НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи моніторингу та оптимізації енергоспоживання в «розумному» будинку»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент **_____ Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент **_____ Дмитро НІКОЛАЄНКО**
(підпис)

Виконав **_____ Михайло НІКОЛАЄНКО**
(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Ніколаєнку Михайлу Станіславовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення системи моніторингу та оптимізації енергоспоживання в «розумному» будинку»
затверджена наказом від «19» грудня 2025 р. № 3196 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: Вимоги до розробки мобільного додатку для моніторингу та оптимізації енергоспоживання в «розумному» будинку.
Перелік питань, що підлягають дослідженню:
1. Системний аналіз предметної області моніторингу та оптимізації енергоспоживання в «розумному» будинку
2.Проєктування інформаційного забезпечення та архітектури програмної системи
3. Розробка програмного забезпечення системи
4. Рекомендації щодо впровадження та експлуатації системи
Перелік графічних документів (за необхідності).
_____
Дата видачі завдання «____» ___грудня_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Дмитро НІКОЛАЄНКО**
(підпис)

Завдання взяв до виконання _____ **Михайло НІКОЛАЄНКО**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс

APK (Android Package Kit) – формат інсталяційного пакета операційної системи Android

CRUD (Create, Read, Update, Delete) – базові операції створення, читання, оновлення та видалення даних

ER (Entity-Relationship) – модель «сутність–зв'язок»

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі гіпертексту

IoT (Internet of Things) – Інтернет речей

JSON (JavaScript Object Notation) – текстовий формат обміну даними на основі JavaScript

JWT (JSON Web Token) – веб-токен на основі формату JSON

$LiFePO_4$ – літій-залізо-фосфат, тип хімії акумуляторної батареї

LTS (Long Term Support) – версія програмного забезпечення з довготривалою підтримкою

MQTT (Message Queuing Telemetry Transport) – протокол телеметрії для систем Інтернету речей

OAuth (Open Authorization) – відкритий протокол авторизації

ORM (Object-Relational Mapping) – об'єктно-реляційне відображення

REST (Representational State Transfer) – архітектурний стиль побудови програмних інтерфейсів

SDK (Software Development Kit) – набір засобів розробки програмного забезпечення

SoC (State of Charge) – рівень заряду акумуляторної батареї

SQL (Structured Query Language) – мова структурованих запитів

SSD (Solid-State Drive) – твердотільний накопичувач

TLS (Transport Layer Security) – протокол захисту транспортного рівня

UML (Unified Modeling Language) – уніфікована мова моделювання

URL (Uniform Resource Locator) – уніфікований локатор ресурсу

YAML (YAML Ain't Markup Language) – мова серіалізації даних для конфігураційних файлів

ДСТУ – Державний стандарт України

НЕК – Національна енергетична компанія

СУБД – система управління базами даних

СУІБ – система управління інформаційною безпекою

ВСТУП

Актуальність. Починаючи з осені 2022 року, оператор системи передачі НЕК «Укренерго» [1] запровадив погодинні графіки обмежень електропостачання за шістьма чергами споживачів. У дефіцитні періоди тривалість відсутності зовнішнього живлення сягає 12–16 годин на добу, що супроводжується аномаліями якості напруги – просіданнями нижче 200 В та перенапругами понад 250 В [2]. Реакцією домогосподарств на ці умови стало масове впровадження гібридних інверторів з акумуляторними батареями типу $LiFePO_4$ [3]. Водночас саме по собі обладнання не розв'язує двох критичних експлуатаційних проблем.

Перша проблема – пікове перевантаження при відновленні живлення. Одночасний пуск побутових приладів з компресорними двигунами (холодильник, кондиціонер, бойлер) спричиняє сумарний пусковий струм, що в 3–5 разів перевищує номінальний і виходить за межі пікової потужності побутового інвертора (~6 кВт). Це призводить до спрацювання захисних автоматів або аварійного відключення інвертора. Друга проблема – відсутність програмного захисту від неякісної напруги. Стандартні автоматичні вимикачі реагують на струм, а не на амплітуду напруги, тому чутлива електроніка залишається незахищеною від «брудної» напруги у перехідних режимах мережі. Сукупність зазначених факторів обумовлює потребу у програмному забезпеченні, здатному координувати порядок увімкнення приладів, контролювати витрачання ресурсу акумуляторної батареї та враховувати графік планових відключень у автоматичному режимі.

