КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації інформаційних технологій

Кафедра інформаційних технологій

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

на тему:

Розробка програмного забезпечення для оптимізації енергоспоживання в мережах

Луценко Олексій Іванович

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т.А.

„___” _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «**Розробка програмного забезпечення для оптимізації енергоспоживання в мережах**»

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач
Луценко Олексій Іванович
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21
Керівник Гончаренко Т.А.
(прізвище та ініціали)
д.т.н. проф.
(вчене звання, науковий ступінь)
Рецензент к.т.н., доц. Доля О.В.
(Прізвище та ініціали)

Ідентичність підтверджую

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет:	Автоматизації інформаційних технологій
Випускова кафедра:	Інформаційних технологій
Освітній ступінь:	Бакалавр
Спеціальність:	Комп’ютерні науки
Освітня програма:	Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО

_____

„___”_____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

	Луценко Олексій Іванович

1. Тема роботи Розробка програмного забезпечення для оптимізації енергоспоживання в мережах

затверджена наказом ректора КНУБА № 235/23/25 від «14» лютого 2025 року

2. Керівник роботи	Гончаренко Т.А., д.т.н., проф.

3. Строк подання Здобувачем роботи до захисту_____

4. Зміст пояснювальної записки за розділами:

Р.1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Р.2 СПЕЦИФІКАЦІЯ ВИМОГ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

Р.3 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

Р.4 ТЕСТУВАННЯ І ОЦІНКА РЕЗУЛЬТАТІВ

5. Графічний матеріал за розділами:

Р.1 Розділ містить 2 рисунка.

Р.2 Розділ містить 1 рисунок.

Р.3 Розділ містить 4 рисунка.

Р.4 Розділ містить 7 рисунків.

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	31.01.2025
Розділ 2	23.02.2025
Розділ 3	09.03.2025
Розділ 4	14.05.2025
Остаточне оформлення роботи	20.05.2025
Направлення роботи для перевірки на плагіат	22.05.2025
Попередній захист роботи на випусковій кафедрі	22.05.2025
Направлення роботи на рецензування	23.05.2025

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Рябчун Ю.В., доц.каф.ІТ	31.01.2025	
Розділ 2	Рябчун Ю.В., доц.каф.ІТ	23.02.2025	
Розділ 3	Рябчун Ю.В., доц.каф.ІТ	09.03.2025	
Розділ 4	Рябчун Ю.В., доц.каф.ІТ	14.05.2025	

8. Дата видачі завдання _____

Зав. кафедри			Гончаренко Т.А.
	(підпис)		(прізвище та ініціали)
Керівники			Гончаренко Т.А.
	(підпис)		(прізвище та ініціали)
Здобувач			Луценко О.І.
	(підпис)		(прізвище та ініціали)

Зміст

ВСТУП

Враховуючи різноманітність сучасних режимів роботи електричних мереж, очевидно, що потрібні нові підходи до оцінки енергетичних процесів та втрат електроенергії в електричних мережах [1-4]. Оптимізація втрат електроенергії та їх оцінка сприяє більш економічній експлуатації мережі електропередачі. Знання того, де відбуваються втрати електроенергії, дозволяє вжити заходів для їх обмеження або зменшення, що призводить до підвищення ефективності системи. Загалом, втрати в енергосистемах можна розділити на технічні втрати, зумовлені внутрішніми факторами, та нетехнічні втрати, зумовлені зовнішніми факторами. Загалом, втрати електроенергії в енергосистемах призводять до збільшення експлуатаційних витрат на обладнання та відповідно впливають на ціни на електроенергію [2]. Збільшення втрат електроенергії при передачі та розподілі зумовлене низкою факторів. Мережа електропередачі в нашій країні характеризується високою розрахунковою густиною струму - близько 1 А/мм², тоді як, наприклад, у країнах з розвиненим сучасним енергетичним комплексом вона становить 0,4-0,6 А/мм². Також спостерігається високий ступінь неоднорідності, оскільки цей параметр практично не враховується в критеріях проектування. Враховуючи гострий дефіцит енергії в Україні в умовах воєнного часу, контроль рівня споживання та ефективне використання енергії є важливими економічними аспектами.

Питання зменшення втрат електроенергії є важливою частиною більш широкої проблеми скорочення енергоспоживання та ефективного використання енергоресурсів. Оптимізація балансу між споживанням та виробництвом електроенергії є важливою частиною більш широкої проблеми скорочення енергоспоживання та ефективного використання енергоресурсів. Електромережа зазнає значних втрат при передачі та розподілі електроенергії, які, з урахуванням технічних та нетехнічних втрат, становлять майже 40% [1-4]. Втрати електроенергії в енергосистемах зазвичай призводять до збільшення

операційних витрат на експлуатацію електроустановок і, відповідно, до зростання цін на електроенергію. Тому зменшення втрат електроенергії в електроенергетичній системі є важливим не лише з фінансово-економічної точки зору, але й з точки зору соціально-економічних вигід для комунальних підприємств, споживачів та країни в цілому.

Питання управління енергією набуває все більшого значення в промисловості на тлі зростання та нестабільності цін на енергоносії, а також все більш мінливого забезпечення енергією. Підприємства конкурують на глобальному ринку та змушені постійно підвищувати свою продуктивність. Коливання та підвищення цін на енергоносії стають фактором ризику витрат і, отже, ставлять енергетичну продуктивність в центр уваги багатьох компаній.

Вирішальним для успішного та сталого управління енергією є своєчасне виявлення майбутніх викликів та найкраще використання синергетичних ефектів, пов'язаних з технологіями та ринком. Енергія тут розглядається не як окремий фактор, а скоріше як частина складної системи виробничих факторів, яку необхідно оптимізувати.

Зі зростанням мінливості фактора енергії для промислового виробництва, рішення для інтелектуальних мереж також стають актуальними у виробництві та відкривають область промислової розумної мережі (на англійській мові - Industrial Smart Grid (ISG)). ISG відкривають інноваційні можливості для створення рішень у сферах безпеки постачання, енергоефективності та енергетичної інтеграції. У цьому контексті ISG можуть стати важливим ключем до вирішення майбутніх завдань і тенденцій у промисловому енергопостачанні.

Зі зростанням значення енергетичної продуктивності в промисловості також зростає кількість програмних рішень для управління енергією та різноманітних підходів до системних рішень. Знайти правильне програмне рішення для компанії - це складний процес, який включає в себе як історію та сьогодення компанії, так і майбутній розвиток компанії на стратегічному та операційному рівнях. Цей процес дуже залежить від компанії та потребує

індивідуального підходу. Тому особливо важливо будувати основу для прийняття рішень на сучасному рівні розвитку технологій в управлінні енергією, а також на тенденціях розвитку в промисловості та енергетичній системі та перетворювати їх на конкурентні переваги.

Тому є актуальним виявлення тенденцій розвитку промислового виробництва та тенденцій розвитку в енергетичній системі та побудови на цій основі програмних рішень для управління енергією (EMS) та програмних рішень для виробничої системи виконання (MES) з енергетичною функціональністю.

Об'єкт розробки – регіональна електромережа.

Предмет розробки – програма оптимізації енергоспоживання в мережах.

Метою даною роботи є розробка програми оптимізації роботи регіональної електромережі.

Для досягнення зазначеної мети, поставлені такі завдання:

а) аналіз предметної області;

б) аналіз існуючих рішень в області оптимізації споживання електроенергії на регіональному рівні;

в) виділення окремого модуля для проектування першої версії програми;

г) здійснити проектування виділеного програмного модулю;

д) здійснити тестування розробленого модулю програми;

е) зробити оцінку можливості розширення роботи програми оптимізації споживання електроенергії в мережі в майбутньому.

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Системою електропостачання називають сукупність електроустановок, призначення яких забезпечувати споживачів електроенергією.

Безпосередньо електроустановки являють собою різноманітні машини, апарати та лінії, а також допоміжне обладнання та споруди, в яких усе це встановлено, службовці для виробництва, перетворення, передачі та розподілу електроенергії.

Система електропостачання є частиною електричного господарства організації або підприємства, при цьому виступає підсистемою по відношенню до більшої електроенергетичної системи.

Електроенергетична система, також названа просто електричною системою, є частиною енергосистеми і включає приймачі електроенергії.

Енергетичні системи включають до себе електростанції, електричні та теплові мережі та з'єднання між ними. Всі вони пов'язані між собою загальним режимом, просто безперервністю процесів виробництва, перетворення та розподілу електричної та теплової енергії. Електроенергія або електрична і теплова енергія виробляється на електростанціях. Електростанція може складатися з одного блоку для виробництва електроенергії або з декількох блоків.Електричні мережі є сукупністю електроустановок, призначення яких - передача та розподіл електричної енергії, що постачається електростанціями. Мережа включає підстанції, лінії електропередач, тоководи, приєднувальну апаратуру, а також засоби управління та захисту.

Підстанції служать прийому, перетворення і розподілу електроенергії. Лінія електропередач, у свою чергу, передає та розподіляє електроенергію, або просто передає її на відстань.

Будь-яке середньостатистичне підприємство завжди має власне електричне господарство, що включає насамперед сукупність електроустановок і різних виробів, що не належать до електричної мережі, проте забезпечують її нормальну роботу. Також до електричного господарства

входять приміщення, будівлі та споруди, що експлуатуються електротехнічним персоналом, людські, енергетичні, матеріальні ресурси та інформаційне забезпечення, покликане підтримувати повноцінну життєдіяльність господарства.

У складі будь-якого електричного господарства завжди є окремі електроприймачі або групи електроприймачів, розміщені на певній обмеженій території якогось об'єкта, та об'єднані єдиним технологічним процесом. Це може бути ціле підприємство чи окремий верстат, цех чи просто конвеєр. У будь-якому випадку таку одиницю чи групу прийнято назвати споживачем електричної енергії.

Функціонування системи електропостачання базується на режимі споживання електричної енергії, а також на технічному та ремонтному обслуговуванні. Справа в тому, що система електропостачання є безперервно працюючою, складною динамічною системою з різноманіттям внутрішніх і зовнішніх зв'язків.

Режим виробництва, передачі та розподілу в системі пов'язаний з режимом системи живлення, а режим і графік навантаження визначається споживачами. Електростанція впливає на систему електропостачання можливістю зміни зі свого боку обсягів потужності, що поставляється, рівня напруги, його частоти, величини струму короткого замикання, стійкості і т. д.

Ступінь стійкості електропостачання переважно визначається тим, наскільки регулярно та якісно виконуються технічні та ремонтні роботи в системі електропостачання. Дані роботи спрямовано підтримку постійної працездатності і справності як устаткування, і ліній електропередач. Сьогодні все це можна досягти завдяки наявності певних законів формування енергосистем та електричних господарств.

Принципово численні та різноманітні споживачі електроенергії в народному господарстві поділяються на чотири великі види (при цьому 10-12% всього енергоспоживання посідає освітлення):

- 55-65% – промислові підприємства;

- 25-35% - житлові та громадські будівлі, комунально-побутові організації та підприємства:

- 10-15% – сільськогосподарське виробництво;

- 2-4% – електрифікований транспорт.

Промислові споживачі електричної енергії на підприємствах можна класифікувати за такими п'ятьма ознаками:

1. За загальною номінальною потужністю встановлених електроприймачів:

- до 5 МВт - малі підприємства;

- від 5 до 75 МВт - середні підприємства;

- понад 75 МВт – великі підприємства.

2. По галузі промисловості, до якої це підприємство належить:

- металургія;

- машинобудування;

- нафтохімія;

Енергетична система (енергосистема) - сукупність електростанцій, електричних і теплових мереж, з'єднаних між собою і пов'язаних загальним режимом в безперервному процесі виробництва, перетворення і розподілу електричної і теплової енергії під загальним управлінням цим режимом;

Електроенергетична система (ЕПС) - це електрична частина енергосистеми та приймачі електроенергії, що живиться з неї, об'єднані загальним процесом виробництва, передачі, розподілу та споживання електроенергії.

Електрична мережа як складова електроенергетичної системи забезпечує можливість видачі потужності електростанцій, її передачі на відстань, перетворення параметрів електроенергії (напруги, струму) на підстанціях і її розподілу по певній території аж до прямих електроприймачів.

Топологічна структура окремих ланок цієї багатоступеневої мережі досить складна, вона налічує десятки, а іноді і сотні вузлів, відгалужень і замкнутих ланцюгів Під цим мається на увазі не тільки різноманіття

навантажень елементів мережі в добовому і річному контексті при нормальному функціонуванні системи, викликане природною зміною часу навантажень споживачів, але і велика кількість режимів, що виникають при винесенні різних елементів мережі на плановий ремонт і при їх аварійних відключеннях.

Всі електроприймачі, генератори, трансформатори та інші елементи електроенергетичних систем призначені для роботи в тривалому нормальному режимі при певній напрузі, при якому ці елементи мають найбільш відповідні техніко-економічні показники. Ці напруги називаються номінальними, і їх значення завжди встановлюються стандартом. В даний час для електричних мереж стандартизовані 4 напруги менше 1 кВ (40, 220, 380 і 660 В) і 12 напруг вище 1 кВ (3, 6, 10, 20, 35, 110, 150, 220, 330, 500, 750, 1150 кВ). Всі перераховані вище цифри відповідають лінійним (міжфазним) значенням напруг трифазної системи змінного струму.

Мережі з напругою до 1 кВ називаються низьковольтними (НН). Мережі з напругою вище 1 кВ, в свою чергу, поділяються на мережі середньої напруги (МВ), високої напруги, надвисоких (ВВ) і надвисоких (НН).

Як уже згадувалося, мережі сучасних енергосистем характеризуються дуже складною структурою і конфігурацією. У цих умовах неможливо класифікувати їх за якоюсь однією ознакою, яку можна було б вважати визначальною. Однак ряд особливостей в тій чи іншій мірі пов'язаний зі значенням номінальної напруги мережі U . До таких особливостей можна віднести покриття території, призначення мережі і, частково, характер її споживачів.

За розмірами території, що покривається мережею, можна виділити так звані місцеві ($U_{ном} \leq 35$ кВ), регіональні (110-220 кВ) і регіональні ($U_{ном} \geq 330$ кВ). Лінії електропередачі HVO, які є основою останньої категорії мереж, служать як для з'єднання окремих ділянок і відносно невеликих енергосистем в регіональних ОЕС, так і для з'єднання великих об'єднань між собою.

Перші виконують функції формування регіональних енергосистем (РЕЗ) шляхом об'єднання своїх електростанцій для паралельної роботи, а також об'єднання РЕЗ та ОЕС між собою. Крім того, вони передають електроенергію на системні підстанції, які виступають джерелами живлення для розподільчих мереж. Розподільчою лінією прийнято вважати лінію, яка живить ряд трансформаторних підстанцій або вводів до електроустановок споживачів. Такі лінії є основою розподільної мережі. Розподільні лінії, в принципі, можуть бути виділені в мережах з різною номінальною напругою. У зв'язку з цим не слід ототожнювати поняття локальних і розподільчих мереж, як це робилося раніше. У той же час мережі 110-220 і навіть 330 кВ поступово набувають характеру розподільчих мереж. Таким чином, оскільки новостворена мережа 750 кВ накладається на мережу 330 кВ на тих ділянках, де остання раніше виконувала функцію магістральної мережі, мережі 330 кВ поступово переходять у розряд розподільчих мереж. Надалі аналогічний процес буде спостерігатися і в тих частинах ОЕС України, де ролі основних з'єднань між ОЕС візьмуть на себе лінії 1150 кВ, в яких мережі 500 кВ зараз є основними.

Нарешті, згідно з таблицею 1.1, місцеві та розподільчі мережі можуть відрізнятися за характером підключених до них споживачів. У той же час мережі, які забезпечують електроенергією промислові підприємства, міста та сільськогосподарські райони і називаються відповідно промисловими, міськими та сільськими, мають певну специфіку. характеризуються значною протяжністю. Вони охоплюють ділянки з відносно невисокою щільністю навантаження, річна кількість годин максимального використання яких також відносно невелика. Навпаки, чисто промислові мережі, будучи відносно невеликими, живлять ділянки з високою щільністю навантаження, і, як правило, графіки навантаження промислових підприємств характеризуються високим ступенем заповнюваності. В якійсь мірі міські мережі займають в цьому плані проміжне положення. Поєднання комунальних і промислових споживачів в міських умовах викликає значну нерівномірність в графіках навантаження вузлів мережі міста. Така нерівномірність в ряді випадків (коли

основними джерелами живлення для міста є теплові електростанції, що працюють за тепловим графіком) викликає необхідність залучення додаткових маневрових потужностей, що дозволяють системі своєчасно і швидко реагувати на різкі спади і підйоми навантаження.

Крім ознак, побічно пов'язаних зі значенням номінальної напруги мережі, існують і інші. Наприклад, мережі класифікуються за типом струму, по конфігурації, по відношенню до приміщення і по конструкції.

Відповідно до типу струму розрізняють мережі змінного і постійного струму. Про першу групу йшлося вище. Крім того, слід зазначити, що в Росії мережі трифазного змінного струму напругою НО кВ і вище виконуються з твердим нульовим заземленням, а мережі більш низьких напруг - з ізольованою або заземленою нейтраллю через реактор придушення дуги.

Мережі постійного струму використовуються для підтримки деяких електротехнологічних процесів в промисловості, наприклад, в електролізних цехах алюмінієвих заводів. Постійний струм використовується для приводу ряду механізмів і часткової електрифікації транспорту.

З точки зору конфігурації розрізняють відкриті і закриті мережі. Відкриті мережі - це мережі, утворені радіальними або радіально-магістральними лініями, що подають електроенергію споживачам від одного джерела живлення, причому кожен споживач отримує живлення з одного напрямку. Найпростішою формою замкнутої мережі є одноконтурна (кільцева) мережа. Живильні мережі, як правило, складно замкнуті, тобто мають велику кількість контурів.

І, нарешті, за конструкцією мережі поділяються на внутрішні електропровідні (до 1 кВ), кабельні (до 500 кВ) та повітряні (до 750-1150 кВ) мережі. допустимо за умовами виробництва. Кабельні мережі 6-20 кВ в даний час є основою міських і промислових розподільчих мереж. Повітряні мережі характерні для електропостачання сільських споживачів, а також для регіональних і магістральних мереж.

Вже понад десять років у багатьох країнах розвивається концепція так званих розумних електричних мереж (Smart Grids) [1]. Ця концепція включає розгляд таких питань, як:

• застосування силових елементів ЕЕС, які мають у своєму складі комп'ютерні (цифрові) пристрої керування та використовуються для виробництва, накопичення, передачі, розподілу та споживання електроенергії;

• застосування сучасних інформаційно-комунікаційних технологій в управлінні ЕЕС;

• нові методи управління ЕЕС, які використовують у тому числі комплексні обчислювальні алгоритми та машинне навчання;

• застосування регулювальних пристроїв, які забезпечують активну поведінку споживачів з управління власним електроспоживанням;

• використання відновлюваної та малої розподіленої генерації.

Особливу актуальність дана концепція має у великих містах, коли до перерахованих вище властивостей електричних мереж додаються електромобілі. У майбутньому зростання частки електричних автомобілів тільки збільшуватиметься. Так, наприклад, під час конференції з клімату в Парижі у 2015 р. було прийнято спільну ініціативу “Паризька декларація з електромобільності та зміни клімату та заклик до дій”. Для досягнення цілей, заданих даною декларацією електромобілі повинні представляти 35% світових продажів автомобілів до 2030 [2].

Для якісної оцінки потенційного впливу електромобілів на електричні мережі було визначено частку участі електромобілів у загальному навантаженні електричних мереж на основі наступних відкритих статистичних даних: дані про кількість автомобільного транспорту душу населення [3]; статистичні дані та прогноз населення по країнах [4]; статистичні дані та прогноз споживання електроенергії по країнах [5].

Також було екстраполовано зростання обсягів продажу електромобілів [6] з урахуванням поточної тенденції та декларованої частки 35 % від загального обсягу продажів автомобілів.

В результаті на прикладі споживання електроенергії таких країн, як Україна, США, Франція та Німеччина (мал. 1), частка прогнозу споживання електромобілів виявляється суттєвою та сягає 25 % (рис. 2). Наведені дані не враховують можливе насичення ринку автомобілів, але для якісної оцінки такий підхід допустимий.

Електромобілі особливо добре висвічують проблеми, що виникають під час моделювання розумних електричних мереж. Тому опишемо структуру процесу оптимізації на прикладі міської електричної мережі з наявністю зарядних станцій для електромобілів та акумулюючих пристроїв, що працюють за технологією V2G. Коли електромобілі підключаються до електричної мережі неузгоджено і на повну потужність, необхідну для їх заряджання, то така нескоординована поведінка призводить до виникнення місцевих проблем енергосистемою, таких як додаткові втрати електроенергії, відхилення напруги, та тим самим до погіршення якості електроенергії та потенційного зниження стійкості ЕЕС.

