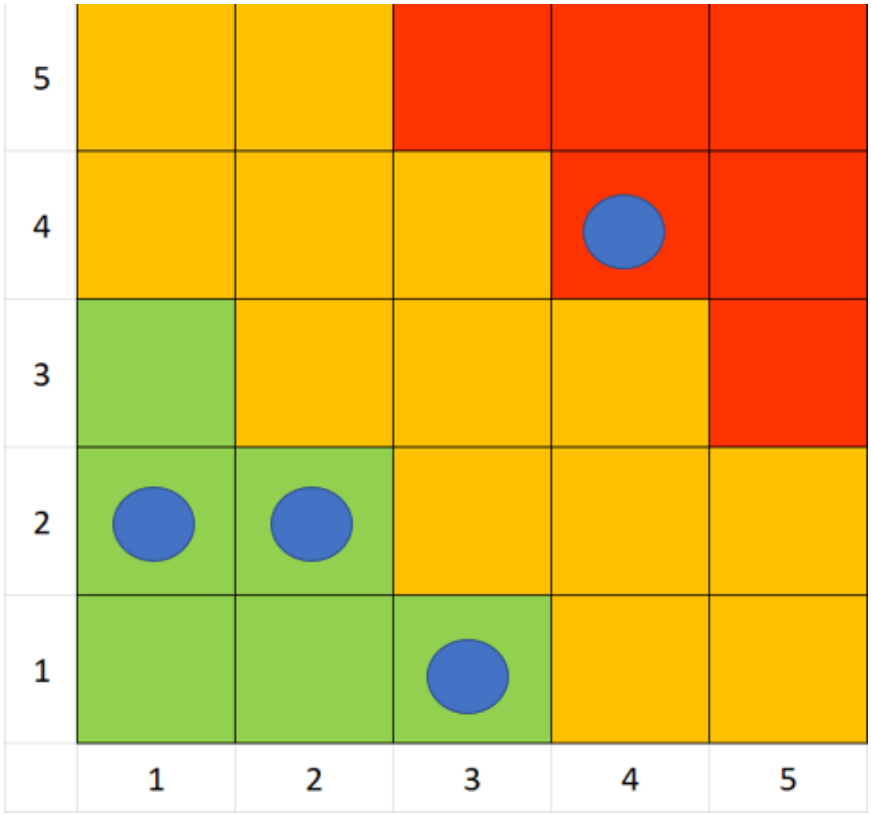


Рисунок А.4 – Матриця ризиків

Джерело: побудовано автором

Далі визначаємо рівні ризиків та ступінь їх дії.

Рівні можуть бути такі:

– допустимі 1<R<4;
– оправдані 5<R<10;
– недопустимі 13<R<25.

Ступінь дії ризиків є наступним:

– ті, що можна проігнорувати 1<R<4;
– незначні 5<R<8;
– помірні 9<R<10;
– істотні 11<R<16;
– критичні 17<R<25.

На основі цих даних була виконана оцінка ступенів та рівнів для кожного ризику в проекті. Результати роботи представлені в таблиці А.4.

Таблиця А.4 – Визначення ступенів та рівнів ризиків Таблиця А.4 – Визначення ступенів та рівнів ризиків

Джерело: побудовано автором

№№	Назва ризику	Ймовірність ризику	Величина втрат	Рівень ризику	Ступінь дії
11	Недотримання календарного плану	1	3	Допустимий	Проігнорувати
22	Некоректна робота програмного забезпечення	4	4	Недопустимий	Істотний
33	Некоректна робота апаратного забезпечення	4	4	Недопустимий	Істотний
44	Хвороба розробника	2	2	Допустимий	Проігнорувати
55	Некоректне тестування	2	1	Допустимий	Проігнорувати

Після проведення аналізу та передбачення можливих ризиків та їх потенційного впливу на результати проекту, були розроблені стратегії для уникнення або зменшення їх впливу, а також визначено відповідні заходи реагування на кожен конкретний ризик. Результати даного етапу представлені в таблиці А.5.