Мета розробки полягає у створенні програмного забезпечення для моніторингу та оптимізації споживання електричної енергії в «розумному» будинку. Розроблювана система не прив'язана до конкретного виробника обладнання, що відрізняє її від фірмових мобільних додатків таких компаній, як EcoFlow або Solax. Система враховує специфіку українського ринку – графіки

планових відключень за чергами споживачів, особливості якості напруги, типові сценарії експлуатації побутових приладів – та надає користувачеві інтерфейс для повсякденної взаємодії без потреби в інженерній кваліфікації.

Аналіз наявних рішень (підрозділ 1.4) виявив характерну поляризацію ринку: універсальні платформи «розумного» будинку (Tuya Smart [25], Home Assistant [5]) забезпечують керування побутовими пристроями, але не оперують даними інвертора та акумуляторної батареї; фірмові додатки виробників енергетичного обладнання (EcoFlow App, Solax Cloud) надають телеметрію установки, але не керують побутовими навантаженнями за межами власної екосистеми. Розроблюване програмне забезпечення EnergyHub поєднує обидві функціональні групи в межах єдиної апаратно-незалежної архітектури.

Об'єкт дослідження – процес моніторингу та управління енергоспоживанням у «розумному» будинку з гібридним електропостачанням.

Предмет дослідження – методи та програмні засоби моніторингу й оптимізації енергоспоживання в «розумному» будинку.

Методи та технології. Серверну частину системи реалізовано мовою програмування JavaScript на платформі Node.js [6] з використанням фреймворку Express [7] для побудови REST API. Для роботи з базою даних застосовано бібліотеку Prisma [8] – ORM, що забезпечує типобезпечний доступ до даних та автоматизує управління схемою бази через систему міграцій. Як систему управління базами даних обрано PostgreSQL [9] – реляційну СУБД з відкритим кодом, що підтримує складні індексні структури для швидкої вибірки часових рядів телеметрії.

Для обміну телеметрією між симулятором гібридного інвертора та серверною частиною застосовано протокол MQTT [10] з брокером Eclipse Mosquitto [11]. Цей протокол є галузевим стандартом для систем Інтернету речей завдяки низьким накладним витратам на трафік та підтримці моделі «публікація–підписка». Розгортання серверних компонентів виконано в контейнерах Docker [12], що забезпечує ідентичні умови роботи в середовищі розробки та цільовому середовищі експлуатації.

Клієнтський додаток розроблено на основі фреймворку React Native [13] у середовищі Expo [14], що дозволило отримати єдину кодову базу для мобільних платформ Android та iOS. Для побудови історичних графіків використано бібліотеку react-native-chart-kit, для організації навігації між екранами – бібліотеку react-navigation. Зазначена комбінація технологій дала змогу зосередити зусилля на реалізації бізнес-логіки системи без потреби у роздільній розробці нативних додатків для двох операційних систем.

Структура записки. Пояснювальна записка складається зі вступу, чотирьох основних розділів та висновків. До роботи додається список використаних джерел та додатки. Загальний обсяг записки – 69 сторінок основного тексту, кількість використаних джерел – 30, кількість додатків – 5.

У першому розділі «Системний аналіз предметної області» розглянуто особливості функціонування гібридних енергетичних установок, проаналізовано вимоги до програмних систем відповідного класу, змодельовано основні процеси за допомогою діаграм прецедентів, послідовності та активності, проведено огляд наявних на ринку рішень та сформульовано завдання на розробку.