1.2 Огляд методів оптимізації енергоспоживання в мережах

1.2.1. Основи оптимізації енергоспоживання

Цілями оптимізації з погляду мережевої компанії можуть бути: мінімізація втрат у мережі, забезпечення необхідних рівнів за напругою, оптимізація за вартістю генерації електроенергії. З погляду користувачів EV, важливим питанням є оптимізація заряджання EV за економічними критеріями: заряджання EV під час мінімальних тарифів, використання власних ресурсів генерації [5].

Така комплексна постановка проблеми спричиняє часткову суперечливість інтересів учасників процесу (мережевих компаній та власників EV). Власники EV бажали б мати можливість заряджати свій електромобіль у будь-який зручний час та у будь-якому зручному місці за найменшими

тарифами. Для мережевої компанії важливо мінімізувати свої витрати на передачу електроенергії. При цьому надійність електропостачання важлива для всіх учасників процесу.

Очевидно, що робота з єдиною моделлю, що є міською електричною мережею і кожен окремий електромобіль не можлива через її складність. Тому найбільш оптимальним варіантом вирішення комплексного завдання оптимізації міської мережі з електромобілями є дворівнева динамічна оптимізація. Верхній рівень становить собою стохастичну модель міської електричної мережі. Нижній рівень – це групові контролери заряду EV, що оптимізують процес заряду на рівні локальних мікрогрид чи зарядних агрегаторів.

Нижній рівень оптимізації. Цільова функція оптимізації на нижньому рівні враховує економічну ефективність заряду та видачі потужності у разі підключення EV за технологією V2G. Цільова функція оптимізації, що мінімізується, може визначатися наступним чином:

$$f\left(p_g(t)\right) = c_e\left(p_{ev}(t)\right) + c_i\left(p_{ev}(t)\right) + c(|p_{ev}(t) - p_d(t)|) \quad (1.1)$$

Перші два члени цієї суми визначають вартість одержуваної із зовнішньої мережі електроенергії (c_e) та вартість електроенергії, що генерується всередині мікромережі (c_i). Також у функції оптимізації є член, що визначає відхилення потужності EV від необхідної.

Графік зміни потужності $p_{ev}(t)$ може бути як потужністю навантаження, так і потужність генерації. Необхідний графік потужності EV ($p_d(t)$) обчислюється внаслідок оптимізації всієї міської електричної мережі та передається із системи управління верхнього рівня [5].

Оптимізація на нижньому рівні має враховувати стохастичний характер поведінки EV. Методами вирішення такого завдання, як правило, є варіанти динамічного програмування, такі як наближене динамічне програмування, що враховує стохастичний характер вихідних даних. Також застосовують методи адаптивного динамічного програмування, що використовують як адаптацію стратегії переходу системи від стану до стану, підхід навчання з підкріпленням.

Для зменшення розмірності розв'язуваної задачі застосовують такі методи як метод опорних векторів.

Крім класичного вирішення задачі оптимізації за допомогою динамічного програмування також можуть бути застосовані евристичні методи машинного навчання як глибокі Q-мережі, генетичні алгоритми, метод рою частинок та ін.

Оптимізація на верхньому рівні є багатоцільовою оптимізацією, яка може вирішуватися різними методами, такими як: оптимум за Парето [5], мінімізація відстані до точки утопії; методи об'єднання в одноцільову функцію. Серед методів, що об'єднують, можна виділити такі: метод виваженої суми, лексикографічний метод, зважений мінімаксний метод, експоненційний зважений критерій, метод виваженого твору, цільове програмування, фізичне програмування.

Найбільш простим та досить ефективним способом є метод оптимізації зваженої суми, коли окремі цільові функції підсумовуються однією за допомогою відповідних ваг.

$$f_c(x) = \sum_{i=1}^{n} w_i f_i(x) \mid g(x) \leq 0 \qquad (1.2)$$

де $f_i(x)$ - цільова функція за одним із аналізованих критеріїв; w_i - ваговий коефіцієнт, що відповідає підзадачі цільової функції; x - вектор управляючих параметрів; $g(x)$ - обмеження на самі керуючі параметри та функціональні обмеження, що визначають системні режимні обмеження. Також до цих обмежень входять обмеження обсяг управління, обчислювані лише на рівні агрегаторів, управляючих зарядкою EV.

Використання ваг дозволяє гармонізувати окремі складові цільової функції, масштабуючи їх таким чином, щоб це максимально задовольняло цілі мережної компанії. Раціоналізувати вибір терезів дозволяє метод аналізу ієрархій.

Перерахуємо складові цільової функції, що входять до завдання оптимізації верхнього рівня міської електричної мережі з електромобілями:

1. Мінімізація вартості генерації. У цій задачі вона включає не лише вартість виробництва енергії у міській енергосистемі, а й мінімізацію витрат на закупівлю електроенергії у генеруючих компаній.

$$F_g(p_g) = \sum_{i \in G}\left(c_{i2}p_{gi}^2 + c_{i1}p_{gi} + c_{i0}\right) \quad (1.3)$$

Керуючі параметри оптимізації: p_g – генерація активної потужності всередині мережі або ін'єкція потужності в мережу із зовнішньої мережі живлення. G - безліч генераторних вузлів.

Пріоритет обліку та масштабування параметрів цільової функції задаються константами: c_{i2}, c_{i1}, c_{i0} – константи, що масштабують облік вартості генерації.

2. Мінімізація втрат в електричній мережі:

$$F_{\Delta P}(V) = \sum_{\{i,j\} \in B} \Delta p_{ij}\left(V_i, V_j\right) \quad (1.4)$$

де Δp_{ij} - втрати у гілки i-j; V_i, V_j - напруги у вузлах суміжних гілок; B - множина індексів вузлів для кожної гілки у схемі заміщення [6].

3. Мінімізація відхилень за напругою:

$$F_{\Delta V}(V) = \sum_{i \in N} |V_i| \quad (1.5)$$

ΔV_i - відхилення напруги від допустимого; N - безліч вузлів, де контролюється напруга.

4. Зменшення впливу на довкілля.

Функція емісії забруднення може бути виражена як поліноміальна функція вихідної активної потужності генератора таким чином:

$$F_{Pol}\left(p_g\right) = \sum_{i \in G} e_{i0} + e_{i1}p_{gi} + e_{i2}{p_{gi}}^2 + \cdots + e_{ik}{p_{gi}}^k \quad (1.6)$$

де e_i - коефіцієнти забруднення; k - порядок.

Як обмеження на цільову функцію (2) виступають: обмеження на ін'єкцію реактивної потужності в балансуючих Q вузлах, обмеження на діапазон зміни керуючих параметрів, а також обмеження у вигляді балансу повної потужності в мережі, який, у свою чергу, обчислюється рішенням системи рівнянь балансу потужності

$$S_N = diag(V) \cdot Y \cdot V^t \quad (1.7)$$

де S_N - вектор ін'єкцій потужності; diag(V) – діагональна матриця складових комплексів вузлових напруг; Y – матриця комплексів провідностей.

Вірогідна постановка завдання. У ймовірності постановки цільова функція (1.5) може бути переписана як

$$p\big(f_c(x)\big) = \sum_{i=1}^{n} w_i p_i\big(f_i(x)\big) \qquad (1.8)$$

Замість детермінованих складових цільової функції $f_i(x)$ тут присутні розподілу ймовірності цих функцій $p\big(f_c(x)\big)$.

Як уже було показано, значення складових узагальненої цільової функції обчислюють на основі вектора стану системи, що є напругою в кожномувузлі електричної мережі, і навіть значень потужності генерації. Розподіл імовірності потужності генерації вузлів, що не є балансуючими, може бути задано апріорно в виді нормального розподілу. Таким чином, вся складність обчислення складових цільової функції зводиться до обчислення розподілу ймовірності напруг, які, у свою чергу, можуть бути отримані з рівнянь балансу потужності електричної мережі.

У літературі така постановка відома як імовірнісний розрахунок потокорозподілу (Probabilistic Powerflow) і вперше запропонована у [5]. Однак при цьому приймають припущення про лінійність системи, нехтують втратами в мережі, а також відсутня залежність між активною та реактивною потужністю. Такий підхід допустимо лише для наближеного розрахунку балансів потужності електричної мережі. Розвиток методу імовірнісного розрахунку потокорозподілу з тих пір тривало в багатьох роботах, таких як [5-7].

Суть імовірнісного розрахунку потокорозподілу полягає у поданні рівняння балансу електричної потужності у імовірнісній постановці в лінеаризованому вигляді:

$$p(S_N) = J^{-1} p(V) \qquad (1.9)$$

де J - матриця Якобі поточного стану системи.

У нових роботах за цим напрямом [8, 9] вирішується проблема обчислювальної складності операції згортки, яка може бути не застосовна для

великих систем. Дана проблема вирішується апроксимацією згортки напівінваріантами характеристичної функції довільної величини. Однак дані роботи також ґрунтуються на допустимості про лінійність розв'язуваної задачі. Таким чином, цей підхід може застосовуватися при плануванні режимів енергосистеми, коли модель може бути спрощена до лінійної, або апроксимація повинна виконуватися на кожній ітерації розв'язання нелінійної системи рівнянь методом Ньютон з використанням оновлених значень матриці J.

У загальному випадку, розподіл ймовірностей ін'єкцій потужності може бути різним для різних вузлів. Так, наприклад, модель вітрової електростанції використовує розподіл Вейбулла для визначення ймовірності швидкості вітру, на основі якої розраховують розподіл генерації вітрової електростанції. Розподіл потужності сонячної електростанції розраховують з використанням функції бета-розподілу, описує сонячну інсоляцію [10]. Модель навантаження зазвичай задають нормальним розподілом із середнім значенням та стандартним відхиленням. Що стосується зарядної навантаження EV, вона також може бути представлена нормальним розподілом для кожного моменту часу. При цьому протягом доби її математичне очікування та дисперсія змінюються.

Значно простіше обчислювати ймовірнісний потокорозподіл можна у разі, якщо всі ін'єкції до мережі представлені нормальними розподілами. Розподіл ймовірності цільової функції для кожного моменту часу в цьому випадку також представлятиме собою нормальний розподіл. Тут є обґрунтованим застосування методів інтервального аналізу [28] для оцінки розподілу ймовірності вектора станів системи *p(V)* і, як наслідок, розподілу ймовірності цільової функції.

При використанні інтервальної арифметики нормальний розподіл ймовірності ін'єкцій потужності представляють у вигляді інтервалу:

$$s_i = [s_{0i} - \sigma_{si}^2, s_{0i} + \sigma_{si}^2] \qquad (1.10)$$

Систему рівнянь балансу електричної потужності записують у вигляді інтервальної системи рівнянь:

$$s = v \cdot Y \cdot v^t \qquad (1.11)$$

де s – брус ін'єкцій потужності; v – потрібний брус змінних стану системи.

Розв'язання цієї задачі може бути виконане за допомогою зовнішнього оцінювання множини рішень інтервальним методом Ньютона [10]. За інтервальними значеннями змінних станів системи апроксимується математичне очікування та дисперсії змінних станів системи.

1.2.2. Методи зменшення енергоспоживання в мережах

Програмні методи відіграють ключову роль у зменшенні енергоспоживання в регіональних електромережах. Вони дозволяють ефективно керувати потоками енергії, оптимізувати навантаження та зменшити втрати. Ось деякі з основних програмних методів:

1. Системи SCADA (Supervisory Control and Data Acquisition). Це системи диспетчерського контролю та збору даних, які дозволяють в режимі реального часу відстежувати стан електромережі, контролювати навантаження, виявляти аварійні ситуації та оперативно реагувати на них. SCADA дозволяють оптимізувати режими роботи мережі, зменшити втрати електроенергії та підвищити надійність електропостачання.

2. Системи EMS (Energy Management Systems). Це системи управління енергоресурсами, які використовуються для планування, прогнозування та оптимізації роботи електромережі. EMS дозволяють прогнозувати споживання електроенергії, планувати режими роботи електростанцій, керувати навантаженням та оптимізувати витрати на виробництво та передачу електроенергії.

3. Системи DMS (Distribution Management Systems). Це системи управління розподільними мережами, які використовуються для контролю та

управління роботою розподільних підстанцій, ліній електропередачі та інших елементів розподільної мережі. DMS дозволяють оптимізувати режими роботи розподільної мережі, зменшити втрати електроенергії та підвищити якість електропостачання.

4. Системи MDMS (Meter Data Management Systems). Це системи управління даними обліку електроенергії, які використовуються для збору, обробки та аналізу даних з інтелектуальних лічильників. MDMS дозволяють отримувати детальну інформацію про споживання електроенергії, виявляти несанкціоноване споживання та оптимізувати режими роботи мережі.

5. Програмні комплекси для моделювання та аналізу електромереж. Ці програмні комплекси використовуються для моделювання та аналізу режимів роботи електромереж, оцінки їх надійності та оптимізації їх параметрів. Вони дозволяють прогнозувати наслідки різних сценаріїв розвитку електромережі та приймати обґрунтовані рішення щодо її розвитку.

6. Технології штучного інтелекту та машинного навчання. Ці технології використовуються для прогнозування споживання електроенергії, виявлення аномалій в роботі електромережі та оптимізації режимів її роботи. Вони дозволяють підвищити ефективність управління електромережею та зменшити втрати електроенергії.

7. Системи управління попитом (Demand Response). Ці системи дозволяють керувати споживанням електроенергії шляхом стимулювання споживачів до зменшення навантаження в періоди пікового споживання. Вони дозволяють зменшити навантаження на електромережу та уникнути перевантажень.

Використання цих програмних методів дозволяє підвищити ефективність управління регіональними електромережами, зменшити втрати електроенергії та оптимізувати витрати на електропостачання.

1.3 Огляд і аналіз існуючих програмних технологій оптимізації енергоспоживання в електромережах

Системи управління енергією (EMS) відіграють вирішальну роль у допомозі організаціям керувати, контролювати та оптимізувати використання енергії. Зі зростаючим акцентом на стійкості та економічній ефективності EMS стали основними інструментами в різних галузях промисловості, включаючи комерційний, промисловий і навіть житловий сектори. Надаючи дані в реальному часі, аналітику та можливості автоматизації, ці системи пропонують уявлення, які дозволяють підприємствам зменшити витрати енергії, заощадити витрати та підвищити ефективність роботи. Вибір правильного EMS передбачає розуміння конкретних потреб, оскільки пропоновані функції можуть дуже відрізнятися — від інструментів прогнозного обслуговування до моніторингу кількох об’єктів. У цьому блозі досліджуються найкращі доступні сьогодні системи керування енергією, зосереджуючись на їхніх унікальних функціях, перевагах і впливі, який вони можуть мати на енергетичну стратегію компанії [11].

Вибираючи систему енергоменеджменту (EMS), враховуйте ключові характеристики, які підвищують енергоефективність і ефективність роботи. Шукайте можливості моніторингу в режимі реального часу, які забезпечують миттєве уявлення про моделі споживання енергії та дозволяють миттєво коригувати. Інструменти аналізу даних необхідні для виявлення неефективності та прогнозування потреб в енергії, тоді як автоматизоване звітування спрощує дотримання вимог і відстеження ефективності. Можливості інтеграції з існуючими системами, такими як HVAC та керування освітленням, забезпечують безперебійну роботу та покращений збір даних. Зручні інтерфейси полегшують навігацію та дають можливість членам команди ефективно взаємодіяти з системою. Крім того, розгляньте можливість масштабування для майбутнього зростання та розвитку стратегій управління

енергією. Ці функції разом підтримують прийняття обґрунтованих рішень і сприяють довгостроковій економії енергії.

Розглянемо найкращі системи управління енергією для ефективного використання енергії

1. Aemaco Energy Management Solutions пропонує комплексну платформу, розроблену для оптимізації енергоспоживання для різноманітних будівель, включаючи комерційні та промислові об’єкти. Це рішення забезпечує моніторинг енергії в режимі реального часу, що дозволяє користувачам відстежувати моделі використання та вносити зміни, які зменшують відходи та знижують витрати. Платформа Aemaco є зручною для користувачів із настроюваною інформаційною панеллю, яка чітко представляє статистичні дані, що полегшує підприємствам виявлення неефективності та впровадження економічно ефективних заходів. Крім того, система надає детальну автоматизовану звітність для підтримки довгострокового енергетичного планування та аналізу, сприяючи сталому використанню енергії в усіх операційних процесах. Гнучкість і прагнення до ефективності Aemaco роблять його ідеальним вибором для організацій, які прагнуть досягти негайного та тривалого енергозбереження [11].

2. EcoStruxure™ Power від Schneider Electric - це інноваційне рішення для управління енергією з підтримкою Інтернету речей, розроблене для великих об’єктів, що дозволяє організаціям ефективно контролювати й оптимізувати споживання енергії. Платформа використовує розширену аналітику, яка дає змогу користувачам отримати уявлення про свої моделі споживання енергії та визначити сфери для покращення, що в кінцевому підсумку знижує експлуатаційні витрати. EcoStruxure™ Power об’єднує можливості прогнозування технічного обслуговування та прогнозування споживання енергії, що дозволяє організаціям передбачати потенційні проблеми до того, як вони переростуть у дорогі проблеми. Багатофункціональна інформаційна панель надає комплексне уявлення про споживання енергії кількома об’єктами, сприяючи стратегічному управлінню

енергією та прийняттю обґрунтованих рішень на кожному рівні. Ця система не тільки підвищує ефективність роботи, але й підтримує ініціативи сталого розвитку, зменшуючи загальний вуглецевий слід [11].

3. Siemens Desigo CC - це вдосконалена інтегрована система управління енергією, призначена для повної автоматизації будівель. Він особливо ефективний у складних інфраструктурах, де одночасне керування кількома системами має вирішальне значення. Desigo CC підтримує енергозбереження, підключаючи та керуючи різними функціями будівлі, такими як опалення, вентиляція, кондиціонування повітря, освітлення та системи безпеки. Завдяки складним мультидисциплінарним моніторинговим і аналітичним можливостям менеджери об'єктів можуть приймати обґрунтовані рішення, які сприяють значному підвищенню енергоефективності. Система також містить інструменти для контролю в реальному часі, що дозволяє користувачам регулювати споживання енергії на основі поточних потреб і рівня зайнятості. Ця можливість робить Siemens Desigo CC особливо корисним у великих комерційних установах, забезпечуючи точне управління енергією та економію коштів.

4. Energy Manager від Honeywell надає хмарну платформу, спеціально розроблену для моніторингу та аналізу споживання енергії під час роботи на кількох майданчиках. Його централізований інтерфейс дозволяє об'єктам відстежувати споживання енергії з єдиної точки, забезпечуючи візуалізацію даних і генерацію корисної інформації. Використовуючи інструменти, які підкреслюють неефективність і потенційні можливості енергозбереження, система Honeywell дає можливість організаціям приймати обґрунтовані рішення щодо їхньої енергетичної стратегії. Платформа підтримує порівняльний аналіз і відстеження продуктивності, сприяючи постійному вдосконаленню методів управління енергією. Завдяки зручному дизайну та надійним аналітичним функціям Honeywell Energy Manager дозволяє організаціям підвищити загальну енергоефективність і ефективно знизити експлуатаційні витрати.

5. Johnson Controls Metasys - це комплексна система управління енергією, розроблена для оптимізації експлуатації будівлі, одночасно підвищуючи енергоефективність за допомогою інтегрованого контролю. Система забезпечує уніфіковану платформу для керування різними джерелами енергії, такими як електроенергія, газ і вода, що дозволяє коригувати в режимі реального часу для оптимізації продуктивності. Metasys використовує надійну аналітику для виявлення неефективності використання енергії та сприяння коригувальним діям, що з часом призводить до значної економії коштів. Завдяки інтеграції з існуючими системами будівлі ця EMS покращує експлуатаційні можливості та підтримує проактивне управління енергією. Крім того, його масштабованість дозволяє організаціям адаптуватися до мінливих енергетичних потреб, що робить Metasys цінним рішенням як для існуючих будівель, так і для нових будівель [11].

6. ABB Ability Energy Manager - це система управління енергією на основі Інтернету речей, яка забезпечує моніторинг і аналіз споживання енергії різними пристроями та системами в реальному часі. Завдяки своїм розумінням, які базуються на даних, ця EMS дає змогу установам виявляти неефективність і ефективно оптимізувати їх використання енергії. Платформа АББ дуже адаптивна, що робить її придатною як для великомасштабних, так і для невеликих операцій, які прагнуть покращити свою практику управління енергією. Він пропонує комплексні інструменти для прогнозування використання енергії, відстеження продуктивності та звітності, які дозволяють організаціям приймати обґрунтовані рішення, що призводять до сталого енергозбереження. Використовуючи передову аналітику, ABB Ability™ Energy Manager сприяє постійному вдосконаленню енергоефективності та ефективності роботи.