Таблиця А.5 – Варіанти запобігання та реакції на виявлені ризики

Джерело: побудовано автором

Ризики проекту	План запобігання ризику	План реакції на ризик
1	2	3
Недотримання календарного плану	Розробка плану втілення проекту, що базується на уважному аналізі всіх необхідних завдань. Узгодження зазначених термінів із замовником. Колективна співпраця над графіком виконання робіт з можливістю внесення коригувань перед його затвердженням усіма учасниками команди.	1. Розгляд альтернативних можливостей внесення змін у графік втілення проекту під час обговорення з керівником та замовником.. 2. Досягнення угоди щодо умов зміни термінів разом із замовником. У випадку, якщо така зміна є неприйнятною, провести переорганізацію роботи так, щоб результатом було дотримання встановлених термінів.
Некоректна робота програмного забезпечення	1. Впровадження ліцензійного програмного забезпечення, отриманого від надійних постачальників, перед розпочатком виконання завдань. 2. Гарантувати встановлення та належне функціонування антивірусного програмного забезпечення.	Виконання повторного запуску або перевстановлення програми, що призвела до неполадок.
Некоректна робота апаратного забезпечення	Періодично, кожні 4-6 місяців, проводити оцінку функціональності апаратної частини для підтримки її ефективності.	Здійснювати ремонт апаратного забезпечення негайно у випадку, коли виконання завдань неможливе через обмежений час, та вирішувати питання тимчасової замін.

Продовження таблиці А.5.

1	2	3
Хвороба розробника	Здійснювати колективне виконання конкретної частини завдань у проекті, з метою забезпечення можливості взаємозаміни членів команди у разі потреби. При плануванні термінів робіт залишати додаткові дні, які можна використовувати у випадках такого роду ситуацій..	Передавати обов'язки виконавця іншому члену команди в разі, якщо це необхідно для виконання робіт в зазначені терміни.
Некоректне тестування	Здійснити пошук спеціаліста у сфері тестування з відповідними кваліфікаціями в даній галузі.	Взяти рішення щодо передачі проекту на додатковий етап тестування, залучивши кваліфікованого фахівця у цей процес.

ДОДАТОК Б

ЛІСТИНГ ПРОГРАМНОГО КОДУ

Створення додатку на FastAPI вміст файлу **app.py**

```
from fastapi import FastAPI
from starlette.middleware.cors import CORSMiddleware

from itenergy.controllers import expert
from itenergy.controllers import rule_base

app = FastAPI(title='IT Energy service', version='0.0.1')
app.include_router(expert.router)
app.include_router(rule_base.router)

app.add_middleware(
  CORSMiddleware,
  allow_origins=["*"],
  allow_methods=["*"],
  allow_headers=["*"],
)
```