Другий розділ «Проєктування програмного забезпечення» присвячено побудові логічної моделі даних у вигляді ER-діаграми, обґрунтуванню її відповідності третій нормальній формі, розробці діаграм класів, пакетів та компонентів, що формалізують архітектуру програмної системи.

Третій розділ «Розробка програмного забезпечення» описує процес реалізації програмної частини системи: вибір інструментарію розробки, створення інформаційної бази, побудову системи управління інформаційною безпекою та алгоритмізацію ключових модулів захисної автоматики.

У четвертому розділі «Рекомендації щодо впровадження та експлуатації системи» подано результати функціонального тестування, визначено вимоги до апаратного та програмного середовища, описано склад інсталяційного пакета та порядок розгортання.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметною областю кваліфікаційної роботи є моніторинг та оптимізація споживання електричної енергії в умовах «розумного» будинку, обладнаного гібридним інвертором з акумуляторною батареєю на елементах літій-залізо-фосфат ($LiFePO_4$ [3]). Гібридна схема передбачає два джерела живлення: зовнішню мережу змінного струму та акумуляторну батарею, між якими інвертор перемикається автоматично. Об'єктом автоматизації виступають побутові електроприлади різних класів критичності, підключені до виходу інвертора через комутаційні пристрої (розумні розетки, реле з підтримкою дистанційного керування). Поняття «розумного» будинку в контексті роботи означає інтелектуальне управління енергоресурсами: система автономно визначає, які прилади вимкнути при низькому заряді батареї, яким чином безпечно відновити живлення після блекауту та, за наявності графіка планових відключень, виконує превентивне перемикання на батарею, що є складовою ширшої концепції інтелектуальних електричних мереж [15; 16]

Стан національної енергоінфраструктури формує специфічні вимоги до програмного забезпечення. З осені 2022 року НЕК «Укренерго» [1] запровадила погодинні графіки обмежень за шістьма чергами споживачів (для м. Києва – DTEK Yasno [17]). У дефіцитні періоди живлення відсутнє 12–16 годин на добу. Окремо фіксуються аномалії якості напруги: просідання нижче 200 В, перенапруга понад 250 В, кидки фаз при перемиканнях. Відповідно до EN 50160 [2], допустимий діапазон напруги для побутових мереж становить ±10 % від номіналу 230 В. Стандартні автоматичні вимикачі реагують на струмове перевантаження та коротке замикання, але не контролюють амплітуду напруги, що обумовлює потребу у програмному захисті чутливого обладнання.

Побутові гібридні інвертори (Qoltec MaxFlow, Deye SUN-6K) мають пікову потужність ~6 кВт. Пусковий струм компресорних двигунів холодильника або

кондиціонера перевищує номінальний у 3–5 разів. При груповому старті після блекауту сума пускових потужностей значно перевищує ліміт інвертора, що спричиняє аварійне відключення або спрацювання захисту. Для запобігання цій ситуації система має контролювати порядок увімкнення приладів із керованою затримкою між пусками – далі в роботі цей механізм позначається терміном «м'який старт».

Циклічний ресурс $LiFePO_4$-батареї залежить від глибини розряду: що глибшим є регулярне розрядження, тим швидше знижується ємність елементів [18]. Водночас повне знеструмлення є недопустимим для критичних пристроїв (освітлення, Wi-Fi маршрутизатор, охоронна сигналізація), тому система має балансувати між збереженням ресурсу батареї та підтриманням живлення пріоритетних навантажень. Збереження ресурсу досягається через каскадне відключення навантажень за класом критичності: другорядні прилади вимикаються першими, важливі – при подальшому зниженні заряду, критичні – залишаються активними до мінімально допустимого рівня.

Зазначені фізичні обмеження обладнання – пікова потужність інвертора, циклічний ресурс батареї, пускові режими компресорних двигунів – у поєднанні з нестабільністю зовнішнього електропостачання формують комплекс вимог, яким має відповідати програмне забезпечення для керування енергоспоживанням «розумного» будинку. Ступінь відповідності наявних на ринку програмних рішень зазначеним вимогам розглядається у підрозділі 1.4.