1.4 Постановка задачі

1. Мета. Розробити програмний продукт (EMS), який інтегрується до комплексної платформи для оптимізації енергоспоживання в різноманітних будівлях (комерційних та промислових об'єктах). EMS повинен забезпечити ефективне управління енергоресурсами, зниження енерговитрат та підвищення енергоефективності об'єктів.

2. Завдання.

Збір даних в режимі реального часу від різних джерел: лічильники електроенергії, датчики температури, освітленості, вологості, обладнання HVAC (опалення, вентиляція та кондиціонування), системи освітлення, виробниче обладнання тощо.

Обробка та аналіз зібраних даних для виявлення закономірностей, аномалій та тенденцій споживання енергії.

Зберігання та архівування даних для подальшого аналізу та звітності.

Моніторинг та візуалізація:

Візуалізація даних про енергоспоживання в зручному та зрозумілому форматі (графіки, діаграми, дашборди).

Моніторинг стану обладнання та систем будівлі в режимі реального часу.

Відображення ключових показників енергоефективності (KPI).

Аналіз та прогнозування:

Аналіз історичних даних для виявлення можливостей оптимізації енергоспоживання.

Прогнозування попиту на енергію з використанням алгоритмів машинного навчання.

Виявлення потенційних проблем та аварійних ситуацій.

Звітність та аналітика:

Генерація звітів про енергоспоживання, KPI та результати оптимізації.

Аналіз ефективності впроваджених заходів з енергозбереження.

Надання рекомендацій щодо покращення енергоефективності.

Інтеграція та масштабованість:

Інтеграція з різними типами обладнання та системами.

Можливість масштабування системи для обслуговування великої кількості об'єктів.

Можливість гнучкого налаштування під потреби різних типів будівель.

Безпека:

Забезпечення захисту даних від несанкціонованого доступу.

Забезпечення кібербезпеки системи.

3. Цільова аудиторія:

Власники та керуючі комерційними та промисловими будівлями.

Енергоменеджери та технічні фахівці.

Компанії, що надають послуги з енергоаудиту та енергоефективності.

4. Очікувані результати:

Зниження енерговитрат будівель.

Підвищення енергоефективності об'єктів.

Покращення комфорту та умов праці.

Зменшення негативного впливу на довкілля.

Збільшення терміну служби обладнання.

Висновки до розділу 1

Розглянута система електропостачання як комплексна система, що включає електростанції, електричні мережі та споживачів, призначена для виробництва, передачі та розподілу електроенергії. Наведені різні параметри класифікації систем електропостачання. Додатково розглянуті так звані розумні електричні мережі (Smart Grids). Це є концепцією, що включає застосування цифрових технологій, інформаційно-комунікаційних систем, машинного навчання та відновлюваних джерел енергії. Актуальна для міських мереж з електромобілями.

Далі розглянуто методологію моделювання та оптимізації електричних мереж з стохастичними елементами. Стохастичні властивості сучасних електричних мереж та електричних мереж ближнього майбутнє пов'язане з розвитком Smart Grids. Електромобілі особливо хороші висвітлення проблем, які виникають при моделюванні розумних мереж. Отже, алгоритм оптимізації описується на прикладі міської електричної мережі з наявністю зарядних станцій для електричних транспортних засобів та пристрої зберігання, що використовують технологію V2G.

Серед різних програмних засобів для оптимізації енергоспоживання в мережах запропонована розробка системи EMS (Energy Management Systems). Це системи управління енергоресурсами, які використовуються для планування, прогнозування та оптимізації роботи електромережі.

Зроблена постановка задачі розробки програмного продукту (EMS), який інтегрується в комплексну платформу для оптимізації енергоспоживання в різноманітних будівлях (комерційних та промислових об'єктах).

Розділ 2. СПЕЦИФІКАЦІЯ ВИМОГ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Глосарій

EMS (Energy Management System) - система управління енергоспоживанням.

KPI (Key Performance Indicators) - ключові показники ефективності.

HVAC (Heating, Ventilation, and Air Conditioning) - система опалення, вентиляції та кондиціонування повітря.

Датчик - пристрій для вимірювання фізичних величин (температура, вологість, освітленість тощо).

Дашборд - інтерактивна панель, що відображає інформацію у вигляді графіків, діаграм, таблиць.

Аномалія - відхилення від нормального шаблону споживання енергії.

Архівування - збереження даних для довгострокового використання та аналізу.

Прогнозування - використання історичних даних та алгоритмів для передбачення майбутнього споживання енергії.

2.2 Концептуальна модель використання інформаційної системи

В результаті дослідження всіх аспектів роботи розроблюваної інформаційної системи розроблена наступна діаграма прецедентів, яка описує роботу програмного модулю (рис. 2.1). Діаграма прецедентів або варіантів використання (use case diagram) діаграма, де зображуються варіанти використання проектованої системи, укладені в кордон суб'єкта та зовнішні актори, а також певні відносини між акторами та варіантами використання. Діаграма варіантів використання призначена для досягнення наступних цілей:

• визначити загальні межі функціональності проектованої системи у контексті предметної області, що моделюється;

• специфікувати вимоги до функціональної поведінці проектованої системи у формі варіантів використання;

• розробити вихідну концептуальну модель системи для її подальшої деталізації у формі логічних та фізичних моделей;

• підготувати вихідну документацію для взаємодії розробників системи з її замовниками та користувачами.

Рис. 2.1. Діаграма прецедентів роботи програмного модулю ІС енергоменеджменту великого багатоквартирного будинку.

Вище розроблена діаграма прецедентів описує взаємодію між трьома акторами - Оператором, Адміністратором та Аналітиком - і програмним модулем інформаційної системи енергоменеджменту великого багатоквартирного будинку. Система призначена для управління енергоспоживанням, збору даних, їх обробки, аналізу та прогнозування, а також для генерації звітів і сповіщень.

Актори:

- Оператор відповідає за щоденне використання системи, зокрема за підключення до датчиків і отримання даних. Це може бути технічний персонал.

- Адміністратор є особою з вищими правами доступу, яка налаштовує систему, аналізує дані, генерує звіти та реагує на сповіщення. Наприклад, керівник або системний адміністратор.

- Аналітик є спеціалістом, який займається глибоким аналізом даних, інтерпретацією результатів, прогнозуванням і наданням рекомендацій для покращення енергоефективності. Аналітик працює з аналітичними інструментами системи, щоб виявляти тенденції та пропонувати рішення.

Система включає наступні прецеденти (Use Cases), які є відображенням функціональних вимог:

- Підключення до датчиків. Оператор підключає систему до датчиків (лічильників електроенергії, води, тепла тощо) для збору даних. Це базова функція для початку роботи системи.

- Отримання даних від лічильників. Оператор отримує дані про енергоспоживання від підключених лічильників. Залежить від прецеденту "Підключення до датчиків".

- Обробка, фільтрація та збереження даних. Система обробляє отримані дані, фільтрує помилкові значення та зберігає їх. Доступно для Оператора.

- Візуалізація та аналітика в інтерфейсі. Адміністратор і Аналітик можуть переглядати візуалізовані дані (графіки, діаграми) та проводити аналітику. Аналітик, зокрема, може зосереджуватися на детальному аналізі,

наприклад, порівнянні споживання за періодами або виявленні неефективних зон.

- Виявлення аномалій та сповіщення персоналу. Система автоматично виявляє аномалії (наприклад, різке зростання споживання) і сповіщає персонал. Адміністратор реагує на сповіщення, а Аналітик може аналізувати причини аномалій.

- Побудова прогнозів споживання. Адміністратор і Аналітик можуть прогнозувати майбутнє енергоспоживання на основі історичних даних. Аналітик, ймовірно, використовує цей прецедент для створення детальних моделей прогнозування, застосовуючи статистичні методи або машинне навчання.

- Генерація звітів. Адміністратор генерує звіти для звітності. Аналітик може брати участь у створенні спеціалізованих звітів, які включають глибокий аналіз і рекомендації (наприклад, звіт про енергоефективність із пропозиціями щодо зменшення витрат).

2.3 Специфікація функціональних та нефункціональних вимог

Інформаційна система енергоменеджменту призначена для автоматизації управління енергоспоживанням у великому багатоквартирному будинку. Система забезпечує збір даних із лічильників, їх обробку, аналіз, прогнозування, виявлення аномалій, генерацію звітів та візуалізацію даних. Користувачами системи є Оператор (технічний персонал), Адміністратор (керівник або системний адміністратор) та Аналітик (спеціаліст із аналізу даних).

Функціональні вимоги описують, що саме система повинна робити для кожного актора та прецеденту.

1. Підключення до датчиків.

FR-1.1 - Система повинна забезпечувати можливість підключення до датчиків (лічильників електроенергії, води, тепла, газу тощо) через стандартні протоколи зв’язку (наприклад, Modbus, MQTT).

FR-1.2 - Оператор повинен мати можливість додавати нові датчики через інтерфейс системи, вказуючи їх тип, ідентифікатор і параметри підключення.

FR-1.3 - Система повинна перевіряти коректність підключення датчиків і повідомляти Оператора про помилки (наприклад, "Датчик не відповідає").

2. Отримання даних від лічильників.

FR-2.1 - Система повинна автоматично отримувати дані від підключених лічильників із заданою періодичністю (наприклад, кожні 15 хвилин).

FR-2.2 - Оператор повинен мати можливість вручну ініціювати отримання даних із конкретного лічильника.

FR-2.3 - Система повинна зберігати отримані дані в базі даних із прив’язкою до часу та ідентифікатора лічильника.

3. Обробка, фільтрація та збереження даних.

FR-3.1 - Система повинна обробляти отримані дані, видаляючи аномальні значення (наприклад, від’ємні показники або значення, що перевищують допустимі межі).

FR-3.2 - Система повинна фільтрувати дані за заданими критеріями (наприклад, за типом ресурсу: електроенергія, вода, тепло).

FR-3.3 - Система повинна зберігати оброблені дані в базі даних із можливістю доступу до історичних даних за період не менше 5 років.

4. Візуалізація та аналітика в інтерфейсі.

FR-4.1 - Система повинна надавати Адміністратору та Аналітику інтерфейс для візуалізації даних у вигляді графіків, діаграм і таблиць (наприклад, споживання електроенергії за день, тиждень, місяць).

FR-4.2 - Система повинна дозволяти порівнювати споживання між різними періодами (наприклад, поточний місяць із попереднім).

FR-4.3 - Аналітик повинен мати можливість застосовувати аналітичні інструменти (наприклад, трендовий аналіз, кореляційний аналіз) для виявлення закономірностей у споживанні.

5. Виявлення аномалій та сповіщення персоналу.

FR-5.1 - Система повинна автоматично виявляти аномалії в даних (наприклад, зростання споживання на 50% порівняно з середнім значенням за місяць).

FR-5.2 - Система повинна надсилати сповіщення Адміністратору через email або вбудований інтерфейс у разі виявлення аномалії.

FR-5.3 - Аналітик повинен мати можливість переглядати деталі аномалії (наприклад, час, лічильник, значення) для подальшого аналізу.

6. Побудова прогнозів споживання.

FR-6.1 - Система повинна прогнозувати майбутнє енергоспоживання на основі історичних даних (наприклад, за допомогою методів часових рядів або машинного навчання).

FR-6.2 - Аналітик повинен мати можливість налаштовувати параметри прогнозування (наприклад, період прогнозу: тиждень, місяць).

FR-6.3: Адміністратор і Аналітик повинні мати доступ до результатів прогнозу у вигляді графіків і таблиць.

7. Генерація звітів.

FR-7.1 - Система повинна генерувати звіти про енергоспоживання за заданий період (день, тиждень, місяць, рік).

FR-7.2 - Адміністратор повинен мати можливість експортувати звіти у формати PDF і Excel.

FR-7.3 - Аналітик повинен мати можливість додавати до звітів аналітичні висновки та рекомендації (наприклад, "Зменшити споживання в нічний час шляхом відключення непотрібного освітлення").

8. Управління доступом.

FR-8.1 - Система повинна розмежовувати доступ для різних ролей: Оператор, Адміністратор, Аналітик.

FR-8.2 - Оператор має доступ лише до функцій збору та обробки даних.

FR-8.3 - Адміністратор має повний доступ до всіх функцій системи.

FR-8.4 - Аналітик має доступ до функцій аналізу, прогнозування та звітності.

Нефункціональні вимоги описують якісні характеристики системи, такі як продуктивність, безпека, масштабованість тощо.

1. Продуктивність

NFR-1.1 - Система повинна обробляти дані від 1000 лічильників одночасно без затримок (менше 1 секунди на обробку одного лічильника).

NFR-1.2 - Час генерації звіту не повинен перевищувати 10 секунд для даних за місяць.

NFR-1.3 - Система повинна забезпечувати оновлення візуалізації даних у реальному часі з затримкою не більше 5 секунд.

2. Надійність

NFR-2.1 - Система повинна мати рівень доступності (uptime) не нижче 99.9% на місяць.

NFR-2.2 - У разі збою система повинна автоматично відновлювати підключення до датчиків протягом 1 хвилини.

NFR-2.3 - Система повинна зберігати резервні копії даних із частотою не рідше одного разу на добу.

3. Масштабованість

NFR-3.1 - Система повинна підтримувати підключення до 5000 лічильників без значного зниження продуктивності.

NFR-3.2 - Система повинна дозволяти додавання нових типів датчиків без необхідності переписування коду (наприклад, через модульну архітектуру).

4. Безпека

NFR-4.1 - Усі дані, що передаються між датчиками та системою, повинні бути зашифровані (наприклад, за допомогою протоколу TLS).

NFR-4.2 - Доступ до системи повинен здійснюватися через автентифікацію (логін і пароль) із можливістю двофакторної автентифікації для Адміністратора.

NFR-4.3 - Система повинна вести журнал дій користувачів (хто, коли і що зробив) для аудиту.

5. Зручність використання (Usability)

NFR-5.1 - Інтерфейс системи повинен бути інтуїтивно зрозумілим, із можливістю навчання Оператора за 1 годину.

NFR-5.2 - Візуалізація даних повинна бути адаптивною для перегляду на різних пристроях (ПК, планшет, смартфон).

NFR-5.3 - Система повинна надавати підказки та документацію для всіх функцій.

6. Сумісність

NFR-6.1 - Система повинна бути сумісною з операційними системами Windows, Linux і macOS.

NFR-6.2 - Система повинна підтримувати інтеграцію з популярними платформами для аналітики (наприклад, Power BI) через API.

7. Обмеження

NFR-7.1 - Система повинна працювати при мінімальній швидкості інтернет-з’єднання 1 Мбіт/с.

NFR-7.2 - Обсяг бази даних для історичних даних не повинен перевищувати 1 ТБ протягом 5 років.

Ця специфікація охоплює основні функціональні вимоги, які відповідають прецедентам діаграми, а також нефункціональні вимоги, що забезпечують якісну роботу системи. Вона враховує потреби всіх акторів (Оператора, Адміністратора, Аналітика) і забезпечує основу для розробки, тестування та впровадження інформаційної системи енергоменеджменту.

2.4 Технічне завдання

Створюється технічне завдання на розробку програмного продукту EMS (Energy Management System)

1. Загальні відомості

Назва проекту: Розробка програмного продукту EMS (Energy Management System) для інтеграції до комплексної платформи оптимізації енергоспоживання.

Мета проекту:

Розробити програмний продукт (EMS), який інтегрується до комплексної платформи для оптимізації енергоспоживання в різноманітних будівлях (комерційних та промислових об’єктах). EMS повинен забезпечити ефективне управління енергоресурсами, зниження енерговитрат та підвищення енергоефективності об’єктів.

Цільова аудиторія:

- Власники та керуючі комерційними та промисловими будівлями.
- Енергоменеджери та технічні фахівці.
- Компанії, що надають послуги з енергоаудиту та енергоефективності.

Очікувані результати:

- Зниження енерговитрат будівель на 10–20%.
- Підвищення енергоефективності об’єктів.
- Покращення комфорту та умов праці в будівлях.
- Зменшення негативного впливу на довкілля шляхом зниження викидів CO_2.
- Збільшення терміну служби обладнання за рахунок своєчасного виявлення проблем.

2. Постановка задачі

Основні завдання розробки EMS:

- Збір даних у режимі реального часу:

    - Забезпечити збір даних із різних джерел: лічильники електроенергії, датчики температури, освітленості, вологості, обладнання HVAC (опалення, вентиляція, кондиціонування), системи освітлення, виробниче обладнання тощо.
- Обробка та аналіз даних:
    - Виконувати обробку зібраних даних для виявлення закономірностей, аномалій та тенденцій енергоспоживання.
    - Забезпечити зберігання та архівування даних для подальшого аналізу.
- Моніторинг та візуалізація:
    - Надавати зручну візуалізацію даних про енергоспоживання (графіки, діаграми, дашборди).
    - Забезпечити моніторинг стану обладнання та систем будівлі в режимі реального часу.
    - Відображати ключові показники енергоефективності (KPI).
- Аналіз та прогнозування:
    - Аналізувати історичні дані для виявлення можливостей оптимізації енергоспоживання.
    - Прогнозувати попит на енергію з використанням алгоритмів машинного навчання.
    - Виявляти потенційні проблеми та аварійні ситуації.
- Звітність та аналітика:
    - Генерувати звіти про енергоспоживання, KPI та результати оптимізації.
    - Аналізувати ефективність впроваджених заходів із енергозбереження.
    - Надавати рекомендації щодо покращення енергоефективності.
- Інтеграція та масштабованість:

- Забезпечити інтеграцію з різними типами обладнання та системами.
- Надавати можливість масштабування для обслуговування великої кількості об’єктів.
- Забезпечити гнучке налаштування під потреби різних типів будівель.

- Безпека:
  - Захищати дані від несанкціонованого доступу.
  - Забезпечити кібербезпеку системи.

3. Функціональні вимоги

3.1. Збір даних у режимі реального часу

FR-1.1: Система повинна забезпечувати підключення до різних джерел даних: лічильники електроенергії, датчики температури, освітленості, вологості, обладнання HVAC, системи освітлення, виробниче обладнання.

FR-1.2: Підтримка стандартних протоколів зв’язку (Modbus, MQTT, BACnet, OPC UA).

FR-1.3: Оператор повинен мати можливість додавати нові джерела даних через інтерфейс із налаштуванням параметрів (тип, ідентифікатор, частота збору).

FR-1.4: Система повинна отримувати дані в режимі реального часу з частотою не рідше 1 разу на 5 хвилин.

3.2. Обробка та аналіз даних

FR-2.1: Система повинна обробляти отримані дані, видаляючи аномальні значення (наприклад, від’ємні показники або значення, що перевищують допустимі межі).

FR-2.2: Система повинна виявляти закономірності (наприклад, пікове споживання в певний час доби) та тенденції (наприклад, зростання споживання взимку).

FR-2.3: Система повинна зберігати дані в базі даних із прив’язкою до часу, джерела та типу ресурсу.

FR-2.4: Система повинна архівувати дані для аналізу за період не менше 10 років.

3.3. Моніторинг та візуалізація

FR-3.1: Система повинна надавати дашборди для відображення даних про енергоспоживання (електроенергія, вода, тепло, газ) у вигляді графіків, діаграм і таблиць.

FR-3.2: Система повинна відображати стан обладнання в реальному часі (наприклад, "HVAC увімкнено", "Освітлення в секторі А вимкнено").

FR-3.3: Система повинна розраховувати та відображати KPI енергоефективності (наприклад, споживання енергії на квадратний метр, коефіцієнт використання обладнання).

FR-3.4: Користувач (Адміністратор, Аналітик) повинен мати можливість налаштовувати дашборди (вибір даних, період, тип візуалізації).

3.4. Аналіз та прогнозування

FR-4.1: Система повинна аналізувати історичні дані для виявлення можливостей оптимізації (наприклад, відключення освітлення в непрацюючих зонах).

FR-4.2: Система повинна прогнозувати попит на енергію з використанням алгоритмів машинного навчання (наприклад, ARIMA, LSTM).

FR-4.3: Система повинна виявляти потенційні аварійні ситуації (наприклад, перегрів HVAC) і повідомляти Адміністратора.

FR-4.4: Аналітик повинен мати можливість налаштовувати параметри прогнозування (період, тип моделі).

3.5. Звітність та аналітика

FR-5.1: Система повинна генерувати звіти про енергоспоживання, KPI та результати оптимізації за заданий період (день, тиждень, місяць, рік).

FR-5.2: Система повинна аналізувати ефективність впроваджених заходів (наприклад, зниження споживання після встановлення енергоефективного освітлення).

FR-5.3: Система повинна надавати рекомендації щодо покращення енергоефективності (наприклад, "Зменшити температуру в зоні B на 2°C у нічний час").

FR-5.4: Звіти повинні експортуватися у формати PDF, Excel і CSV.