Контролер експерта всі CRUD операції вміст файлу **expert.py**

```
from fastapi import APIRouter
from starlette import status
from starlette.responses import Response

from itenergy.controllers.models.incoming import EquipmentRequest
from itenergy.db.engine import engine
from itenergy.repositories import equipments
from itenergy.repositories.equipments import Equipment

router = APIRouter(
  prefix='/expert/equipment',
  tags=['Expert']
)


@router.post('/', status_code=status.HTTP_201_CREATED)
def post_equipment(request: EquipmentRequest) -> Equipment:
  print(request)
  with engine.begin() as conn:
    equipment = equipments.new(
      id=request.id,
      voltage_deviation=request.voltage_deviation,
      phase_voltage_ua=request.phase_voltage_ua,
      phase_voltage_ub=request.phase_voltage_ub,
      phase_voltage_uc=request.phase_voltage_uc,
      interphase_voltage_uab=request.interphase_voltage_uab,
      interphase_voltage_uac=request.interphase_voltage_uac,
      interphase_voltage_ubc=request.interphase_voltage_ubc,
      asymmetry_coefficient_k2u=request.asymmetry_coefficient_k2u,
      asymmetry_coefficient_k0u=request.asymmetry_coefficient_k0u,
      capacity_battery_pb=request.capacity_battery_pb,
      current_solar_power=request.current_solar_power,
      current_wind_power=request.current_wind_power,
```

```
        capacity=request.capacity,
        solar_battery_power=request.solar_battery_power,
        wind_power=request.wind_power,
        power_consumption=request.power_consumption,
        conn=conn)

  return equipment


@router.patch('/', status_code=status.HTTP_200_OK)
def patch_equipment(request: EquipmentRequest) -> Equipment:
  with engine.begin() as conn:
    equipment = equipments.new(
      id=request.id,
      voltage_deviation=request.voltage_deviation,
      phase_voltage_ua=request. phase_voltage_ua,
      phase_voltage_ub=request. phase_voltage_ub,
      phase_voltage_uc=request.phase_voltage_uc,
      interphase_voltage_uab=request.interphase_voltage_uab,
      interphase_voltage_uac=request.interphase_voltage_uac,
      interphase_voltage_ubc=request.interphase_voltage_ubc,
      asymmetry_coefficient_k2u=request.asymmetry_coefficient_k2u,
      asymmetry_coefficient_k0u=request.asymmetry_coefficient_k0u,
      capacity_battery_pb=request.capacity_battery_pb,
      current_solar_power=request.current_solar_power,
      current_wind_power=request.current_wind_power,
      capacity=request.capacity,
      solar_battery_power=request.solar_battery_power,
      wind_power=request.wind_power,
      power_consumption=request.power_consumption,
      conn=conn)

  return equipment


@router.get('/', response_model=list[Equipment], status_code=status.HTTP_200_OK)
def get_all_equipments() -> list[Equipment]:
  with engine.begin() as conn:
    equipment = equipments.get_equipments(conn=conn)

  return equipment


@router.get('/{id}', response_model=Equipment, status_code=status.HTTP_200_OK)
def get_equipment(id: int) -> Equipment:
  with engine.begin() as conn:
    equipment = equipments.get_equipment(id=id, conn=conn)

  return equipment


@router.delete('/{id}', status_code=status.HTTP_204_NO_CONTENT)
def delete_equipment(id: int) -> Response:
  with engine.begin() as conn:
    equipments.delete(id=id, conn=conn)

  return Response(status_code=status.HTTP_204_NO_CONTENT)
```

Контролер бази правил всі CRUD операції вміст файлу **rule_base.py**

```
import json

from fastapi import APIRouter
from starlette import status
from starlette.responses import Response

from itenergy.controllers.models.rule_base import RuleBaseRequest
from itenergy.db.engine import engine
from itenergy.repositories import rule_base
from itenergy.repositories.rule_base import RuleBase

router = APIRouter(
  prefix='/rule_base',
  tags=['Rule base']
)


@router.post('/', status_code=status.HTTP_201_CREATED)
def post_rule_base(request: RuleBaseRequest) -> RuleBase:
  with engine.begin() as conn:
    rulebase = rule_base.new(
      id=request.id,
      input=request.input,
      switch=request.switch,
      conn=conn)

  return rulebase


@router.get('/{id}', response_model=RuleBase, status_code=status.HTTP_200_OK)
def get_rule_base(id: int) -> RuleBase:
  with engine.begin() as conn:
    rulebase = rule_base.get_rule_base(id=id, conn=conn)

  return rulebase


@router.delete('/{id}', status_code=status.HTTP_204_NO_CONTENT)
def delete_rule_base(id: int) -> Response:
  with engine.begin() as conn:
    rule_base.delete(id=id, conn=conn)

  return Response(status_code=status.HTTP_204_NO_CONTENT)
```

Валідація api request вміст файлу **incoming.py**

```
from pydantic import BaseModel


class EquipmentRequest(BaseModel):
  id: int
  voltage_deviation: float | None = None
  phase_voltage_ua: float | None = None
  phase_voltage_ub: float | None = None
  phase_voltage_uc: float | None = None
```

```
    interphase_voltage_uab: float | None = None
    interphase_voltage_uac: float | None = None
    interphase_voltage_ubc: float | None = None
    asymmetry_coefficient_k2u: float | None = None
    asymmetry_coefficient_k0u: float | None = None
    capacity_battery_pb: float | None = None
    current_solar_power: float | None = None
    current_wind_power: float | None = None
    capacity: float | None = None
    solar_battery_power: float | None = None
    wind_power: float | None = None
    power_consumption: float | None = None


class IndicatorRequest(BaseModel):
    id: int
    voltage_deviation: float | None = None
    phase_voltage_ua: float | None = None
    phase_voltage_ub: float | None = None
    phase_voltage_uc: float | None = None
    interphase_voltage_uab: float | None = None
    interphase_voltage_uac: float | None = None
    interphase_voltage_ubc: float | None = None
    asymmetry_coefficient_k2u: float | None = None
    asymmetry_coefficient_k0u: float | None = None
    capacity_battery_pb: float | None = None
    current_solar_power: float | None = None
    current_wind_power: float | None = None
    capacity: float | None = None
    solar_battery_power: float | None = None
    wind_power: float | None = None
    power_consumption: float | None = None


class RuleBaseRequest(BaseModel):
    id: int
    input: Any
    switch: Any
```