1.2 Аналіз вимог до програмної системи

Чинники, що формують вимоги, поділяються на три групи: фізичні обмеження обладнання (пікова потужність інвертора, ресурс батареї, пускові режими двигунів), особливості нестабільної електромережі (планові відключення, просідання та перенапруги) та очікування користувача (простота налаштування, прозорість стану системи, можливість створення сценаріїв).

За результатами аналізу зазначених чинників формується перелік

функціональних та нефункціональних вимог до програмного забезпечення. Функціональні вимоги для зручності розгляду згруповані у три тематичні блоки: моніторинг енергетичного стану, керування побутовими навантаженнями та оптимізація споживання.

Моніторинг енергетичного стану. Система збирає телеметрію від інвертора (потужність, SoC, напруга, джерело живлення), зберігає історію з часовою міткою для побудови графіків. До цього блоку належать наступні вимоги:

- Збір телеметрії від енергетичного обладнання. Система отримує від гібридного інвертора показники миттєвого споживання потужності, рівня заряду акумуляторної батареї, напруги змінного струму на стороні навантажень та поточного джерела живлення (мережа або батарея). Періодичність надходження забезпечує своєчасну реакцію на зміну стану.
- Зберігання історії спостережень. Кожне вимірювання зберігається у внутрішній базі даних з часовою міткою. Накопичена історія використовується для побудови графіків споживання у різних масштабах часу протягом дня.
- Відображення важливих подій на дашборді. Система відображає в інтерфейсі мобільного додатка інформацію про значущі події: перехід на батарею, зниження заряду нижче порогів каскадного відключення, автоматичне відключення пристроїв. Журнал подій на сервері фіксує всі дії захисної автоматики з часовими мітками.

Керування побутовими навантаженнями включає: класифікацію стану мережі (норма, низька, перенапруга, відсутність); каскадне відключення за SoC з трьома класами важливості (другорядні, важливі, критичні); «м'який старт» з повторною перевіркою напруги; захист від перевищення піка інвертора; ручне керування з підпорядкуванням захисним правилам; користувацькі сценарії (тригери: час доби, режим присутності).

- Класифікація стану зовнішньої електромережі. Система оперує дискретними станами, придатними для використання в умовах правил автоматизації:
    - нормальний режим – напруга перебуває в допустимому діапазоні якості;
    - низька(«брудна» напруга) – значення нижче межі норми, що становить ризик для імпульсних блоків живлення електронної техніки;
    - перенапруга – значення вище верхньої межі норми, типове в момент відновлення мережі після відключення;
    - відсутність живлення – повна втрата зовнішньої напруги внаслідок планового або аварійного відключення.
- Каскадне відключення навантажень за рівнем заряду батареї. У режимі автономної роботи система автоматично знижує навантаження за заздалегідь визначеним пріоритетом, що відповідає трьом класам критичності пристроїв:
    - другорядні – побутові прилади, відключення яких не порушує комфорт мешканців (праска, посудомийна машина, електрочайник);
    - важливі – прилади, тимчасове відключення яких допустиме, але небажане (пральна машина, духова шафа, кондиціонер);
    - критичні – прилади, відключення яких недопустиме (Wi-Fi, освітлення, сигналізація, холодильник);
- Превентивна реакція на графік планових відключень. Система періодично отримує добовий розклад планових відключень від постачальника електроенергії та використовує цю інформацію для попереднього зниження навантаження на батарею. Заздалегідь до прогнозованого моменту відключення другорядні споживачі вимикаються, що зменшує миттєвий стрибок навантаження в момент перемикання джерела живлення.

Рекомендації користувачу щодо оптимізації споживання. На основі статистики споживання за тривалі проміжки часу система формує рекомендації – наприклад, перенесення часу запуску високопотужних приладів на нічний тариф, перерозподіл пристроїв між класами критичності, виявлення приладів з аномально високим фоновим споживанням.