3.6. Інтеграція та масштабованість

FR-6.1: Система повинна інтегруватися з різними типами обладнання та системами через API або стандартні протоколи.

FR-6.2: Система повинна підтримувати масштабування для роботи з 10–1000 об’єктів одночасно.

FR-6.3: Система повинна надавати гнучке налаштування (наприклад, додавання нових типів KPI, налаштування дашбордів під конкретний об’єкт).

3.7. Управління доступом

FR-7.1: Система повинна розмежовувати доступ для ролей: Оператор, Адміністратор, Аналітик.

FR-7.2: Оператор має доступ до збору та обробки даних.

FR-7.3: Адміністратор має повний доступ до всіх функцій.

FR-7.4: Аналітик має доступ до аналізу, прогнозування, звітності та рекомендацій.

4. Нефункціональні вимоги

4.1. Продуктивність

NFR-1.1: Система повинна обробляти дані від 5000 джерел одночасно з затримкою не більше 1 секунди на джерело.

NFR-1.2: Час генерації звіту не повинен перевищувати 15 секунд для даних за рік.

NFR-1.3: Оновлення дашбордів у реальному часі повинно відбуватися з затримкою не більше 3 секунд.

4.2. Надійність

NFR-2.1: Рівень доступності системи (uptime) — не нижче 99.95% на місяць.

NFR-2.2: У разі збою система повинна автоматично відновлювати підключення до джерел даних протягом 30 секунд.

NFR-2.3: Система повинна створювати резервні копії даних щоденно.

4.3. Масштабованість

NFR-3.1: Система повинна підтримувати підключення до 10 000 джерел даних без зниження продуктивності.

NFR-3.2: Система повинна бути модульною, щоб додавання нових типів джерел даних не вимагало переписування коду.

4.4. Безпека

NFR-4.1: Усі дані, що передаються між системою та джерелами, повинні бути зашифровані (протокол TLS 1.3).

NFR-4.2: Доступ до системи — через автентифікацію (логін, пароль) із підтримкою двофакторної автентифікації для Адміністратора.

NFR-4.3: Система повинна вести журнал дій користувачів для аудиту.

NFR-4.4: Система повинна відповідати стандартам кібербезпеки (наприклад, ISO 27001).

4.5. Зручність використання

NFR-5.1: Інтерфейс повинен бути інтуїтивно зрозумілим, із можливістю навчання користувача за 2 години.

NFR-5.2: Дашборди повинні бути адаптивними для перегляду на ПК, планшетах і смартфонах.

NFR-5.3: Система повинна надавати вбудовану документацію та підказки.

4.6. Сумісність

NFR-6.1: Система повинна працювати на ОС Windows, Linux, macOS.

NFR-6.2: Система повинна інтегруватися з платформами аналітики (наприклад, Power BI, Tableau) через API.

4.7. Обмеження

NFR-7.1: Система повинна працювати при швидкості інтернет-з’єднання від 1 Мбіт/с.

NFR-7.2: Обсяг бази даних для 10 років не повинен перевищувати 5 ТБ.

5. Технічні вимоги

5.1. Архітектура системи

Система повинна бути побудована за мікросервісною архітектурою для забезпечення масштабованості та гнучкості.

Основні компоненти:

- Модуль збору даних (Data Acquisition Module).
- Модуль обробки та аналізу даних (Data Processing & Analytics Module).
- Модуль візуалізації (Visualization Module).
- Модуль прогнозування (Forecasting Module).
- Модуль звітності (Reporting Module).
- Модуль інтеграції (Integration Module).
- Модуль безпеки (Security Module).

5.2. Технологічний стек.

Backend: Python (FastAPI) або Java (Spring Boot) для обробки даних і API.

Frontend: React.js або Angular для створення адаптивного інтерфейсу.

База даних: PostgreSQL для зберігання даних, InfluxDB для часових рядів.

Машинне навчання: TensorFlow для прогнозування.

Хмарна інфраструктура: AWS, Azure або Google Cloud для масштабування.

Протоколи зв’язку: MQTT.

5.3. Інтеграція

Система повинна надавати REST API та WebSocket для інтеграції з іншими платформами.

Підтримка експорту даних у формати JSON, CSV для інтеграції з аналітичними інструментами.

6. Етапи розробки

Аналіз вимог (2 тижні):

Збір і уточнення вимог.

Створення детальної специфікації.

Проектування (3 тижні):

Розробка архітектури системи.

Проектування бази даних і API.

Розробка (12 тижнів):

Розробка модулів збору даних, обробки, аналізу, візуалізації, прогнозування, звітності, інтеграції та безпеки.

Реалізація інтерфейсу користувача.

Тестування (4 тижні):

Модульне, інтеграційне та навантажувальне тестування.

Тестування безпеки (перевірка на вразливості).

Впровадження (2 тижні):

Розгортання системи на сервері замовника або в хмарі.

Навчання користувачів.

Підтримка (постійна):

Технічна підтримка та оновлення системи.

Загальний термін розробки: 23 тижні (≈5.5 місяців).

7. Критерії приймання:

- Система успішно збирає дані з усіх типів джерел (лічильники, датчики, HVAC, освітлення).

- Система коректно обробляє, аналізує та зберігає дані.

- Дашборди відображають дані в реальному часі з затримкою не більше 3 секунд.

- Прогнози енергоспоживання мають точність не нижче 85%.

- Звіти генеруються коректно та експортуються у всіх зазначених форматах.

- Система проходить тести на безпеку (відсутність критичних вразливостей).

- Користувачі (Оператор, Адміністратор, Аналітик) можуть виконувати свої завдання після 2-годинного навчання.

8. Документація

- Технічна документація включає опис архітектури, API, модулів, інструкції з розгортання.

- Посібник користувача включає інструкції для Оператора, Адміністратора, Аналітика.

9. Ризики та обмеження

- Ризики:
    - Затримки в інтеграції з обладнанням через несумісність протоколів.
    - Недостатня точність прогнозів через обмежену кількість історичних даних.
- Обмеження:
    - Система потребує стабільного інтернет-з’єднання (мін. 1 Мбіт/с).
    - Для прогнозування потрібен період накопичення даних (мін. 6 місяців).

Висновки до розділу 2

Створені специфікації функціональних та нефункціональних вимог для програмного продукту EMS (Energy Management System) є важливим документом, який деталізує, що саме система повинна робити (функціональні вимоги) і якими характеристиками вона повинна володіти (нефункціональні вимоги). Функціональні вимоги базуються на описаних у постановці задачі можливостях системи, таких як збір даних у реальному часі, обробка та аналіз даних, моніторинг, візуалізація, прогнозування, звітність, інтеграція та забезпечення безпеки. Нефункціональні вимоги враховують технічні аспекти (продуктивність, масштабованість, безпека) та якісні характеристики (зручність використання, сумісність), які забезпечують ефективну роботу системи в умовах комерційних і промислових об’єктів.

Створено технічне завдання (ТЗ) для розробки програмного продукту EMS є документом, який детально описує цілі, вимоги, технічні аспекти та етапи реалізації проєкту. Воно базується на попередньо розглянутих функціональних і нефункціональних вимогах, а також на постановці задачі. ТЗ враховує всі аспекти постановки задачі, включаючи інтеграцію з різними типами обладнання, масштабування для великої кількості об’єктів, прогнозування з використанням алгоритмів машинного навчання, захист даних і кібербезпеку, а також забезпечення зручності для цільової аудиторії - власників, енергоменеджерів і компаній з енергоаудиту.

Розділ 3. ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

3.1 Опис вихідних і вхідних даних

Для програмного продукту EMS (Energy Management System), розробленого для інтеграції до комплексної платформи оптимізації енергоспоживання в комерційних та промислових об’єктах, наводиться вхідних та вихідних даних, враховуючи функціональні вимоги, описані в технічному завданні та специфікаціях.

1. Вхідні дані

Вхідні дані - це інформація, яку система отримує від зовнішніх джерел, користувачів або інших систем для виконання своїх функцій. Вони включають дані від обладнання, введені користувачами параметри та налаштування.

1.1. Дані від джерел (обладнання та датчиків)

Типи джерел:

- лічильники електроенергії;
- датчики температури;
- датчики освітленості;
- датчики вологості;
- обладнання HVAC (опалення, вентиляція, кондиціонування);
- системи освітлення.

Формат даних:

- Числові значення:
    - спожита електроенергія в кВт·год;
    - температура в °C;
    - рівень освітленості в люксах;
    - вологість у %.
- Логічні значення:
    - статус обладнання: увімкнено/вимкнено).
- Часові мітки (timestamp) для кожного значення ("2025-04-14 10:00:00").

- Протокол передачі MQTT.

- Частота збору:

  - Не рідше 1 разу на 5 хвилин (в режимі реального часу).

1.2. Дані, введені користувачами.

Роль користувача:

- Оператор;
- Адміністратор;
- Аналітик.

Типи даних:

- Налаштування підключення до джерел даних (Оператор):

- Ідентифікатор джерела.

- Тип джерела (лічильник, датчик).

- Протокол зв’язку MQTT.

- Частота збору даних (наприклад, кожні 5 хвилин).

- Параметри фільтрації та обробки даних (Оператор, Адміністратор):

- Діапазон допустимих значень (наприклад, температура: 0–50°C).

- Період аналізу (наприклад, "з 2025-04-01 до 2025-04-14").

- Параметри прогнозування (Аналітик):

- Період прогнозу (наприклад, на тиждень вперед).

- Тип моделі (наприклад, ARIMA або LSTM).

- Налаштування звітів і дашбордів (Адміністратор, Аналітик):

- Період звіту (наприклад, "квітень 2025").

- Типи даних для відображення (наприклад, електроенергія, температура).

- Формат експорту (PDF, Excel, CSV).

1.3. Дані від зовнішніх систем (інтеграція)

Типи даних:

- Дані від платформ аналітики через API.

- Історичні дані з інших систем (наприклад, попередні записи енергоспоживання у форматі CSV).

- Формат: JSON, CSV.

2. Вихідні дані

Вихідні дані — це інформація, яку система генерує в результаті обробки вхідних даних. Вони включають результати аналізу, прогнози, звіти, сповіщення та візуалізацію.

2.1. Оброблені та збережені дані

Опис:

- Оброблені дані після фільтрації та видалення аномалій, збережені в базі даних.

Формат:

- Табличний формат із прив’язкою до часу, джерела та типу ресурсу.

2.2. Дані для моніторингу та візуалізації

Опис:

- Дані, відображені на дашбордах у реальному часі.

Формат:

- JSON для передачі до frontend.

- Графічне представлення: графіки, діаграми, таблиці.

Типи даних:

- Поточне енергоспоживання (електроенергія, вода, тепло).

- Стан обладнання (наприклад, HVAC увімкнено/вимкнено).

- KPI енергоефективності (наприклад, споживання на $м^2$).

2.3. Результати аналізу та прогнозування

Опис:

- Результати обробки даних, виявлення аномалій, прогнозування попиту.

Формат:

- JSON для передачі до інтерфейсу.

- Текстові описи (для рекомендацій).

Типи даних:

- Виявлені аномалії (наприклад, різке зростання споживання).

- Прогнози енергоспоживання (наприклад, на наступний тиждень).

3.2 Розробка об’єктної моделі

З врахуванням функціональних вимог, вхідних та вихідних даних, а також ролі користувачів (Оператор, Адміністратор, Аналітик) починається розробка об’єктної моделі інформаційної системи енергоменеджменту (EMS), яка призначена для інтеграції до комплексної платформи оптимізації енергоспоживання в комерційних та промислових об’єктах, необхідно Об’єктна модель або по іншому діаграма класів UML описує основні класи системи, їхні атрибути, методи та зв’язки між ними, що відображають структуру даних і логіку роботи системи.

Об’єктна модель будується на основі функціональних вимог (збір даних, обробка, аналіз, візуалізація, прогнозування, звітність, інтеграція, безпека), ролей користувачів (Оператор (збирає та обробляє дані), Адміністратор (управляє системою, переглядає звіти), Аналітик (аналізує дані, прогнозує, надає рекомендації)), вхідних та вихідних даних (дані від датчиків, параметри налаштувань, звіти, прогнози, сповіщення) та модульної структури, коли система поділена на модулі (збір даних, аналіз, візуалізація тощо), що відображається в об’єктах.

Модель представляється у вигляді класів із атрибутами, методами та зв’язками між ними. Для цього використовується нотація UML (Unified Modeling Language).

Далі розглядаються розроблені класи об’єктної моделі, сама діаграма класів наведена в Додатку 1.

Клас DataSource (Джерело даних) представляє джерело даних, таке як лічильник або датчик, має наступні атрибути:

- sourceId - Int - унікальний ідентифікатор джерела.

- Type - String - тип джерела.

- protocol: String - протокол зв’язку.

- frequency - Int - частота збору даних (секунди, наприклад, 300 для 5 хвилин).

- status - String - статус джерела ("підключено", "не підключено").

Методи:

- connect() - Boolean - підключення до джерела.

- disconnect() - Boolean - відключення від джерела.

- getData() - RawData - отримання необроблених даних із джерела.

Клас RawData (Необроблені дані) представляє необроблені дані, отримані від джерела.

Атрибути:

- dataId - String - унікальний ідентифікатор запису.

- sourceId - String - ідентифікатор джерела, з якого отримані дані.

- value - Double - значення.

- unit - String - одиниця виміру.

- timestamp - DateTime - часова мітка.

Методи:

- validate() - Boolean - перевірка даних на коректність.

Клас ProcessedData (Оброблені дані) представляє дані після обробки та фільтрації.

Атрибути:

- dataId - String - унікальний ідентифікатор запису.

- sourceId - String - ідентифікатор джерела.

- value - Double - оброблене значення.

- unit - String - одиниця виміру.

- timestamp - DateTime - часова мітка.

- status - String - статус обробки.

Методи:

- filter(): Boolean - фільтрація даних (видалення аномалій).

- save(): Boolean - збереження даних у базі даних.

Клас DataProcessor (Обробник даних) відповідає за обробку та фільтрацію даних.

Атрибути:

- processorId - String - ідентифікатор обробника.

- filterId - Int – ідентифікатор параметрів фільтрації.

Методи:

- process(rawData: RawData): ProcessedData - обробка необроблених даних.

- applyFilters(data: RawData): Boolean - застосування фільтрів до даних.

Клас Dashboard (Дашборд) представляє інтерфейс для візуалізації даних.

Атрибути:

- dashboardIdь - String - унікальний ідентифікатор дашборда.

- userId - String - ідентифікатор користувача (Адміністратор, Аналітик, Оператор).

- dataTypes - String - типи даних для відображення.

- period - String - період відображення.

- visualizationType - String - тип візуалізації ("графік", "таблиця", "діаграма").

Методи:

- render() - VisualizationData - створення даних для візуалізації.

- updateSettings(): Boolean - оновлення налаштувань дашборда.

Клас VisualizationData (Дані для візуалізації) представляє підготовлені дані для відображення на дашборді.

Атрибути:

- dataId: String - ідентифікатор даних.

- timestamps – DateTime - список часових міток.

- values – Double - список значень.

- unit - String - одиниця виміру.

- kpi - String - ключові показники ефективності.

Методи:

- formatForDisplay(): String - форматування даних для відображення.

Клас Forecast (Прогноз) представляє прогноз енергоспоживання.

Атрибути:

- forecastId - String - унікальний ідентифікатор прогнозу.
- period - String - період прогнозу.
- values – Double - прогнозовані значення.
- unit - String - одиниця виміру.
- modelType - String - тип моделі (наприклад, "ARIMA", "LSTM").

Методи:

- generateForecast(): Boolean - створення прогнозу.
- evaluateAccuracy(): Double - оцінка точності прогнозу.

Клас Anomaly (Аномалія) представляє виявлену аномалію в даних.

Атрибути:

- anomalyId - String - унікальний ідентифікатор аномалії.
- sourceId - String - ідентифікатор джерела.
- value - Double - аномальне значення.
- expectedValue - Double - очікуване значення.
- timestamp - DateTime - час виявлення.

Методи:

- notifyUser(user: User): Boolean — відправка сповіщення користувачу.

Клас Filter (Аномалія) представляє фільтр для обробки даних.

Атрибути:

- filterId - String - унікальний ідентифікатор фільтра.
- type - String - тип фільтра (наприклад, "range", "threshold").
- parameters – String - параметри фільтра.

Методи:

- apply(data: RawData): Boolean - застосування фільтра до даних.

Клас Report (Звіт) представляє згенерований звіт.

Атрибути:

- reportId - String - унікальний ідентифікатор звіту.
- period - String - період звіту.
- data – String - дані для звіту.

- kpi - String - ключові показники.

- recommendation – String - рекомендації.

- format - String - формат експорту ("PDF", "Excel", "CSV").

Методи:

- generateReport(): Boolean - генерація звіту.

- export(): File - експорт звіту у вказаному форматі.

Клас User (Користувач) представляє користувача системи.

Атрибути:

- userId - String - унікальний ідентифікатор користувача.

- role - String - роль ("Operator", "Administrator", "Analyst").

- username - String - ім’я користувача.

- password - String - зашифрований пароль.

- email - String - email для сповіщень.

Методи:

- authenticate(username: String, password: String): Boolean - автентифікація.

- getPermissions(): String - отримання прав доступу.

Клас SecurityManager (Менеджер безпеки) відповідає за безпеку даних і доступу.

Атрибути:

- encryptionKey - String - ключ шифрування.

Методи:

- encryptData(data: String): String - шифрування даних.

- decryptData(): String - розшифрування даних.

Робота моделі складається з наступних етапів:

1. Збір даних:

- DataSource підключається до датчиків і генерує RawData.

- DataProcessor обробляє RawData, створюючи ProcessedData, які зберігаються в базі даних.

2. Моніторинг і візуалізація:

- Dashboard отримує ProcessedData і генерує VisualizationData для відображення графіків, таблиць, KPI.

3. Аналіз і прогнозування:

- ProcessedData аналізуються для створення Forecast (прогнозів) і виявлення Anomaly (аномалій).

- Anomaly сповіщає User (Адміністратора).

4. Звітність:

- Report формується на основі ProcessedData, включаючи KPI та рекомендації, і експортується в потрібному форматі.

5. Безпека:

- SecurityManager шифрує дані та управляє доступом User до системи.

Об’єктна модель використовується для подальшої реалізації системи, заснованої на використанні об’єктно-орієнтованої мови програмування Python. На основі даної моделі в подальшому будується реляційна база даних.

3.3 Розробка архітектури

Розробка архітектури для інформаційної системи енергоменеджменту (EMS), призначеної для інтеграції до комплексної платформи оптимізації енергоспоживання в комерційних та промислових об’єктах, базується на функціональних і нефункціональних вимогах, об’єктній моделі, а також на необхідності масштабованості, інтеграції, безпеки та продуктивності. Архітектура повинна забезпечити модульність, гнучкість і ефективну взаємодію між компонентами системи.

Архітектура системи EMS розробляється з урахуванням таких принципів як модульність, масштабованість, інтеграція, безпека, продуктивність та зручність. Система поділяється на незалежні модулі для збору даних, обробки, аналізу, візуалізації, прогнозування, звітності, інтеграції та безпеки. Система повинна підтримувати велику кількість об’єктів (до 1000) і джерел даних (до 10 000). Підтримка різних протоколів (Modbus, MQTT,

BACnet, OPC UA) та API для зовнішніх систем. Шифрування даних, автентифікація, журналювання дій. Обробка даних у реальному часі з затримками не більше 1 секунди на джерело. Адаптивний інтерфейс для різних пристроїв (ПК, планшети, смартфони).

Для реалізації використовується мікросервісна архітектура, яка забезпечує незалежність модулів, легке масштабування та розгортання в хмарі. Додатково використовується шаруватий підхід (Layered Architecture) для логічного розділення компонентів.

Причини вибору мікросервісної архітектури наступні:

- Кожен модуль (збір даних, аналіз, візуалізація) може розроблятися, розгортатися та масштабуватися незалежно.

- Мікросервіси дозволяють масштабувати окремі компоненти (наприклад, модуль збору даних для великої кількості джерел).

- Кожен мікросервіс може для гнучкості використовувати різні протоколи та технології.

- Проблеми в одному модулі (наприклад, прогнозування) не впливають на інші (наприклад, візуалізацію).

Система поділяється на такі мікросервіси:

- Data Acquisition Service (Сервіс збору даних):
    - Відповідає за підключення до джерел даних (лічильники, датчики, HVAC) та збір даних у реальному часі.
    - Зберігає необроблені дані в черзі для подальшої обробки.
- Data Processing Service (Сервіс обробки даних):
    - Виконує фільтрацію, валідацію та обробку даних.
    - Перетворює необроблені дані (RawData) в оброблені (ProcessedData).
    - Зберігає оброблені дані в базі даних.
- Analytics Service (Сервіс аналізу):
    - Виявляє аномалії, закономірності та тенденції.
    - Генерує рекомендації щодо оптимізації енергоспоживання.