Створення з’єднаня з базою даних вміст файлу **engine.py**

```
import os

import sqlalchemy as sa


def get_db_url() -> str:
    return 'postgresql://%s:%s@%s:%s/%s' % (
        os.getenv('POSTGRES_USER', 'itenergy'),
        os.getenv('POSTGRES_PASSWORD', 'password'),
        os.getenv('PGHOST', 'itenergy-service-postgres'),
        os.getenv('PGPORT', 5432),
        os.getenv('POSTGRES_DB', 'itenergy'),
    )


engine = sa.create_engine(get_db_url(), pool_size=500, max_overflow=0, pool_pre_ping=True)

metadata = sa.MetaData()
```

Опис схеми бази даних вміст файлу **db.py**

```
from sqlalchemy import FLOAT, Column, Integer, Table

from itenergy.db.engine import metadata

equipment_data = Table(
  'equipment',
  metadata,
  Column('id', Integer, primary_key=True, index=True, unique=True),
  Column('voltage_deviation', FLOAT, nullable=True),
  Column('phase_voltage_ua', FLOAT, nullable=True),
  Column('phase_voltage_ub', FLOAT, nullable=True),
  Column('phase_voltage_uc', FLOAT, nullable=True),
  Column('interphase_voltage_uab', FLOAT, nullable=True),
  Column('interphase_voltage_uac', FLOAT, nullable=True),
  Column('interphase_voltage_ubc', FLOAT, nullable=True),
  Column('asymmetry_coefficient_k2u', FLOAT, nullable=True),
  Column('asymmetry_coefficient_k0u', FLOAT, nullable=True),
  Column('capacity_battery_pb', FLOAT, nullable=True),
  Column('current_solar_power', FLOAT, nullable=True),
  Column('current_wind_power', FLOAT, nullable=True),
  Column('capacity', FLOAT, nullable=True),
  Column('solar_battery_power', FLOAT, nullable=True),
  Column('wind_power', FLOAT, nullable=True),
  Column('power_consumption', FLOAT, nullable=True)
)

indicator_data = Table(
  'indicator',
  metadata,
  Column('id', Integer, primary_key=True, index=True, unique=True),
  Column('voltage_deviation', Integer, nullable=False),
  Column('phase_voltage_ua', Integer, nullable=False),
  Column('phase_voltage_ub', Integer, nullable=False),
  Column('phase_voltage_uc', Integer, nullable=False),
  Column('interphase_voltage_uab', Integer, nullable=False),
  Column('interphase_voltage_uac', Integer, nullable=False),
  Column('interphase_voltage_ubc', Integer, nullable=False),
  Column('asymmetry_coefficient_k2u', Integer, nullable=False),
  Column('asymmetry_coefficient_k0u', Integer, nullable=False),
  Column('capacity_battery_pb', Integer, nullable=False),
  Column('current_solar_power', Integer, nullable=False),
  Column('current_wind_power', Integer, nullable=False),
  Column('capacity', Integer, nullable=False),
  Column('solar_battery_power', Integer, nullable=False),
  Column('wind_power', Integer, nullable=False),
  Column('power_consumption', Integer, nullable=False)
)
```

Репозиторій для роботи з таблицею `equipments` вміст файлу **equipments.py**

```
from dataclasses import dataclass

from sqlalchemy.dialects.postgresql import insert
from sqlalchemy.engine import Connection

from itenergy.db.schema import equipment_data
```

```
@dataclass
class Equipment:
    id: int
    voltage_deviation: float | None = None
    phase_voltage_ua: float | None = None
    phase_voltage_ub: float | None = None
    phase_voltage_uc: float | None = None
    interphase_voltage_uab: float | None = None
    interphase_voltage_uac: float | None = None
    interphase_voltage_ubc: float | None = None
    asymmetry_coefficient_k2u: float | None = None
    asymmetry_coefficient_k0u: float | None = None
    capacity_battery_pb: float | None = None
    current_solar_power: float | None = None
    current_wind_power: float | None = None
    capacity: float | None = None
    solar_battery_power: float | None = None
    wind_power: float | None = None
    power_consumption: float | None = None


def new(id: int,
        voltage_deviation: float | None,
        phase_voltage_ua: float | None,
        phase_voltage_ub: float | None,
        phase_voltage_uc: float | None,
        interphase_voltage_uab: float | None,
        interphase_voltage_uac: float | None,
        interphase_voltage_ubc: float | None,
        asymmetry_coefficient_k2u: float | None,
        asymmetry_coefficient_k0u: float | None,
        capacity_battery_pb: float | None,
        current_solar_power: float | None,
        current_wind_power: float | None,
        capacity: float | None,
        solar_battery_power: float | None,
        wind_power: float | None,
        power_consumption: float | None,
        conn: Connection
        ) -> Equipment:
    set_fields = dict(
        id=id,
        voltage_deviation=voltage_deviation,
        phase_voltage_ua=phase_voltage_ua,
        phase_voltage_ub=phase_voltage_ub,
        phase_voltage_uc=phase_voltage_uc,
        interphase_voltage_uab=interphase_voltage_uab,
        interphase_voltage_uac=interphase_voltage_uac,
        interphase_voltage_ubc=interphase_voltage_ubc,
        asymmetry_coefficient_k2u=asymmetry_coefficient_k2u,
        asymmetry_coefficient_k0u=asymmetry_coefficient_k0u,
        capacity_battery_pb=capacity_battery_pb,
        current_solar_power=current_solar_power,
        current_wind_power=current_wind_power,
        capacity=capacity,
        solar_battery_power=solar_battery_power,
        wind_power=wind_power,
        power_consumption=power_consumption)
```

```
    stmt = (insert(equipment_data).values(
        set_fields,
    ).on_conflict_do_update(constraint='equipment_pkey', set_=set_fields).returning(equipment_data))
    return Equipment(**conn.execute(stmt).mappings().one())


def get_equipments(conn: Connection) -> list[Equipment]:
    equipments = conn.execute(equipment_data.select()).mappings().fetchall()

    return [Equipment(**equipment) for equipment in equipments]


def get_equipment(id: int, conn: Connection) -> Equipment:
    equipment = conn.execute(equipment_data.select().where(
        equipment_data.c.id == id)).mappings().one()

    return Equipment(**equipment)


def delete(id: int, conn: Connection) -> None:
    conn.execute(equipment_data.delete().where(equipment_data.c.id == id))
```

Репозиторій для роботи з таблицею `indicator` вміст файлу **indicator.py**

```
from dataclasses import dataclass
from typing import Any

from sqlalchemy.dialects.postgresql import insert
from sqlalchemy.engine import Connection

from itenergy.db.schema import indicator_data


@dataclass
class RuleBase:
    id: int
    input: Any
    switch: Any


def new(id: int, input: Any, switch: Any, conn) -> RuleBase:
    set_fields = dict(id=id, input=input, switch=switch)
    stmt = (insert(indicator_data).values(
        set_fields,
    ).on_conflict_do_update(constraint='indicator_pkey', set_=set_fields).returning(indicator_data))
    return RuleBase(**conn.execute(stmt).mappings().one())


def get_rule_base(id: int, conn: Connection) -> RuleBase:
    indicator = conn.execute(indicator_data.select().where(
        indicator_data.c.id == id)).mappings().one()

    return RuleBase(**indicator)


def delete(id: int, conn: Connection) -> None:
    conn.execute(indicator_data.delete().where(indicator_data.c.id == id))
```