- Forecasting Service (Сервіс прогнозування):
    - Використовує алгоритми машинного навчання (ARIMA, LSTM) для прогнозування енергоспоживання.
    - Надає прогнози для заданого періоду.
- Visualization Service (Сервіс візуалізації):
    - Генерує дані для дашбордів (графіки, діаграми, KPI).
    - Забезпечує адаптивний інтерфейс для різних пристроїв.
- Reporting Service (Сервіс звітності):
    - Генерує звіти про енергоспоживання, KPI, рекомендації.
    - Експортує звіти в PDF, Excel, CSV.
- Integration Service (Сервіс інтеграції):
    - Забезпечує обмін даними з зовнішніми системами (Power BI, Tableau) через API.
    - Підтримує імпорт/експорт даних у JSON, CSV.
- Security Service (Сервіс безпеки):

Управляє автентифікацією та авторизацією користувачів.

- Шифрує дані (TLS 1.3).
- Веде журнал дій користувачів.

- Notification Service (Сервіс сповіщень):
    - Надсилає сповіщення про аномалії та аварійні ситуації (email, інтерфейс).

Для опису мікросервісної архітектури з шаруватим підходом розроблюваної системи створюється діаграма компонентів (рис. 3.1). Далі наводиться опис створеної діаграми компонентів.

Архітектура програми складається з 4 шарів:

- Шар представлення (Presentation Layer) містить інтерфейс користувача (Frontend).

- Шар бізнес-логіки (Application Layer) містить мікросервіси для обробки, аналізу, прогнозування тощо.

- Шар даних (Data Layer) містить бази даних (PostgreSQL, InfluxDB).

- Шар інтеграції (Integration Layer) забезпечує зв’язок із джерелами даних і зовнішніми системами.

Кожен мікросервіс представлений як окремий компонент із чітко визначеними інтерфейсами (API). Джерела даних (лічильники, датчики), хмарна інфраструктура (AWS), черги повідомлень (Kafka), зовнішні платформи (Power BI).

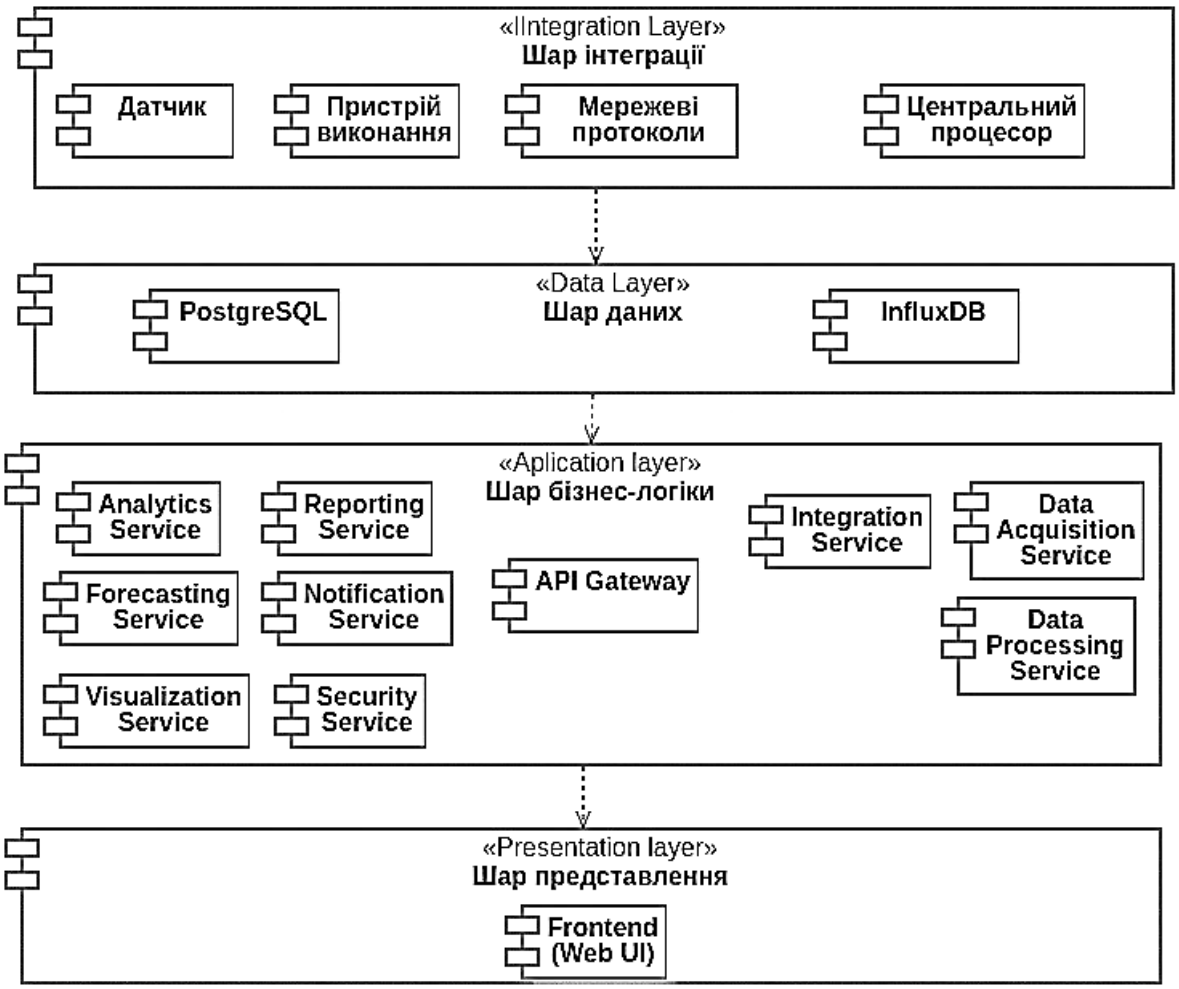


Рис. 3.1. Діаграма компонентів

Шар представлення (Presentation Layer) відповідає за взаємодію з користувачем (Оператор, Адміністратор, Аналітик).

Компонент: Frontend (Web UI)

Технологія: Angular.js.

Опис: Відображає дашборди, звіти, прогнози, сповіщення. Адаптивний інтерфейс для ПК, планшетів, смартфонів.

Інтерфейси:

Надані: HTTP/REST API для доступу до мікросервісів (Visualization Service, Reporting Service, Notification Service).

Потрібні: WebSocket для оновлення даних у реальному часі (наприклад, для дашбордів).

Зв’язки:

Підключається до API Gateway для маршрутизації запитів до мікросервісів.

Шар бізнес-логіки (Application Layer) містить основні мікросервіси, які реалізують логіку системи.

Компонент: API Gateway

Технологія: AWS API Gateway.

Опис: Точка входу для всіх запитів від Frontend. Забезпечує маршрутизацію, балансування навантаження, автентифікацію.

Інтерфейси:

Надані: REST API для Frontend.

Потрібні: REST API для взаємодії з мікросервісами.

Зв’язки:

Приймає запити від Frontend і перенаправляє їх до відповідних мікросервісів (Visualization Service, Reporting Service тощо).

Компонент: Data Acquisition Service

Технологія: Python (FastAPI).

Опис: Збирає дані від джерел (лічильники, датчики, HVAC) у реальному часі.

Інтерфейси:

Надані: REST API для налаштування джерел (наприклад, додавання нового датчика).

Потрібні: Протоколи MQTT, Modbus, BACnet для зв’язку з джерелами даних.

Зв’язки:

Надсилає необроблені дані (RawData) до бази даних.

Отримує дані від зовнішніх джерел (лічильники, датчики).

Компонент: Data Processing Service

Технологія: Python (FastAPI).

Опис: Обробляє та фільтрує дані, перетворюючи RawData у ProcessedData.

Інтерфейси:

Надані: REST API для управління фільтрами.

Потрібні: Kafka Consumer API для читання з черги.

Зв’язки:

Отримує RawData з бази даних.

Зберігає ProcessedData в InfluxDB та PostgreSQL.

Компонент: Analytics Service

Технологія: Python (FastAPI).

Опис: Виявляє аномалії, закономірності, генерує рекомендації.

Інтерфейси:

Надані: REST API для доступу до результатів аналізу.

Потрібні: JDBC/ODBC для доступу до InfluxDB і PostgreSQL.

Зв’язки:

Читає ProcessedData з InfluxDB.

Зберігає аномалії в PostgreSQL (anomalies).

Надсилає сповіщення через Notification Service.

Компонент: Forecasting Service

Технологія: Python (FastAPI) + TensorFlow.

Опис: Генерує прогнози енергоспоживання.

Інтерфейси:

Надані: REST API для доступу до прогнозів.

Потрібні: JDBC/ODBC для доступу до InfluxDB і PostgreSQL.

Зв’язки:

Читає історичні дані (ProcessedData) з InfluxDB.

Зберігає прогнози в PostgreSQL (forecasts).

Компонент: Visualization Service

Технологія: Python (FastAPI).

Опис: Готує дані для відображення на дашбордах.

Інтерфейси:

Надані: REST API для Frontend.

Потрібні: JDBC/ODBC для доступу до InfluxDB і PostgreSQL.

Зв’язки:

Читає ProcessedData з InfluxDB.

Читає налаштування дашбордів із PostgreSQL (dashboards).

Використовує Redis для кешування даних.

Компонент: Reporting Service

Технологія: Python (FastAPI).

Опис: Генерує та експортує звіти.

Інтерфейси:

Надані: REST API для Frontend.

Потрібні: JDBC/ODBC для доступу до PostgreSQL.

Зв’язки:

Читає дані з PostgreSQL (reports, ProcessedData через InfluxDB).

Зберігає звіти в базі даних.

Компонент: Notification Service

Технологія: Python (FastAPI).

Опис: Надсилає сповіщення про аномалії.

Інтерфейси:

Надані: REST API для інших сервісів.

Потрібні: SMTP для відправки email.

Зв’язки:

Отримує дані від Analytics Service.

Надсилає сповіщення користувачам.

Компонент: Security Service

Технологія: ASP .NET Core.

Опис: Управляє автентифікацією, авторизацією, шифруванням.

Інтерфейси:

Надані: OAuth 2.0 API для автентифікації.

Потрібні: JDBC для доступу до PostgreSQL.

Зв’язки:

Зберігає журнали в PostgreSQL (security_logs).

Використовується API Gateway для автентифікації запитів.

Компонент: Integration Service

Технологія: Java (Spring Boot).

Опис: Забезпечує інтеграцію з зовнішніми системами (Power BI).

Інтерфейси:

Надані: REST API для експорту/імпорту даних.

Потрібні: JDBC для доступу до PostgreSQL.

Зв’язки:

Читає дані з PostgreSQL (integrations).

Експортує дані до зовнішніх систем.

Шар даних (Data Layer) містить бази даних і системи зберігання.

Компонент: PostgreSQL

Опис: Зберігає структуровані дані (користувачі, звіти, прогнози, аномалії).

Зв’язки:

Використовується мікросервісами (Security Service, Reporting Service, Forecasting Service тощо).

Компонент: InfluxDB

Опис: Зберігає часові ряди (RawData, ProcessedData).

Зв’язки:

Використовується Data Processing Service, Analytics Service, Visualization Service.

Компонент: InfluxDB

Опис: Кеш для швидкого доступу до даних (наприклад, налаштувань дашбордів).

Зв’язки:

Використовується Visualization Service.

Компонент: база даних

Опис: Зберігає згенеровані звіти (PDF, Excel).

Зв’язки:

Використовується Reporting Service.

Шар інтеграції (Integration Layer) забезпечує зв’язок із зовнішніми системами та джерелами даних.

Компонент: Data Sources (External)

Опис: Лічильники, датчики, HVAC, системи освітлення.

Зв’язки:

Підключаються до Data Acquisition Service через MQTT, Modbus, BACnet.

Компонент: база даних

Опис: Черга для асинхронного обміну даними між мікросервісами.

Зв’язки:

Data Acquisition Service надсилає дані в чергу.

Data Processing Service читає дані з черги.

Компонент: External Systems (Power BI, Tableau)

Опис: Зовнішні платформи для аналітики.

Зв’язки:

Підключаються до Integration Service через REST API.

3.4 Засоби розробки

Для первинної розробки використовується такий інструмент як Flask, що є легким і гнучким веб-фреймворком для Python, дозволяючи швидко створювати веб-додатки. Він мінімалістичний, надаючи лише основні інструменти для розробки, що робить його простим у використанні та легким для вивчення. Flask базується на Werkzeug (WSGI-бібліотека) та Jinja2 (рушій шаблонів), що забезпечує зручну обробку HTTP-запитів і створення динамічних HTML-сторінок.

Основні властивості Flask включають вбудований сервер для розробки, підтримку маршрутизації URL, обробку GET, POST та інших типів запитів, а також можливість інтеграції з базами даних, формами та іншими бібліотеками через розширення. Він підтримує шаблони для створення інтерфейсів, має зручну систему налагодження та дозволяє легко масштабувати проєкти. Flask не нав’язує строгих правил чи структури, що дає розробникам свободу вибору інструментів і архітектури. Завдяки цьому він ідеально підходить для невеликих і середніх проєктів, прототипів, API та мікросервісів.

Flask надає набір основних інструментів для створення веб-додатків, зберігаючи мінімалістичний підхід. Ось конкретні інструменти, які входять до складу фреймворку:

- Вбудований WSGI-сервер. Flask має легкий сервер для розробки, який дозволяє швидко тестувати додатки без необхідності налаштування зовнішнього сервера (наприклад, Gunicorn використовується для продакшену).

- Маршрутизація URL. Декоратор @app.route() дозволяє пов’язувати URL-адреси з функціями Python, обробляючи запити до різних ендпоінтів (наприклад, /home, /api/data).

- Обробка HTTP-запитів. Підтримка методів GET, POST, PUT, DELETE тощо, з доступом до параметрів запиту, заголовків і тіла через об’єкт request.

- Шаблонізатор Jinja2. Вбудований рушій шаблонів для створення динамічних HTML-сторінок, який підтримує змінні, цикли, умови та спадкування шаблонів.

- Контекст додатка та запиту. Об’єкти app і request дозволяють управляти станом додатка, зберігати дані сесії, конфігурацію та обробляти запити.

- Система конфігурації. Flask дозволяє налаштовувати додаток через об’єкт app.config, де можна задавати параметри, такі як секретні ключі, підключення до баз даних тощо.

- Підтримка сесій. Вбудована система управління сесіями для зберігання даних користувача між запитами (на основі підписаних cookies).

- Обробка помилок. Інструменти для створення кастомних сторінок помилок (наприклад, 404, 500) через декоратори типу @app.errorhandler.

- Сигнали. Механізм для обробки подій у додатку (за допомогою бібліотеки blinker), що дозволяє реагувати на певні дії (наприклад, перед або після запиту).

- Тестовий клієнт. Вбудований інструмент для симуляції HTTP-запитів під час тестування додатка без запуску сервера.

- Підтримка Blueprint. Механізм для модульної організації коду, що дозволяє групувати маршрути та логіку для великих проєктів.

- Розширення та інтеграції. Flask легко інтегрується з бібліотеками, такими як SQLAlchemy (для баз даних), WTForms (для форм), Flask-RESTful (для API) тощо, хоча ці інструменти не входять до ядра фреймворку.

- Налагоджувальний режим. Вбудований дебагер, який виводить детальну інформацію про помилки прямо в браузері, полегшуючи розробку.

Flask не включає ORM, інструменти для роботи з формами чи автентифікацію «з коробки», але його гнучкість дозволяє легко додавати ці функції через сторонні бібліотеки. Завдяки цьому він підходить для створення як простих, так і складних веб-додатків.

В якості системи управління інформаційною базою додатку обирається реляційна база даних SQLite виключно для цілей розробки.

SQLite - це вбудована реляційна база даних, що поставляється з вихідними кодами. Вперше випущена в 2000 році, призначена для предоставлення звичних можливостей реляційних баз даних без властивих їм накладних витрат. За час експлуатації встигла заслужити репутацію як переносної, легкої у використанні, компактної, продуктивної і надійної бази даних.

Вбудовуваність бази даних означає, що вона існує не як процес, окремий від обслуговується процесу, а є його частиною - частиною деякого прикладного застосування. Зовнішній спостерігач не помітить, що прикладний додаток користується СУБД. Додаток просто робить його роботу, движок БД просто міститься всередині. Це позбавляє від необхідності мережевих налаштувань і адміністрування. Подумайте як це полегшує життя: ніяких файерволів, ніяких мережевих адрес, ніяких користувачів і конфліктів їх прав доступу. І клієнт, і сервер працюють в одному процесі. Це позбавляє від проблем конфігурації. Все, чого потребує програміст вже скомпільовано в його додатку.

Згідно рис. 3.2, скрипт Perl, звичайна програма на Сі, скрипт PHP - все використовують SQLite.

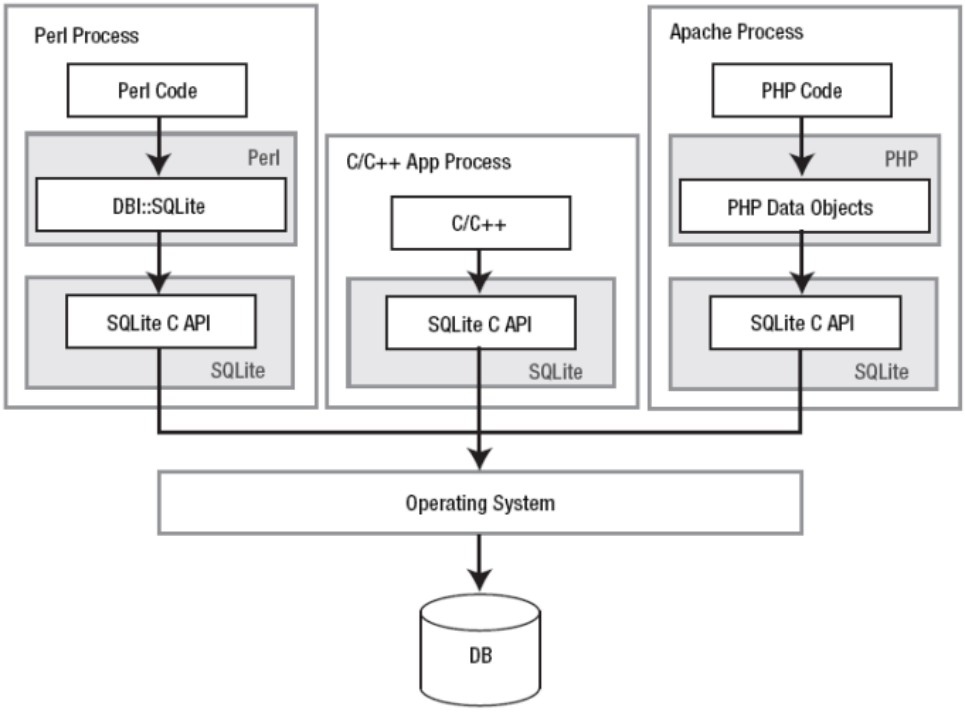


Рис. 3.2. SQLite вбудовується до додатків.

Зрештою, кожен з трьох процесів використовується SQLite прикладним інтерфейсом C. Таким чином, SQLite вбудований до адресного простору кожного з них. Кожен з них стає незалежним сервером бази даних. І, крім того, хоча кожен процес являє незалежний сервер, вони можуть виконувати операції на одному і тому ж файлі (ах) бази даних. SQLite дозволяє їм справлятися з синхронізацією і блокуваннями.

На поточному ринку вбудованих баз даних представлено багато продуктів від різних виробників, але тільки один з них поставляється з відкритим кодом, не вимагає ліцензійних зборів і спроектований як вбудована БД - це SQLite.

Архітектура. SQLite має елегантну модульну архітектуру, яка буде показувати унікальні підходи до управління реляційними базами даних. Вісім окремих модулів згруповані до трьох головних підсистем (див. 2.2). Вони поділяють обробку запиту на окремі завдання, які працюють подібно конвеєру. Верхні модулі компілюють запити, середні виконують їх, а нижні управляються з диском і взаємодіють з операційною системою.

Інтерфейс. Інтерфейс є верхнім модулем і складається з програмного інтерфейсу мови C (API). Це означає, що через нього з SQLite взаємодіють додатки, скрипти і бібліотеки. Образно кажучи, через цей інтерфейс розробники, адміністратори, студенти і божевільні вчені розмовляють з SQLite.

Компілятор. Процес компіляції починається з лексичного аналізатора і парсеру.

Віртуальна машина (движок БД) називається virtual database engine (VDBE)

Незважаючи на маленький розмір, SQLite надає визначний спектр особливостей і можливостей. Він підтримує вельми повний набір стандарту ANSI SQL92 для особливостей мови SQL, а так ¬ такі особливості як тригера, індекси, стовпці з автоінкрементом, LIMIT / OFFSET особливості. Так само підтримуються такі рідкісні властивості, як динамічна типізація і вирішення конфліктів.

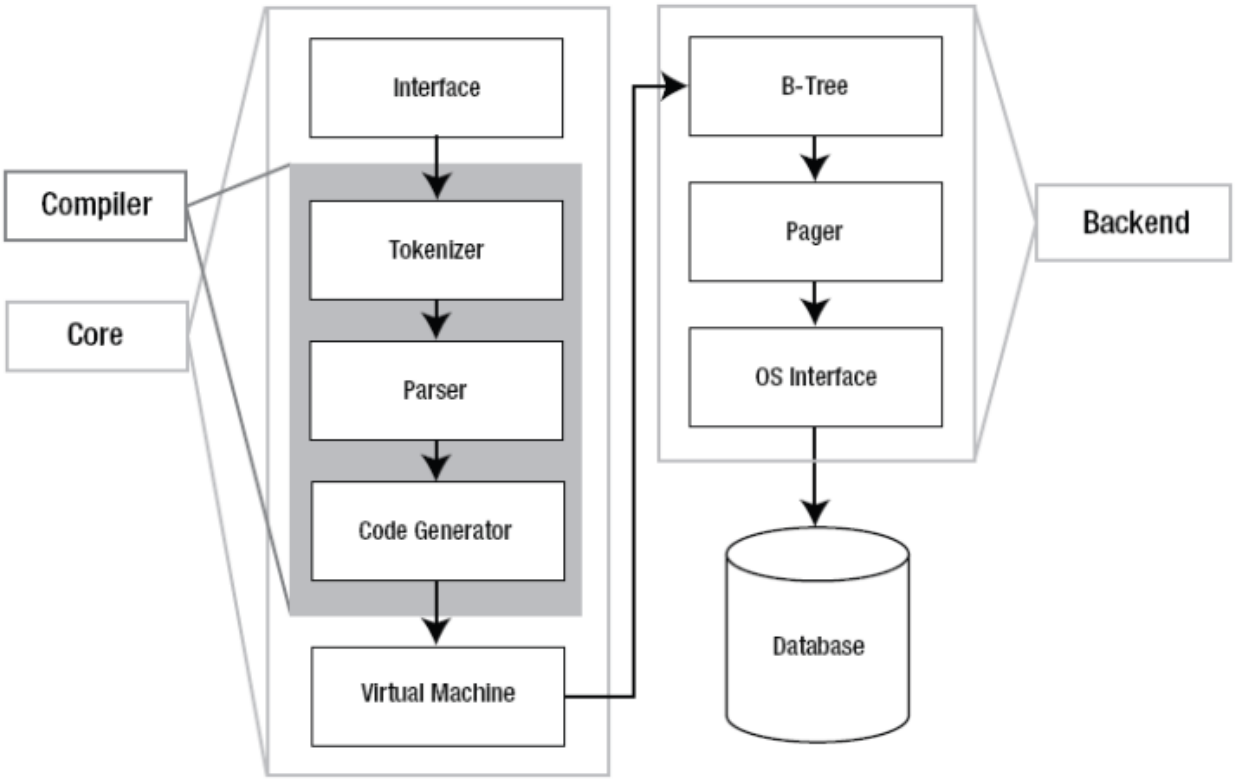


Рис. 3.3. Архітектура SQLite

Для SQLite використовується один з діалектів мови програмування SQL.

SQL (Structured Query Language - Структурована мова запитів) - мова управління базами даних для реляційних баз даних.

Для практичного створення бази даних використовується спеціалізована програма DB Browser for SQLite (DB4S), що є високоякісним візуальним інструментом із відкритим кодом для створення, проектування та редагування файлів баз даних, сумісних із SQLite.

3.5 Проєктування інтерфейсу програмної системи

Для моделювання графічного інтерфейсу використовується побудова діаграми активності, яка дозволяє точно розділити дії користувачів та системи в процесі поточної роботи. Розроблена діаграма активності UML для системи EMS чітко відображає послідовність і паралельність дій у системі, враховуючи ролі користувачів і мікросервісну архітектуру. Вона показує, як система

обробляє дані в реальному часі, реагує на аномалії, генерує прогнози та звіти, забезпечуючи ефективне управління енергоспоживанням.

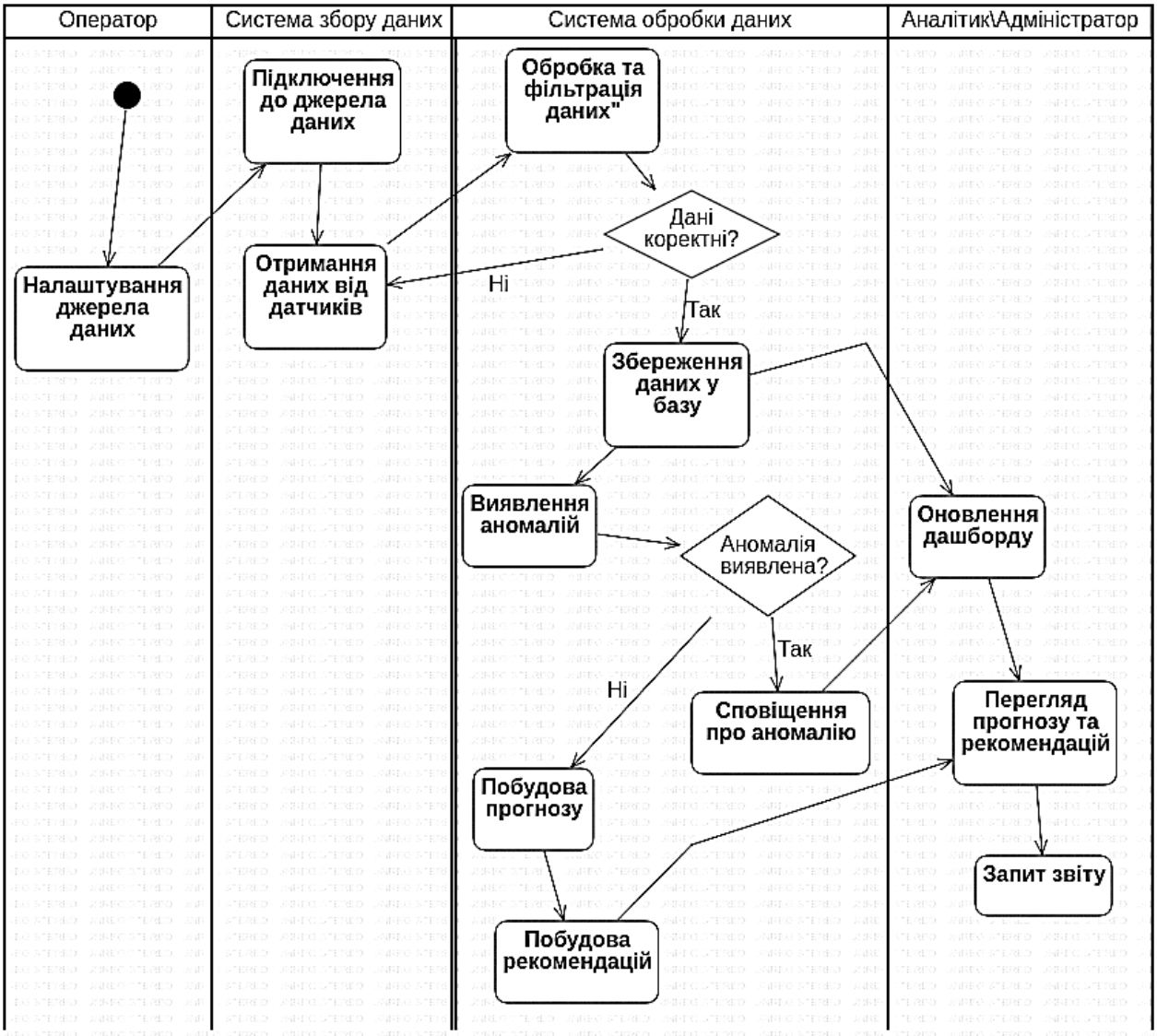


Рис. 3.3. Діаграма активності.

Графічний інтерфейс для системи EMS розроблений в програмі Figma з урахуванням потреб користувачів (Оператор, Адміністратор, Аналітик) і функціональних вимог (моніторинг, аналіз, прогнозування, звітність). Він є інтуїтивно зрозумілим, адаптивним і підтримує швидкий доступ до ключових функцій через дашборди, меню та фільтри. Використання сучасного дизайну (світла тема, чітка типографіка, адаптивність) забезпечує зручність використання на різних пристроях, що відповідає вимогам системи.

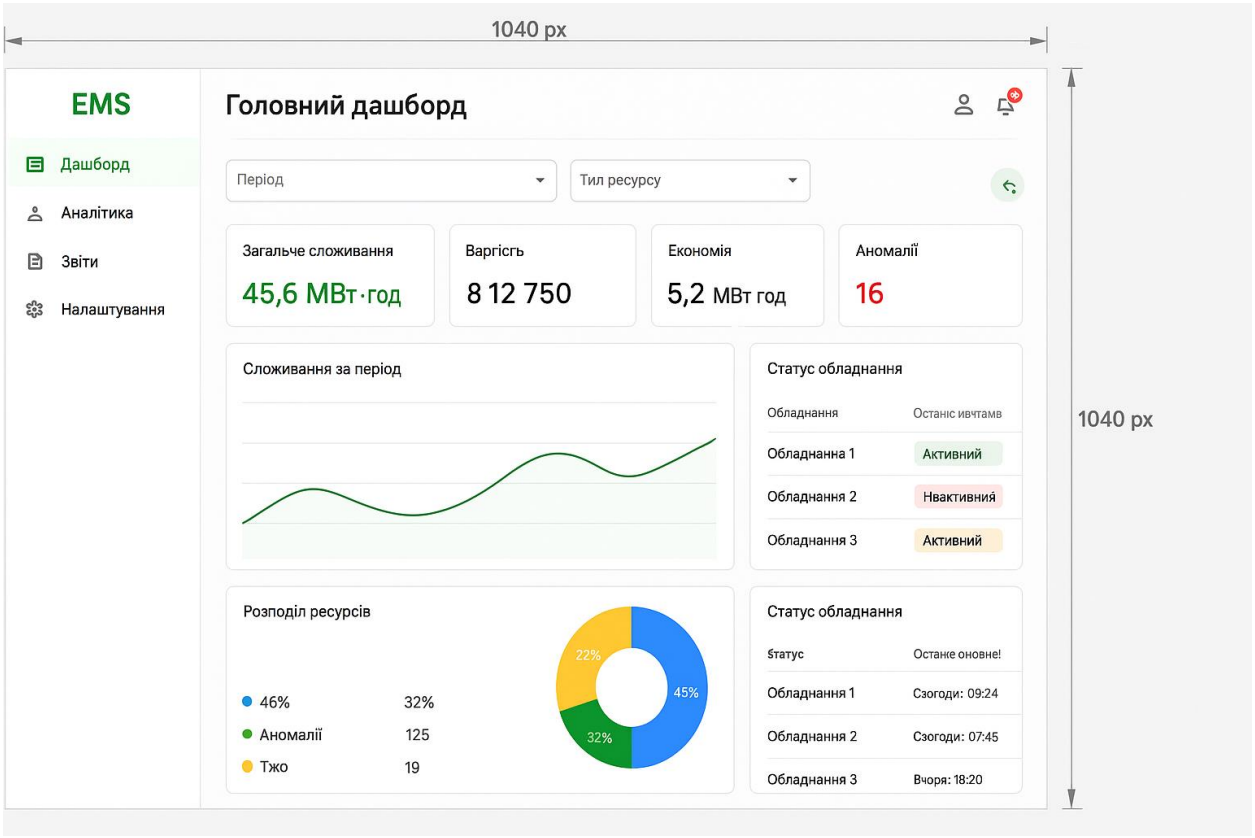


Рис. 3.4. Прототип графічного інтерфейсу головного вікна додатку

3.6 Опис програмної реалізації

Файл app.py є серверною частиною програми для системи керування енергоспоживанням (EMS - Energy Management System). Він реалізований на основі фреймворку Flask і забезпечує API для взаємодії з базою даних energy.db, обробки даних, керування джерелами даних, користувачами та аномаліями. Нижче наведено детальний опис програмної реалізації файлу app.py з урахуванням рекомендацій та внесених змін.

Основна мета є забезпечити серверну логіку для створення, керування та отримання даних про енергоспоживання, підтримуючи інтеграцію з клієнтською частиною (client.py) та генератором даних (generate_energy_data.py).

Ключові компоненти це засоби ініціалізація бази даних, класи для роботи з даними (DataSource, RawData, ProcessedData тощо), API-ендпоінти для керування джерелами, даними та автентифікацією.

Функція init_db() створює таблиці в базі energy.db.

Файл містить набір класів для роботи з даними та логікою системи (DataSource, RawData, ProcessedData, DataProcessor та інші згідно розробленої раніше діаграмі класів):

```
class DataSource:
    def __init__(self, sourceId: int, type: str, protocol: str, frequency: int, status: str):
        self.sourceId = sourceId
        self.type = type
        self.protocol = protocol
        self.frequency = frequency
        self.status = status
```

API-ендпоінти забезпечують взаємодію з клієнтською частиною (client.py) та генератором даних:

POST /api/datasource:

Створює нове джерело даних у таблиці data_sources.

Очікує JSON із полями sourceId, type, protocol, frequency, status.

POST /api/datasource/<sourceid>/connect</sourceid>:

Оновлює статус джерела на "підключено".

Повертає помилку 404, якщо джерело не знайдено.

GET /api/datasource/<sourceid>/data</sourceid>:

Повертає один запис сирих даних для джерела (статичний, для демонстрації).

Формат відповіді: JSON із dataId, sourceId, value, unit, timestamp.

GET /api/datasource/<sourceid>/timeseries</sourceid>:

Повертає всі записи з таблиці raw_data для заданого sourceId, відсортовані за timestamp.

Формат відповіді: список JSON-об'єктів із timestamp і value.

GET /api/raw_data/<sourceid></sourceid>:

Повертає сирі дані за sourceId з можливістю фільтрації за періодом (day або всі дані).

Використовує параметр запиту period.

POST /api/user:

Створює нового користувача в таблиці users із захешованим паролем.

POST /api/login:

Автентифікує користувача за логіном і паролем, повертаючи userId у разі успіху.

Файл generate_energy_data.py є скриптом для генерації синтетичних часових рядів даних, що імітують споживання електроенергії, та їх запису в базу даних SQLite energy.db. Він розроблений для створення реалістичних даних із добовими, тижневими та сезонними коливаннями, а також із додаванням шуму та аномалій. Ці дані використовуються для тестування серверної (app.py) та клієнтської (client.py) частин системи керування енергоспоживанням (EMS). Нижче наведено детальний опис програмної реалізації файлу.

Відбувається програмна генерація часових рядів даних із профілями споживання електроенергії та їх збереження в таблиці raw_data бази energy.db.

Основна функція: generate_energy_timeseries(start_date, days, source_ids, interval_minutes=5) - створює дані для заданих джерел (source_ids) за вказаний період із заданим інтервалом. Скрипт взаємодіє з базою energy.db, створеною в app.py, і генерує дані, які можуть бути використані клієнтською частиною (client.py) для відображення на дашборді. Функція є основним компонентом скрипта і генерує часові ряди з урахуванням профілів споживання.

Параметри:

- start_date: str: Початкова дата у форматі 'YYYY-MM-DD' (наприклад, '2025-05-28').

- days: int: Кількість днів для генерації даних.

- source_ids: list: Список ідентифікаторів джерел (наприклад, [1, 2, 3]).

- interval_minutes: int: Інтервал між записами в хвилинах (за замовчуванням 5 хвилин).

Файл client.py є клієнтською частиною системи керування енергоспоживанням (EMS - Energy Management System), реалізованою за

допомогою бібліотеки tkinter для створення графічного інтерфейсу користувача (GUI). Він забезпечує відображення даних про енергоспоживання, джерела даних, аномалії, прогнози та звіти, взаємодіючи з серверною частиною (app.py) через REST API та базу даних SQLite (energy.db).

Використовуються наступні бібліотеки: tkinter для GUI, requests для HTTP-запитів, matplotlib для графіків, sqlite3 для прямого доступу до бази даних.

Клас EMSApp є основою клієнтської програми, керуючи всіма екранами та взаємодією з сервером.

Ініціалізація:

```
def __init__(self, root):
    self.root = root
    self.root.title("EMS - Energy Management System")
    self.root.geometry("1280x720")
    self.root.configure(bg=COLORS["background"])
    self.current_user = None
    self.show_login_screen()
```

Налаштовує головне вікно (root) розміром 1280x720 пікселів із білим фоном.

Відображає екран керування джерелами даних. Містить кнопку "Додати джерело" та таблицю з інформацією про джерела (sourceId, тип, протокол, частота, статус). Виконує запит до /api/datasource/1/data для демонстрації (показує статичний приклад).

Висновки до розділу 3

Детально описані проєктні рішення, прийняті для розробки інформаційної системи керування енергоспоживанням (EMS), призначеної для інтеграції в платформу оптимізації енергоспоживання комерційних та промислових об’єктів.

Розроблена UML-діаграма класів, яка включає ключові класи (DataSource, RawData, ProcessedData, DataProcessor, Dashboard, VisualizationData, Forecast, Anomaly, Filter, Report, User, SecurityManager). Ці класи відображають логіку збору, обробки, аналізу, візуалізації, прогнозування та безпеки, забезпечуючи модульність і підтримку ролей користувачів. Використовується мікросервісна архітектура з шаруватим підходом, що включає шари представлення, бізнес-логіки, даних та інтеграції. Мікросервіси (Data Acquisition, Data Processing, Analytics, Forecasting, Visualization, Reporting, Integration, Security, Notification) забезпечують модульність, масштабованість і незалежність компонентів.

Для первинної реалізації обрано фреймворк Flask завдяки його легкості, гнучкості та підтримці REST API. SQLite використовується як вбудована база даних для розробки через простоту, портативність і відсутність потреби в мережевій конфігурації.

Серверна частина в вигляді файлу app.py на Flask забезпечує API для керування джерелами, даними, користувачами та аномаліями, взаємодіючи з SQLite (energy.db). Містить класи, відповідні об’єктній моделі, і ендпоінти (/api/datasource, /api/timeseries, /api/login тощо). Клієнтська частина в вигляді файлу client.py на tkinter відображає дашборди, графіки, аномалії та керує джерелами, взаємодіючи з API або SQLite. Використовує matplotlib для візуалізації.