Налаштування pytest, створення тестового кліенту вміст файлу **conftest.py**

```
import pytest
from fastapi.testclient import TestClient

from itenergy.app import app


@pytest.fixture(scope='session')
def test_client() -> TestClient:
    return TestClient(app)
```

Тести для контролера **expert.py**

```
import random

from fastapi.testclient import TestClient


def test_equipment(test_client: TestClient) -> None:
    equipment_id = random.randint(0, 9)
    equipment_body = dict(
        id=equipment_id,
        voltage_deviation=random.uniform(1, 200),
        phase_voltage_ua=random.uniform(1, 200),
        phase_voltage_ub=random.uniform(1, 200),
        phase_voltage_uc=random.uniform(1, 200),
        interphase_voltage_uab=random.uniform(1, 200),
        interphase_voltage_uac=random.uniform(1, 200),
        interphase_voltage_ubc=random.uniform(1, 200),
        asymmetry_coefficient_k2u=random.uniform(1, 200),
        asymmetry_coefficient_k0u=random.uniform(1, 200),
        capacity_battery_pb=random.uniform(1, 200),
        current_solar_power=random.uniform(1, 200),
        current_wind_power=random.uniform(1, 200),
        capacity=random.uniform(1, 200),
        solar_battery_power=random.uniform(1, 200),
        wind_power=random.uniform(1, 200),
        power_consumption=random.uniform(1, 200))

    with test_client:
        # test POST
        create_equipment = test_client.post(
            url='/expert/equipment/',
            json=equipment_body)

        assert create_equipment.status_code == 201

        # test PATCH
        equipment_body['wind_power'] = 15.7
        create_equipment = test_client.patch(
            url='/expert/equipment/',
            json=equipment_body)

        assert create_equipment.status_code == 200

        # test GET List
```

```
    create_equipment = test_client.get(
        url='/expert/equipment/')

    assert create_equipment.status_code == 200

    # test GET equipment
    create_equipment = test_client.get(
        url=f'/expert/equipment/{equipment_id}')

    assert create_equipment.status_code == 200

    # test DELETE
    create_equipment = test_client.delete(url=f'/expert/equipment/{equipment_id}')

    assert create_equipment.status_code == 204
```

Тести для контролера **rule_base.py**

```
import random

from fastapi.testclient import TestClient


def test_rule_base_controller(test_client: TestClient) -> None:
    rule_base_id = random.randint(0, 9)
    input_json = {"input": [
        {
            "name": "Відхилення напруги δUy",
            "setsName": [
                "Low",
                "Normal",
                "High"
            ],
            "sets": [
                [
                    0,
                    0,
                    0.9,
                    0.95
                ],
                [
                    0.9,
                    0.95,
                    1.05,
                    1.1
                ],
                [
                    1.05,
                    1.1,
                    2,
                    2
                ]
            ]
        }]
    }
    switch_json = {
        "switch": [
            {
                "mkol_off": 16,
```

```
            "mkol_on": 26,
            "kol_off_diap": [
              2,
              2,
              2,
              2,
              2,
              2,
              2,
              2
            ]
          }
        ]
      }
      rule_base_body = dict(
        id=rule_base_id,
        input=input_json,
        switch=switch_json)

      with test_client:
        # test POST
        create_equipment = test_client.post(
          url='/rule_base/',
          json=rule_base_body)

        assert create_equipment.status_code == 201

        # test GET
        create_equipment = test_client.get(
          url=f'/rule_base/{rule_base_id}')

        assert create_equipment.status_code == 200

        # test DELETE
        create_equipment = test_client.delete(url=f'/rule_base/{rule_base_id}')

        assert create_equipment.status_code == 204
```