Розділ 4. ТЕСТУВАННЯ І ОЦІНКА РЕЗУЛЬТАТІВ

4.1 Опис програмного продукту

В програмі Visual Studio Code створюється папка проекту ems, до якої завантажуються розроблені раніше файли app.py, generate_energy_data.py та client.py. Активується віртуальне середовище, й через два окремих термінали запускається серверна та клієнтська частина додатку командами:

python app.py

python generate_energy_data.py

python client.py

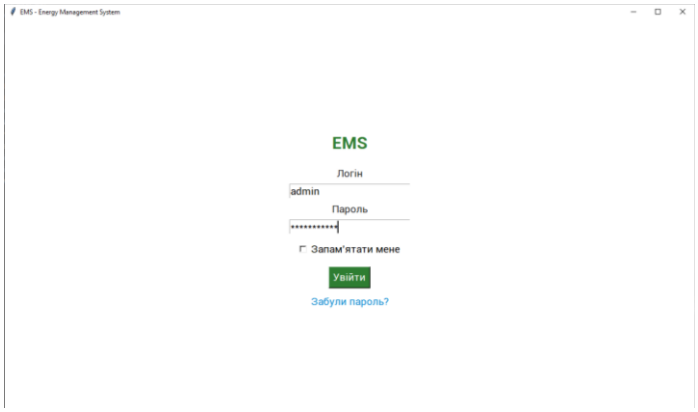


Рис. 4.1. Авторизація в програмі. Логін – admin, пароль – password123

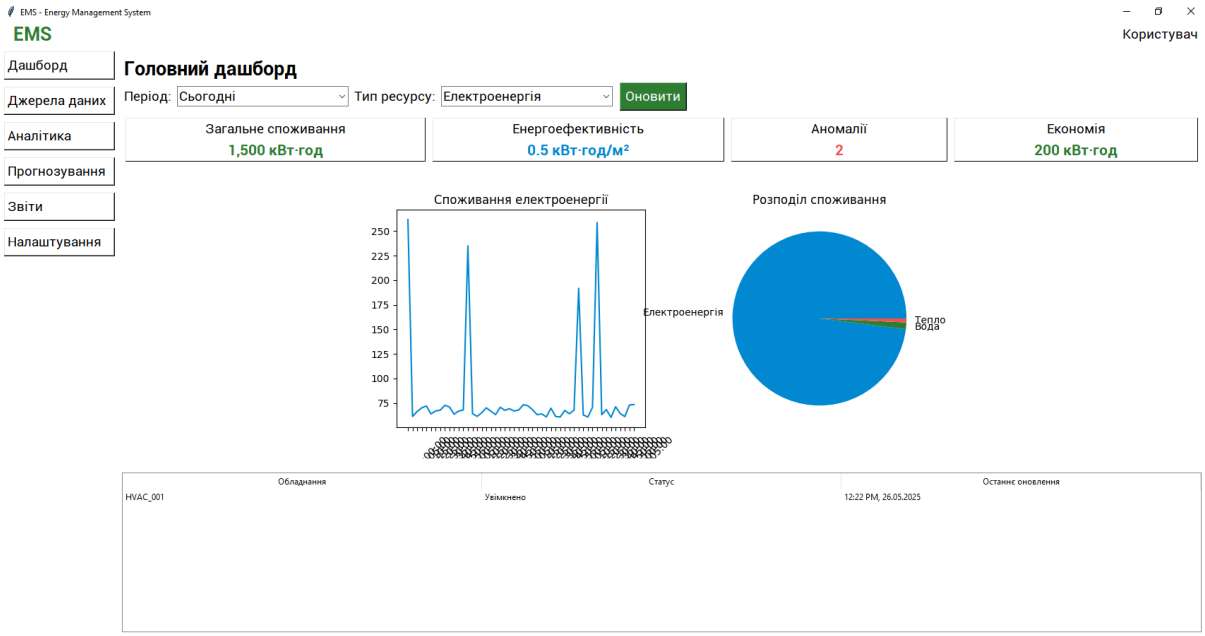


Рис. 4.2. Головне вікно програми

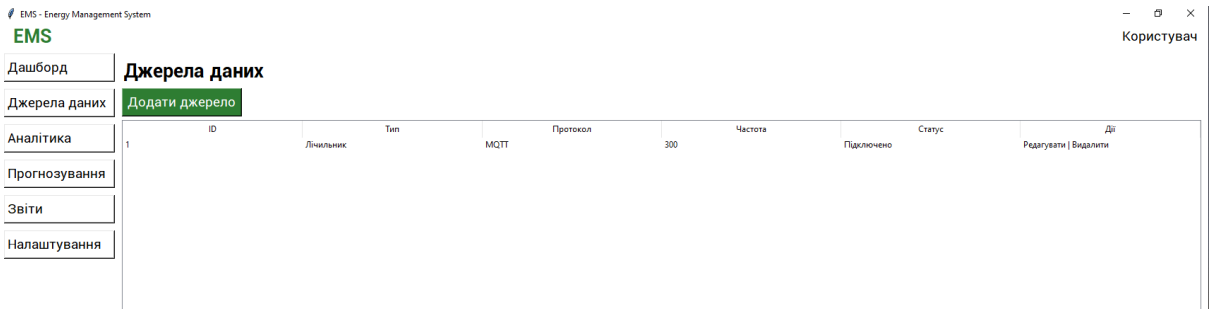


Рис. 4.3. Перехід до сторінки «Джерела»

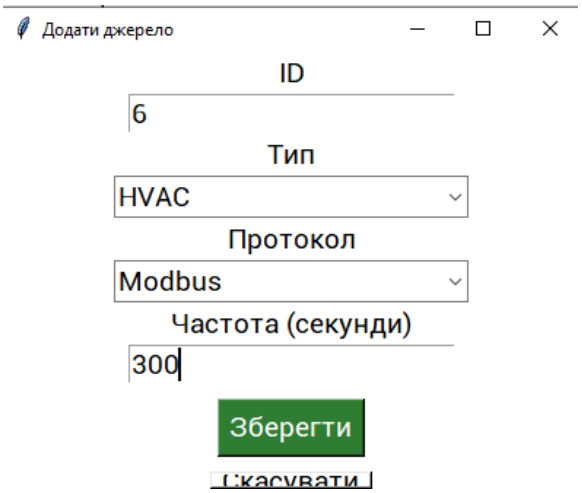


Рис. 4.4. Додавання нового датчика

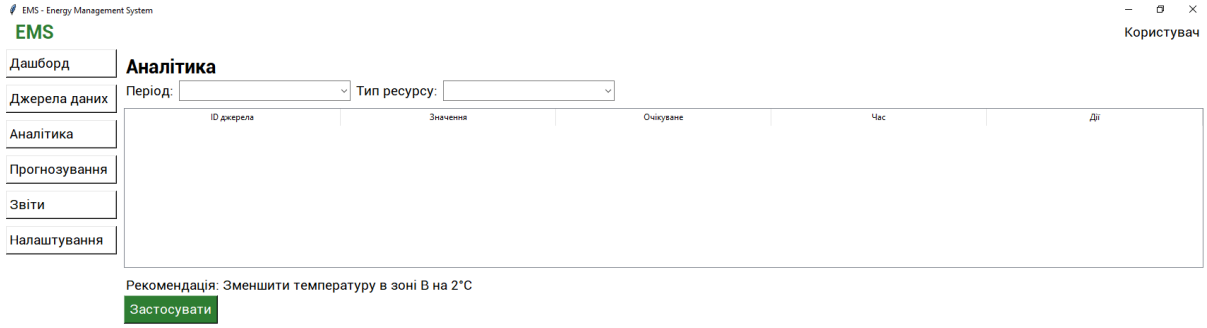


Рис. 4.5. Перехід до сторінки «Аналітика»

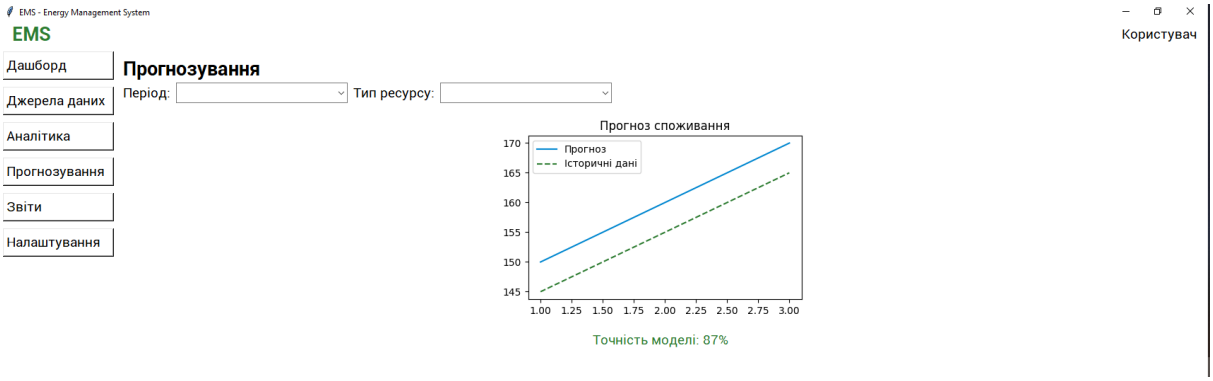


Рис. 4.6. Перехід до сторінки «Прогнозування»

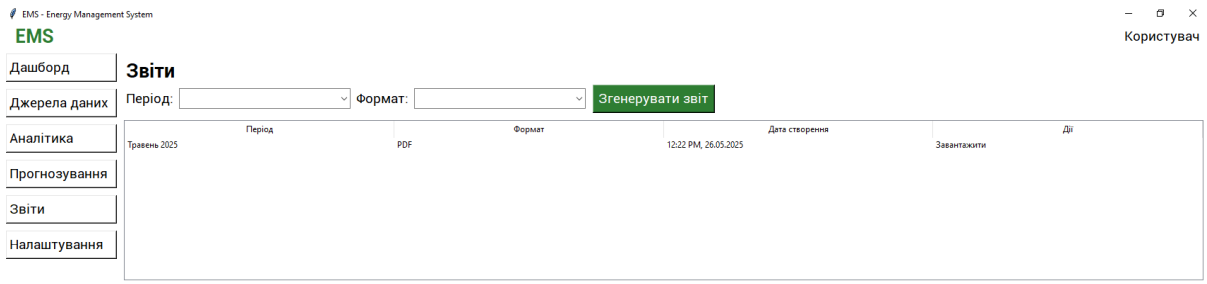


Рис. 4.7. Перехід до сторінки «Звіти»

4.2 Методи тестування програмного забезпечення

4.2.1. Тестування на реальних і змодельованих даних

Для комплексного тестування розроблюваного додатку потрібно застосувати кілька методів, щоб охопити серверну, клієнтську частини, базу даних та їх взаємодію. Основні методи включають:

1. Модульне тестування (Unit Testing)

Мета: Перевірити коректність роботи окремих компонентів (класів, методів, функцій) у ізоляції.

Застосування до EMS:

Серверна частина:

Тестування класів, таких як DataSource, RawData, ProcessedData тощо.

Наприклад, перевірка методу DataSource.connect() на оновлення статусу в базі даних.

Перевірка методу ProcessedData.filter() на правильне виявлення аномалій.

Клієнтська частина:

Тестування методу EMSApp.login() на коректну обробку успішного та неуспішного входу.

Тестування методу save_source() на валідацію введених даних.

Інструменти:

Python: unittest, pytest.

2. Інтеграційне тестування (Integration Testing)

Мета: Перевірити взаємодію між компонентами системи (API, база даних, клієнт).

Застосування до EMS:

Перевірка коректності роботи API-ендпоінтів, таких як /api/datasource та /api/login.

Тестування повного циклу: створення джерела даних через клієнт, збереження в базі даних, відображення в таблиці show_data_sources.

Перевірка коректності відображення графіків у show_main_dashboard при отриманні даних з API або бази даних.

Інструменти: pytest з requests для тестування API, selenium або pywinauto для тестування GUI.

3. Функціональне тестування (Functional Testing)

Мета: Перевірити, чи відповідає функціонал вимогам.

Застосування до EMS:

Перевірка, чи може користувач увійти в систему з правильними/неправильними обліковими даними.

Перевірка, чи додається нове джерело даних через діалогове вікно show_add_source_dialog.

Перевірка коректності відображення KPI, графіків та таблиць у всіх розділах (Дашборд, Аналітика, Прогнозування, Звіти).

Інструменти: Ручне тестування або автоматизоване через selenium для GUI, postman для API.

4. Тестування інтерфейсу користувача (UI Testing)

Мета: Перевірити, чи коректно працює графічний інтерфейс.

Застосування до EMS:

Перевірка, чи всі елементи (кнопки, комбобокси, таблиці) відображаються правильно.

Тестування навігації між розділами (Дашборд, Джерела даних тощо).

Перевірка реакції інтерфейсу на некоректні введення (наприклад, порожні поля в show_add_source_dialog).

Інструменти: selenium, pywinauto.

Тестування на реальних і змодельованих даних є ключовим для розроблюваного додатку, оскільки EMS працює з даними енергоспоживання.

Використовуйте реальні дані з датчиків енергоспоживання (наприклад, лічильників електроенергії, води чи тепла).

Якщо доступні реальні дані, імпортуйте їх у базу даних raw_data через CSV або API.

Сценарії тестування:

Перевірте коректність відображення графіків у show_main_dashboard для реальних даних за день, тиждень, місяць.

Перевірте, чи метод ProcessedData.filter() правильно виявляє аномалії (наприклад, значення > 500 кВт·год).

Перевірте коректність розрахунків KPI (загальне споживання, енергоефективність) у show_main_dashboard.

Приклад тесту:

Завантажте реальні дані за тиждень (наприклад, 1000 записів із різними значеннями value та timestamp).

Перевірте, чи відображаються дані в графіку show_main_dashboard та чи правильно обчислюється сума споживання.

Інструмент: Python-скрипт для імпорту даних, sqlite3 для перевірки.

Змодельовані дані

Генерація даних:

Використовуйте скрипт для створення синтетичних даних, як показано вище.

Імітуйте аномалії (наприклад, значення value > 500) для перевірки роботи ProcessedData.filter():

```
def generate_anomaly_data():
    with sqlite3.connect('energy.db') as conn:
```

```
c = conn.cursor()
c.execute("INSERT INTO raw_data VALUES (?, ?, ?, ?, ?)",
    (str(uuid.uuid4()), "1", 600.0, "kWh", datetime.now().isoformat()))
conn.commit()
```

Сценарії тестування:

Перевірте, чи аномалії з’являються в таблиці anomalies після обробки даних.

Перевірте відображення аномалій у show_analytics.

Перевірте, чи коректно відображаються графіки для великих наборів даних (наприклад, 50,000 записів).

Інструменти: Python-скрипти, pytest для автоматизації.

Порівняння реальних і змодельованих даних

Мета: Переконатися, що система однаково добре обробляє обидва типи даних.

Тестування:

Порівняйте результати графіків і KPI для реальних і змодельованих даних.

Перевірте, чи однаково виявляються аномалії для обох типів даних.

Перевірте швидкість обробки великих наборів даних (наприклад, 1 млн записів).

4.2.2. Аналіз результатів оптимізації енергоспоживання

Припустимо, додаток використовується для моніторингу енергоспоживання в офісній будівлі. У будівлі встановлено лічильник електроенергії (джерело даних з sourceId=1), який фіксує споживання кожні 5 хвилин. Дані зберігаються в таблиці raw_data. Мета аналізу — виявити аномалії в споживанні, надати рекомендацію для зменшення витрат енергії та оцінити економію після її застосування.

Налаштування даних

Для аналізу використаємо набір даних за один день (288 записів, оскільки 24 години × 12 вимірів на годину). Дані включають нормальне споживання (100–150 кВт·год) та кілька аномалій (споживання > 500 кВт·год через увімкнене обладнання в неробочий час).

Скрипт для генерації даних (змодельовані дані, що імітують реальні):

```
import sqlite3
from datetime import datetime, timedelta
import uuid

def generate_test_data():
    with sqlite3.connect('energy.db') as conn:
        c = conn.cursor()
        base_time = datetime(2025, 5, 28, 0, 0)
        for i in range(288):
            timestamp = (base_time + timedelta(minutes=5 * i)).isoformat()
            value = 100 + (i % 50) * 2  # Нормальне споживання 100–150 кВт·год
            if i in [50, 150, 250]:  # Аномалії в певний час
                value = 600  # Високе споживання
            c.execute("INSERT INTO raw_data (dataId, sourceId, value, unit, timestamp) VALUES (?, ?, ?, ?, ?)",
                      (str(uuid.uuid4()), "1", value, "kWh", timestamp))
        conn.commit()

generate_test_data()
```

Обробка даних у системі

Додаток обробляє дані через клас ProcessedData, який викликає метод filter() для виявлення аномалій (значення > 500 кВт·год). Аномалії записуються в таблицю anomalies.

```
def process_data():
    with sqlite3.connect('energy.db') as conn:
        c = conn.cursor()
        c.execute("SELECT dataId, sourceId, value, unit, timestamp FROM raw_data WHERE sourceId='1'")
        for row in c.fetchall():
```

```
        raw_data = RawData(row[0], row[1], row[2], row[3], row[4])
        processor = DataProcessor("proc1", 1)
        processed = processor.process(raw_data)
        if processed and processed.filter():
            processed.save()
        else:
            # Аномалія вже записана в таблицю anomalies у методі filter()
            pass

process_data()
```

Після виконання цього скрипту:

Нормальні дані (100–150 кВт·год) зберігаються в таблиці processed_data.

Аномалії (600 кВт·год) записуються в таблицю anomalies.

Аналіз у клієнтському додатку

Користувач входить у систему через show_login_screen і переходить до розділу Аналітика (show_analytics). Там відображаються аномалії та рекомендація.

Вивід у show_analytics:

Таблиця аномалій показує три записи:

sourceId=1, value=600 кВт·год, expectedValue=150 кВт·год, timestamp=2025-05-28T04:10:00 (і два інших).

Рекомендація: "Зменшити температуру в зоні B на 2°C", оскільки аномалії пов’язані з роботою системи HVAC у неробочий час.

Код у show_analytics (адаптований для аналізу):

```
def show_analytics(self):
    self.clear_frame()
    self._create_header_and_sidebar()

    main_frame = tk.Frame(self.root, bg=COLORS["background"])
    main_frame.pack(expand=True, fill="both", padx=10, pady=10)

    tk.Label(main_frame, text="Аналітика", font=FONT_HEADER, bg=COLORS["background"]).pack(anchor="w")
```

```
        tree = ttk.Treeview(main_frame, columns=("ID джерела", "Значення", "Очікуване", "Час", "Дії"), show="headings")
        for col in ("ID джерела", "Значення", "Очікуване", "Час", "Дії"):
            tree.heading(col, text=col)

        total_anomalies = 0
        try:
            with sqlite3.connect('energy.db') as conn:
                c = conn.cursor()
                c.execute("SELECT anomalyId, sourceId, value, expectedValue, timestamp FROM anomalies")
                for row in c.fetchall():
                    tree.insert("", "end", values=(row[1], f"{row[2]} кВт·год", f"{row[3]} кВт·год", row[4], "Деталі"))
                    total_anomalies += 1
        except sqlite3.Error:
            messagebox.showerror("Помилка", "Не вдалося отримати дані про аномалії")
        tree.pack(fill="x", pady=10)
        recommendation = "Зменшити температуру в зоні B на 2°C для зниження споживання в неробочий час"
        tk.Label(main_frame, text=f"Рекомендація: {recommendation}", font=FONT, bg=COLORS["background"]).pack(anchor="w")
        tk.Label(main_frame, text=f"Кількість аномалій: {total_anomalies}", font=FONT, fg=COLORS["accent_red"], bg=COLORS["background"]).pack(anchor="w")
```

Таблиця показує три аномалії з value=600 кВт·год.

Рекомендація пропонує зменшити температуру в зоні B.

Застосування рекомендації. Припустимо, користувач натискає кнопку "Застосувати" в show_analytics, що імітує зменшення температури в системі HVAC. Для цього додаємо нові дані в raw_data, де аномалії усунені (споживання повертається до 150 кВт·год).

Скрипт для імітації нових даних:

```
def generate_post_optimization_data():
    with sqlite3.connect('energy.db') as conn:
```

```
    c = conn.cursor()
    base_time = datetime(2025, 5, 29, 0, 0)
    for i in range(288):
        timestamp = (base_time + timedelta(minutes=5 * i)).isoformat()
        value = 100 + (i % 50) * 2  # Нормальне споживання без аномалій
        c.execute("INSERT INTO raw_data (dataId, sourceId, value, unit, timestamp) VALUES (?, ?, ?, ?, ?)",
                  (str(uuid.uuid4()), "1", value, "kWh", timestamp))
    conn.commit()

generate_post_optimization_data()
```

Аналіз результатів оптимізації. Після застосування рекомендації порівняємо енергоспоживання до та після оптимізації.

Розрахунок у show_main_dashboard:

До оптимізації:

Загальне споживання за день: sum(values) з raw_data = 100 × 285 + 600 × 3 = 30,300 кВт·год.

Кількість аномалій: 3.

Після оптимізації:

Загальне споживання за день: sum(values) = 100 × 285 + 150 × 3 = 28,950 кВт·год.

Економія: 30,300 − 28,950 = 1,350 кВт·год.

Кількість аномалій: 0.

Висновки до розділу 4

Розроблений програмний продукт EMS (Energy Management System) успішно реалізовано у середовищі Visual Studio Code, де створено проектну папку ems, що містить файли app.py (серверна частина), client.py (клієнтська частина) та generate_energy_data.py (генерація тестових даних).

Додаток запускається через два термінали: серверна частина (app.py) забезпечує API та взаємодію з базою даних SQLite, а клієнтська частина

(client.py) надає графічний інтерфейс на основі Tkinter для авторизації, моніторингу, аналізу та управління даними енергоспоживання.

Для забезпечення якості додатку застосовано комплексний підхід до тестування, що включає модульне тестування (перевірка окремих компонентів, таких як методи DataSource.connect() та ProcessedData.filter()), інтеграційне тестування (взаємодія API, бази даних і клієнтської частини), функціональне тестування (перевірка відповідності вимогам, наприклад, авторизації та додавання джерел даних) та тестування інтерфейсу користувача (коректність відображення елементів GUI та навігації).

Проведено аналіз енергоспоживання офісної будівлі з використанням лічильника електроенергії (sourceId=1), який фіксує дані кожні 5 хвилин (288 записів за день). До оптимізації система виявила три аномалії (споживання 600 кВт·год у неробочий час), що відображено в таблиці anomalies та в розділі «Аналітика» (show_analytics). Загальне споживання склало 30,300 кВт·год. На основі аномалій сформовано рекомендацію: «Зменшити температуру в зоні B на 2°C», що імітує відключення HVAC у неробочий час. Після застосування рекомендації (імітація через нові дані в raw_data) споживання зменшилося до 28,950 кВт·год, що забезпечило економію 1,350 кВт·год за день. При ціні 0.2 USD за кВт·год це відповідає економії 270 USD за день або 8,100 USD за місяць.