Налаштування модуля міграцій бази даних

```
import os
from logging.config import fileConfig

from alembic import context
from sqlalchemy import create_engine

from itenergy.db.engine import metadata
from itenergy.db.schema import equipment_data  # noqa: F401
from itenergy.db.schema import indicator_data  # noqa: F401

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
if config.config_file_name is not None:
    fileConfig(config.config_file_name)
```

```
# add your model's MetaData object here
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata
target_metadata = metadata

# other values from the config, defined by the needs of env.py,
# can be acquired:
# my_important_option = config.get_main_option("my_important_option")
# ... etc.


def get_url():
    return 'postgresql://%s:%s@%s:%s/%s' % (
        os.getenv('POSTGRES_USER', 'itenergy'),
        os.getenv('POSTGRES_PASSWORD', 'password'),
        os.getenv('PGHOST', 'itenergy-service-postgres'),
        os.getenv('PGPORT', '5432'),
        os.getenv('POSTGRES_DB', 'itenergy'),
    )


def run_migrations_offline() -> None:
    """Run migrations in 'offline' mode.

    This configures the context with just a URL
    and not an Engine, though an Engine is acceptable
    here as well.  By skipping the Engine creation
    we don't even need a DBAPI to be available.

    Calls to context.execute() here emit the given string to the
    script output.

    """

    url = get_url()
    context.configure(
        url=url,
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
        compare_type=True
    )

    with context.begin_transaction():
        context.run_migrations()


def run_migrations_online() -> None:
    """Run migrations in 'online' mode.

    In this scenario we need to create an Engine
    and associate a connection with the context.

    """
    connectable = create_engine(get_url())

    with connectable.connect() as connection:
        context.configure(
```

```
            connection=connection, target_metadata=target_metadata
        )

        with context.begin_transaction():
            context.run_migrations()


if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Файли міграцій після зміни структури бази даних

```
"""Add all fields to equipment table

Revision ID: 602147c8dff9
Revises: 0aac60b54a54
Create Date: 2023-12-11 21:31:33.748053

"""
import sqlalchemy as sa
from alembic import op

# revision identifiers, used by Alembic.
revision = '602147c8dff9'
down_revision = '0aac60b54a54'
branch_labels = None
depends_on = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column('equipment', sa.Column('voltage_deviation', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('phase_voltage_ua', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('phase_voltage_ub', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('phase_voltage_uc', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('interphase_voltage_uab', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('interphase_voltage_uac', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('interphase_voltage_ubc', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('asymmetry_coefficient_k2u', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('asymmetry_coefficient_k0u', sa.FLOAT(), nullable=True))
    op.add_column('equipment', sa.Column('capacity_battery_pb', sa.FLOAT(), nullable=True))
    op.alter_column('equipment', 'current_solar_power',
           existing_type=sa.DOUBLE_PRECISION(precision=53),
           nullable=True)
    op.alter_column('equipment', 'current_wind_power',
           existing_type=sa.DOUBLE_PRECISION(precision=53),
           nullable=True)
    op.alter_column('equipment', 'capacity',
           existing_type=sa.DOUBLE_PRECISION(precision=53),
           nullable=True)
    op.alter_column('equipment', 'solar_battery_power',
           existing_type=sa.DOUBLE_PRECISION(precision=53),
           nullable=True)
    op.alter_column('equipment', 'wind_power',
           existing_type=sa.DOUBLE_PRECISION(precision=53),
           nullable=True)
```

```
    op.alter_column('equipment', 'power_consumption',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=True)
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.alter_column('equipment', 'power_consumption',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=False)
    op.alter_column('equipment', 'wind_power',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=False)
    op.alter_column('equipment', 'solar_battery_power',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=False)
    op.alter_column('equipment', 'capacity',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=False)
    op.alter_column('equipment', 'current_wind_power',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=False)
    op.alter_column('equipment', 'current_solar_power',
                    existing_type=sa.DOUBLE_PRECISION(precision=53),
                    nullable=False)
    op.drop_column('equipment', 'capacity_battery_pb')
    op.drop_column('equipment', 'asymmetry_coefficient_k0u')
    op.drop_column('equipment', 'asymmetry_coefficient_k2u')
    op.drop_column('equipment', 'interphase_voltage_ubc')
    op.drop_column('equipment', 'interphase_voltage_uac')
    op.drop_column('equipment', 'interphase_voltage_uab')
    op.drop_column('equipment', 'phase_voltage_uc')
    op.drop_column('equipment', 'phase_voltage_ub')
    op.drop_column('equipment', 'phase_voltage_ua')
    op.drop_column('equipment', 'voltage_deviation')
    # ### end Alembic commands ###

"""Add indicator table

Revision ID: 78f004ff08b4
Revises: 602147c8dff9
Create Date: 2023-12-13 21:58:54.890420