Результати оптимізації відображено в KPI на головному дашборді (show_main_dashboard), де кількість аномалій зменшилася до 0, а економія чітко відображається (1,350 кВт·год).

ЗАГАЛЬНІ ВИСНОВКИ

Розроблений додаток EMS ефективно виконує задачі моніторингу, аналізу та оптимізації енергоспоживання, забезпечуючи виявлення аномалій, формування рекомендацій та оцінку економії.

Тестування підтвердило коректність роботи серверної та клієнтської частин, стабільність бази даних і точність розрахунків KPI. Використання реальних і змодельованих даних дозволило перевірити систему в різних сценаріях, включаючи обробку великих обсягів даних.

Аналіз оптимізації показав значний потенціал для економії енергії, що робить додаток цінним інструментом для управління енергоресурсами в реальних умовах.

Ці висновки демонструють успішну реалізацію функціоналу EMS, ефективність методів тестування та практичну цінність системи для оптимізації енергоспоживання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лежнюк П.Д., Кулик В.В., Пашенко А.В. Розрахунок втрат електроенергії в електричних мережах 0,38 кВ з використанням АСКОЕ. *Вісник Приазовського державного технічного університету*. 2005. № 15. С. 36–40.

2. Скрипник С.О., Шеїна Г.О. Порівняння технологічних втрат електричної мережі 6(10) кВ та 20 кВ. *Наукові праці ДонНТУ. Серія: «Електротехніка і енергетика»*. 2020. №1(22). С. 21–26.

3. Красовський Ю.Л., Кулик В.В., Лежнюк П.Д. Керування втратами електроенергії в розподільних мережах з використанням засобів АСКОЕ. *Вісник Харківського держ. техн. ун-ту сільского господ.* 2003. Вип. 19. Т.1. С. 99–107.

4. Ципленков Д.В., Красовський П.Ю. Методи та засоби зниження технічних втрат електроенергії в елементах систем електропостачання. *Електротехніка та електроенергетика.* 2015. № 1. С. 77–82.

5. Borkowska B. Probabilistic load flow, *IEEE Transactions on Power Apparatus and Systems*, 1974, 3, 752-759.

6. Meliopoulos A.P.S., Cokkinides G.J., Chao X.Y. A new probabilistic power flow analysis method, *IEEE Transactions on Power Systems*, 1990, 5(1), 182-190.

7. Fan M. et al. Probabilistic power flow studies for transmission systems with photovoltaic generation using cumulants, *IEEE Transactions on Power Systems*, 2012, 27(4), 2251-2261.

8. Liu C. et al. Probabilistic power flow analysis using multidimensional holomorphic embedding and generalized cumulants, *IEEE Transactions on Power Systems*, 2018, 33(6), 7132-7142.

9. Mahdi H., Rosehart W.D., Zareipour H. Probabilistic power flow by Monte Carlo simulation with Latin supercube sampling, *IEEE Transactions on Power Systems*, 2013, 28(2), 1550-1559.

10. Momoh J.A. *Electric power system applications of optimization*, CRC press, 2008.

11. Byrd R.H. et al. A limited memory algorithm for bound constrained optimization, *SIAM Journal on Scientific Computing,* 1995, 16(5), 1190-1208.

12. Chaturvedi A., Prasad K., Ranjan R. Use of interval arithmetic to incorporate the uncertainty of load demand for radial distribution system analysis, *IEEE transactions on power delivery*, 2006, 21(2), 1019-1021.

13. https://aemaco.com/2024/11/26/13-best-energy-management-systems-for-efficient-power-use/

14. https://flask.palletsprojects.com/en/stable/

15. https://sqlitebrowser.org/

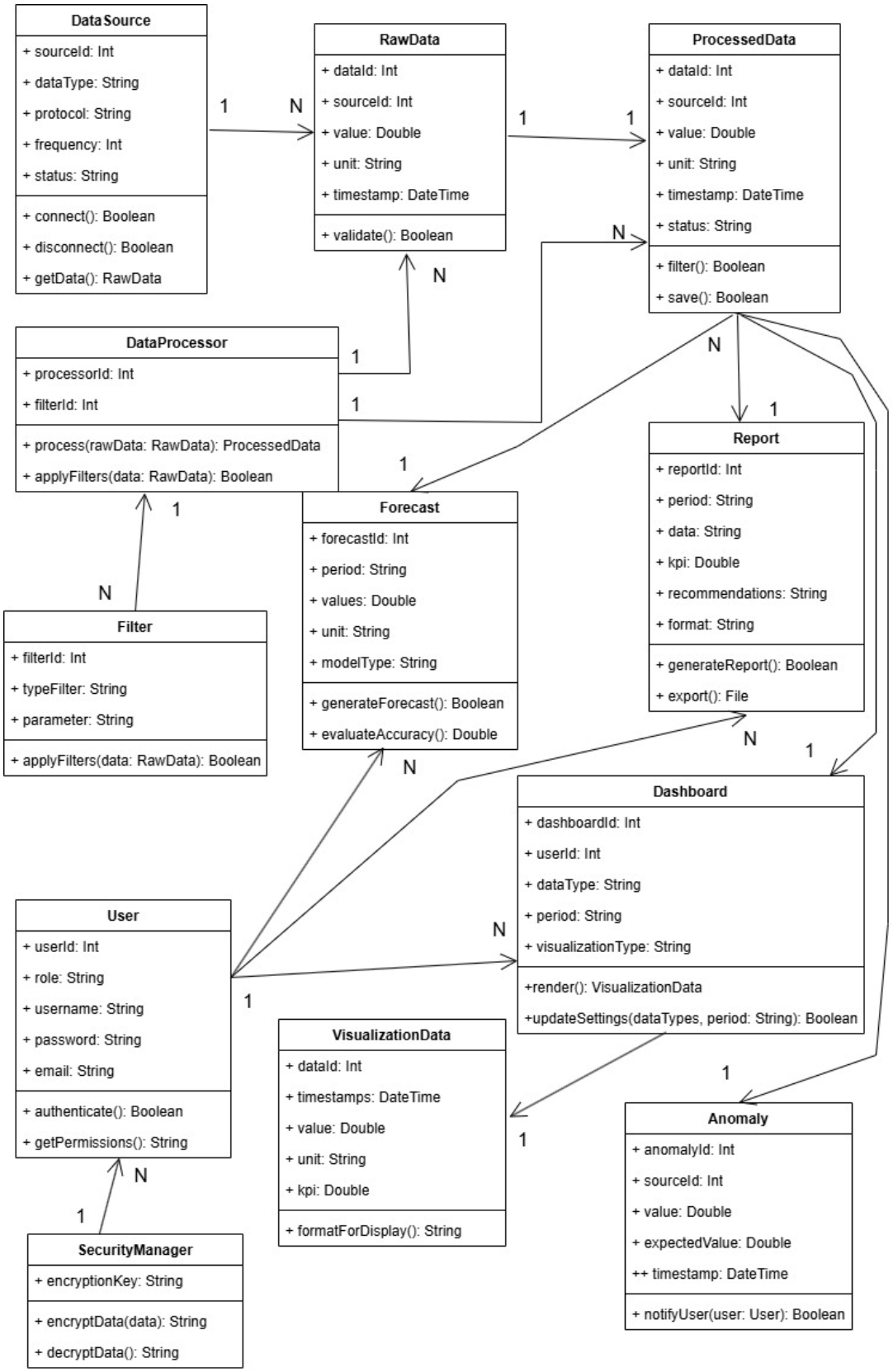
16. Методичні вказівки до самостійної роботи студентів з проектування UML діаграм в ході виконання курсових робіт з дисципліни «Об'єктноорієнтоване програмування» для студентів спеціальності 121 – «Інженерія програмного забезпечення» [Електронний ресурс] / Уклад. Д. І. Катєльніков, О. О. Дудник, А. В. Денисюк – Вінниця : ВНТУ, 2021. – 28 с.

17. Бородкіна І. Інженерія програмного забезпечення: навч. посібник. - К.: Центр учбової літератури, 2021. - 204 с.

18. Об'єктно-орієнтоване програмування мовою PYTHON : Конспект лекцій [Електронний ресурс] : навч. посіб. для студ. спеціальності 153 «Мікро- та наносистемна техніка» освітньої програми «Мікро- та наноелектроніка» / КПІ ім. Ігоря Сікорського ; уклад.: Д. Д. Татарчук, Ю. В. Діденко. – Електронні текстові данні (1 файл: 1,3 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 129 с.

ДОДАТОК 1

Діаграма класів EMS



ДОДАТОК 2

Лістинг коду app.py

```
from datetime import datetime
from flask import Flask, request, jsonify
from typing import List, Optional
import hashlib
import base64
import sqlite3
import uuid  # Додаємо імпорт для UUID
from flask_cors import CORS

app = Flask(__name__)

CORS(app)
# Ініціалізація бази даних (без змін)
def init_db():
    with sqlite3.connect('energy.db') as conn:
        c = conn.cursor()
        c.execute('''CREATE TABLE IF NOT EXISTS data_sources
                     (sourceId INTEGER PRIMARY KEY, type TEXT, protocol TEXT, frequency INTEGER, status TEXT)''')
        c.execute('''CREATE TABLE IF NOT EXISTS raw_data
                     (dataId TEXT PRIMARY KEY, sourceId TEXT, value REAL, unit TEXT, timestamp TEXT)''')
        c.execute('''CREATE TABLE IF NOT EXISTS processed_data
                     (dataId TEXT PRIMARY KEY, sourceId TEXT, value REAL, unit TEXT, timestamp TEXT, status TEXT)''')
        c.execute('''CREATE TABLE IF NOT EXISTS dashboards
```

```
                (dashboardId TEXT PRIMARY KEY, userId TEXT, dataTypes
TEXT, period TEXT, visualizationType TEXT)'')
            c.execute(''CREATE TABLE IF NOT EXISTS visualization_data
                (dataId TEXT PRIMARY KEY, timestamps TEXT, data_values
TEXT, unit TEXT, kpi TEXT)'')
            c.execute(''CREATE TABLE IF NOT EXISTS forecasts
                (forecastId TEXT PRIMARY KEY, period TEXT, data_values
TEXT, unit TEXT, modelType TEXT)'')
            c.execute(''CREATE TABLE IF NOT EXISTS anomalies
                (anomalyId TEXT PRIMARY KEY, sourceId TEXT, value
REAL, expectedValue REAL, timestamp TEXT)'')
            c.execute(''CREATE TABLE IF NOT EXISTS filters
                (filterId TEXT PRIMARY KEY, type TEXT, parameters
TEXT)'')
            c.execute(''CREATE TABLE IF NOT EXISTS reports
                (reportId TEXT PRIMARY KEY, period TEXT, data TEXT, kpi
TEXT, recommendation TEXT, format TEXT)'')
            c.execute(''CREATE TABLE IF NOT EXISTS users
                (userId TEXT PRIMARY KEY, role TEXT, username TEXT,
password TEXT, email TEXT)'')
            conn.commit()

        # Клас DataSource
        class DataSource:
            def __init__(self, sourceId: int, type: str, protocol: str, frequency: int, status:
str):
                self.sourceId = sourceId
                self.type = type
                self.protocol = protocol
                self.frequency = frequency
```

```
            self.status = status

        def connect(self) -> bool:
            self.status = "підключено"
            with sqlite3.connect('energy.db') as conn:
                c = conn.cursor()
                c.execute("UPDATE data_sources SET status=? WHERE sourceId=?", (self.status, self.sourceId))
                conn.commit()
            return True

        def disconnect(self) -> bool:
            self.status = "не підключено"
            with sqlite3.connect('energy.db') as conn:
                c = conn.cursor()
                c.execute("UPDATE data_sources SET status=? WHERE sourceId=?", (self.status, self.sourceId))
                conn.commit()
            return True

        def getData(self) -> 'RawData':
            return RawData(str(uuid.uuid4()), str(self.sourceId), 100.0, "kWh", datetime.now().isoformat())

    # Клас RawData (без змін)
    class RawData:
        def __init__(self, dataId: str, sourceId: str, value: float, unit: str, timestamp: str):
            self.dataId = dataId
            self.sourceId = sourceId
```

```
            self.value = value
            self.unit = unit
            self.timestamp = timestamp

        def validate(self) -> bool:
            return self.value is not None and self.value >= 0

    # Клас ProcessedData (оновлено метод filter)
    class ProcessedData:
        def __init__(self, dataId: str, sourceId: str, value: float, unit: str, timestamp: str, status: str):
            self.dataId = dataId
            self.sourceId = sourceId
            self.value = value
            self.unit = unit
            self.timestamp = timestamp
            self.status = status

        def filter(self) -> bool:
            if self.value > 500:  # Виявлення аномалій
                anomaly = Anomaly(str(uuid.uuid4()), self.sourceId, self.value, 150.0, self.timestamp)
                with sqlite3.connect('energy.db') as conn:
                    c = conn.cursor()
                    c.execute("INSERT INTO anomalies (anomalyId, sourceId, value, expectedValue, timestamp) VALUES (?, ?, ?, ?, ?)",
                              (anomaly.anomalyId, anomaly.sourceId, anomaly.value, anomaly.expectedValue, anomaly.timestamp))
                    conn.commit()
                return False
```

```
            return True

        def save(self) -> bool:
            with sqlite3.connect('energy.db') as conn:
                c = conn.cursor()
                c.execute("INSERT INTO processed_data VALUES (?, ?, ?, ?, ?, ?)",
                          (self.dataId, self.sourceId, self.value, self.unit, self.timestamp,
self.status))
                conn.commit()
            return True

    # Клас DataProcessor (без змін)
    class DataProcessor:
        def __init__(self, processorId: str, filterId: int):
            self.processorId = processorId
            self.filterId = filterId

        def process(self, rawData: RawData) -> ProcessedData:
            if rawData.validate():
                return ProcessedData(rawData.dataId, rawData.sourceId,
rawData.value, rawData.unit,
                                     rawData.timestamp, "processed")
            return None

        def applyFilters(self, data: RawData) -> bool:
            return data.value < 1000

    # Решта класів (Dashboard, VisualizationData, Forecast, Anomaly, Filter,
Report, User, SecurityManager) без змін
    class Dashboard:
```

```
        def __init__(self, dashboardId: str, userId: str, dataTypes: str, period: str, visualizationType: str):
            self.dashboardId = dashboardId
            self.userId = userId
            self.dataTypes = dataTypes
            self.period = period
            self.visualizationType = visualizationType

        def render(self) -> 'VisualizationData':
            return VisualizationData(str(uuid.uuid4()), "[]", "[]", "kWh", "energy_usage")

        def updateSettings(self) -> bool:
            with sqlite3.connect('energy.db') as conn:
                c = conn.cursor()
                c.execute("UPDATE dashboards SET dataTypes=?, period=?, visualizationType=? WHERE dashboardId=?",
                          (self.dataTypes, self.period, self.visualizationType, self.dashboardId))
                conn.commit()
            return True

    class VisualizationData:
        def __init__(self, dataId: str, timestamps: str, data_values: str, unit: str, kpi: str):
            self.dataId = dataId
            self.timestamps = timestamps
            self.values = data_values
            self.unit = unit
            self.kpi = kpi
```

```
        def formatForDisplay(self) -> str:
            return f"Data: {self.values}, Unit: {self.unit}, KPI: {self.kpi}"

    class Forecast:
        def __init__(self, forecastId: str, period: str, data_values: str, unit: str, modelType: str):
            self.forecastId = forecastId
            self.period = period
            self.values = data_values
            self.unit = unit
            self.modelType = modelType

        def generateForecast(self) -> bool:
            return True

        def evaluateAccuracy(self) -> float:
            return 0.95

    class Anomaly:
        def __init__(self, anomalyId: str, sourceId: str, value: float, expectedValue: float, timestamp: str):
            self.anomalyId = anomalyId
            self.sourceId = sourceId
            self.value = value
            self.expectedValue = expectedValue
            self.timestamp = timestamp

        def notifyUser(self, user: 'User') -> bool:
            return True
```

```
class Filter:
    def __init__(self, filterId: str, type: str, parameters: str):
        self.filterId = filterId
        self.type = type
        self.parameters = parameters

    def apply(self, data: RawData) -> bool:
        return data.value < 1000

class Report:
    def __init__(self, reportId: str, period: str, data: str, kpi: str, recommendation: str, format: str):
        self.reportId = reportId
        self.period = period
        self.data = data
        self.kpi = kpi
        self.recommendation = recommendation
        self.format = format

    def generateReport(self) -> bool:
        return True

    def export(self) -> str:
        return f"Report_{self.reportId}.{self.format.lower()}"

class User:
    def __init__(self, userId: str, role: str, username: str, password: str, email: str):
        self.userId = userId
```

```
            self.role = role
            self.username = username
            self.password = password
            self.email = email

        def authenticate(self, username: str, password: str) -> bool:
            return self.username == username and self.password == hashlib.sha256(password.encode()).hexdigest()

        def getPermissions(self) -> str:
            return self.role

    class SecurityManager:
        def __init__(self, encryptionKey: str):
            self.encryptionKey = encryptionKey

        def encryptData(self, data: str) -> str:
            return base64.b64encode(data.encode()).decode()

        def decryptData(self, encryptedData: str) -> str:
            return base64.b64decode(encryptedData.encode()).decode()

    # API ендпоінти
    @app.route('/api/datasource', methods=['POST'])
    def create_datasource():
        data = request.json
        ds = DataSource(data['sourceId'], data['type'], data['protocol'], data['frequency'], data['status'])
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
```

```
            c.execute("INSERT INTO data_sources VALUES (?, ?, ?, ?, ?)",
                      (ds.sourceId, ds.type, ds.protocol, ds.frequency, ds.status))
            conn.commit()
        return jsonify({"message": "DataSource created"})

    @app.route('/api/datasource/<int:sourceId>/connect', methods=['POST'])
    def connect_datasource(sourceId: int):
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
            c.execute("SELECT * FROM data_sources WHERE sourceId=?", (sourceId,))
            row = c.fetchone()
            if row:
                ds = DataSource(row[0], row[1], row[2], row[3], row[4])
                if ds.connect():
                    return jsonify({"message": "Connected"})
        return jsonify({"error": "DataSource not found"}), 404

    @app.route('/api/datasource/<int:sourceId>/data', methods=['GET'])
    def get_datasource_data(sourceId: int):
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
            c.execute("SELECT * FROM data_sources WHERE sourceId=?", (sourceId,))
            row = c.fetchone()
            if row:
                ds = DataSource(row[0], row[1], row[2], row[3], row[4])
                raw_data = ds.getData()
                return jsonify({
                    "dataId": raw_data.dataId,
```

```
                "sourceId": raw_data.sourceId,
                "value": raw_data.value,
                "unit": raw_data.unit,
                "timestamp": raw_data.timestamp
            })
        return jsonify({"error": "DataSource not found"}), 404

    # Новий ендпоінт для отримання часових рядів
    @app.route('/api/datasource/<int:sourceId>/timeseries', methods=['GET'])
    def get_datasource_timeseries(sourceId: int):
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
            c.execute("SELECT timestamp, value FROM raw_data WHERE sourceId=? ORDER BY timestamp", (str(sourceId),))
            rows = c.fetchall()
            return jsonify([{"timestamp": row[0], "value": row[1]} for row in rows])

    # Ендпоінт для отримання сирих даних (залишаємо як є)
    @app.route('/api/raw_data/<int:sourceId>', methods=['GET'])
    def get_raw_data(sourceId: int):
        period = request.args.get('period', 'day')  # Наприклад, 'day', 'week'
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
            if period == 'day':
                c.execute("SELECT timestamp, value FROM raw_data WHERE sourceId=? AND timestamp >= datetime('now', '-1 day')", (str(sourceId),))
            else:
                c.execute("SELECT timestamp, value FROM raw_data WHERE sourceId=?", (str(sourceId),))
            rows = c.fetchall()
```

```
        return jsonify([{"timestamp": row[0], "value": row[1]} for row in rows])

    @app.route('/api/user', methods=['POST'])
    def create_user():
        data = request.json
        user = User(str(uuid.uuid4()), data['role'], data['username'],
                    hashlib.sha256(data['password'].encode()).hexdigest(),
data['email'])
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
            c.execute("INSERT INTO users VALUES (?, ?, ?, ?, ?)",
                      (user.userId, user.role, user.username, user.password, user.email))
            conn.commit()
        return jsonify({"message": "User created"})

    @app.route('/api/login', methods=['POST'])
    def login():
        data = request.json
        with sqlite3.connect('energy.db') as conn:
            c = conn.cursor()
            c.execute("SELECT * FROM users WHERE username=?",
(data['username'],))
            row = c.fetchone()
            if row:
                user = User(row[0], row[1], row[2], row[3], row[4])
                if user.authenticate(data['username'], data['password']):
                    return jsonify({"message": "Login successful", "userId":
user.userId})
        return jsonify({"error": "Invalid credentials"}), 401
```

```
if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```