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = '78f004ff08b4'
down_revision = '602147c8dff9'
branch_labels = None
depends_on = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column('indicator', sa.Column('input', sa.JSON(), nullable=False))
    op.add_column('indicator', sa.Column('switch', sa.JSON(), nullable=False))
```

```
    op.drop_column('indicator', 'current_solar_power')
    op.drop_column('indicator', 'power_consumption')
    op.drop_column('indicator', 'interphase_voltage_ubc')
    op.drop_column('indicator', 'asymmetry_coefficient_k2u')
    op.drop_column('indicator', 'phase_voltage_ua')
    op.drop_column('indicator', 'asymmetry_coefficient_k0u')
    op.drop_column('indicator', 'phase_voltage_uc')
    op.drop_column('indicator', 'solar_battery_power')
    op.drop_column('indicator', 'wind_power')
    op.drop_column('indicator', 'capacity_battery_pb')
    op.drop_column('indicator', 'capacity')
    op.drop_column('indicator', 'voltage_deviation')
    op.drop_column('indicator', 'current_wind_power')
    op.drop_column('indicator', 'phase_voltage_ub')
    op.drop_column('indicator', 'interphase_voltage_uac')
    op.drop_column('indicator', 'interphase_voltage_uab')
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column('indicator', sa.Column('interphase_voltage_uab', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('interphase_voltage_uac', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('phase_voltage_ub', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('current_wind_power', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('voltage_deviation', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('capacity', sa.INTEGER(), autoincrement=False, nullable=False))
    op.add_column('indicator', sa.Column('capacity_battery_pb', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('wind_power', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('solar_battery_power', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('phase_voltage_uc', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('asymmetry_coefficient_k0u', sa.INTEGER(),
autoincrement=False, nullable=False))
    op.add_column('indicator', sa.Column('phase_voltage_ua', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('asymmetry_coefficient_k2u', sa.INTEGER(),
autoincrement=False, nullable=False))
    op.add_column('indicator', sa.Column('interphase_voltage_ubc', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('power_consumption', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.add_column('indicator', sa.Column('current_solar_power', sa.INTEGER(), autoincrement=False,
nullable=False))
    op.drop_column('indicator', 'switch')
    op.drop_column('indicator', 'input')
    # ### end Alembic commands ###
```

Використані налаштування для модулів перевірки коду та тестування

```
[flake8]
max-line-length=100
ignore=W503
exclude=alembic/*,*__pychache__*,.venv

[tool:pytest]
addopts=-vv --cov
env=
  env=test
  PGHOST=localhost
  PGPORT=5433
[isort]
known_third_party=alembic,fastapi,sqlalchemy
line_length=100
multi_line_output=5
src_paths=itenergy
skip_glob=*__pychache__*,.venv

[coverage:run]
source=itenergy
omit=/itenergy/tests/*

[coverage:report]
fail_under=100
show_missing=True
skip_covered=False
exclude_lines=
  if __name__ == .__main__.:
  pragma: no cover

[mypy]
ignore_missing_imports = True
show_error_codes=True
allow_untyped_calls=True
strict=True
files=itenergy
allow_untyped_decorators=True
exclude=alembic
```

Вміст файлу **docker-compose.yml**

```
---
services:
 itenergy-service:
  build: .
  container_name: itenergy
  depends_on:
   - itenergy-service-postgres
  volumes:
   - ${PWD}:/app
  command: ["uvicorn", "itenergy.app:app", "--host", "0.0.0.0", "--port", "8000"]

 itenergy-service-postgres:
  container_name: itenergy-service-postgres
  image: postgres:14.2
  environment:
   PGHOST: itenergy-service-postgres
   PGPORT: 5432
   POSTGRES_DB: itenergy
```

```
    POSTGRES_USER: itenergy
    POSTGRES_PASSWORD: password
  ports:
    - 5433:5432
```

Вміст файлу **Dockerfile**

```
FROM python:3.11

WORKDIR /app
COPY ./requirements.txt /itenergy/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /itenergy/requirements.txt
COPY itenergy /itenergy/app
ENV PYTHONPATH=/app/
```