- Апаратна незалежність (hardware-agnostic). Архітектура системи повинна забезпечувати ізоляцію джерела телеметрії та каналу керування пристроями від ядра бізнес-логіки. Заміна моделі гібридного інвертора або переведення системи на іншу екосистему розумних розеток повинні вимагати лише розробки окремого програмного адаптера, без зміни логіки автоматичних правил та користувацького інтерфейсу. Виконання цієї вимоги відрізняє академічні та open-source рішення від фірмових мобільних додатків виробників обладнання, що працюють виключно зі своєю лінійкою пристроїв.
- Прогнозування ресурсу акумуляторної батареї. На основі історії циклів заряд-розряд та поточної глибини розряду система оцінює залишковий ресурс батареї в циклах та орієнтовно – в роках експлуатації за поточного режиму використання. Прогноз дозволяє користувачеві своєчасно планувати заміну елементів та коригувати порогові значення каскадного відключення.
- Швидкодія реагування на критичні події. Час між фіксацією нестабільної напруги в потоці телеметрії та виконанням захисної дії повинен бути мінімальним. Затримка визначається передусім періодом опитування телеметрії інвертора та особливостями каналу передачі команд керування.
- Кросплатформеність клієнтського додатка. Користувацький інтерфейс має бути доступним з мобільних платформ Android та iOS на основі єдиної кодової бази для скорочення витрат на супровід та забезпечення однакового користувацького досвіду на обох операційних системах.

1.3 Моделювання предметної області

Для формального опису предметної області побудовано три моделі, що розкривають об’єкт дослідження з різних ракурсів з використанням нотацій UML [19]. Діаграма прецедентів описує, які дії виконують зовнішні актори щодо системи. Діаграма послідовності фіксує часовий порядок взаємодії компонентів при настанні типової події. Діаграма активності відображає логічну структуру алгоритму прийняття рішень при зміні стану зовнішнього живлення.

1.3.1 Діаграма прецедентів

Діаграма прецедентів (use case) є засобом формалізації функціональних вимог через перелік дій, які зовнішні актори виконують у системі. Для розглядуваного класу програмного забезпечення виділено двох ключових акторів. Перший актор – Користувач, який представляє людину-власника гібридної енергоустановки, що взаємодіє з системою через клієнтський додаток. Другий актор – Автоматична підсистема керування (далі – підсистема керування), що моделює внутрішнього програмного агента, відповідального за виконання захисних та оптимізаційних правил без участі людини. Включення підсистеми керування як окремого актора є типовою практикою для систем, у яких значна частина дій виконується автоматично за тригерами зі сторони обладнання або календарних подій, а не за командою користувача.

До прецедентів Користувача належать:

- **Перегляд поточного стану** – отримання зведеної інформації про активне джерело живлення, рівень заряду батареї, миттєву потужність, стан кожного підключеного пристрою (дашборд клієнтського додатка).
- **Налаштування пристроїв** – задання параметрів окремого побутового приладу: класу критичності, номінальної потужності, індивідуальної затримки м’якого старту.
- **Керування сценаріями** – створення, редагування та видалення власних правил автоматизації, заснованих на часі доби або режимі присутності.

- **Перегляд графіка відключень** – ознайомлення з добовим розкладом планових відключень для відповідної черги споживача.

  До прецедентів підсистеми керування належать:

- **Каскадне відключення за SoC** – автоматичне знеструмлення побутових навантажень за пріоритетом критичності при подоланні порогових значень рівня заряду батареї.
- **«м'який старт»** – послідовне вмикання раніше відключених пристроїв із керованою індивідуальною затримкою при відновленні якісної напруги в зовнішній мережі.
- **Превентивна реакція на графік** – попереднє відключення некритичних споживачів за визначений час до прогнозованого моменту планового відключення зовнішнього живлення.
- **Виконання сценарію по тригеру** – спрацювання користувацького правила автоматизації при настанні відповідної умови (часу доби або зміни режиму присутності).

Графічне представлення діаграми прецедентів подано на рис. 1.1.

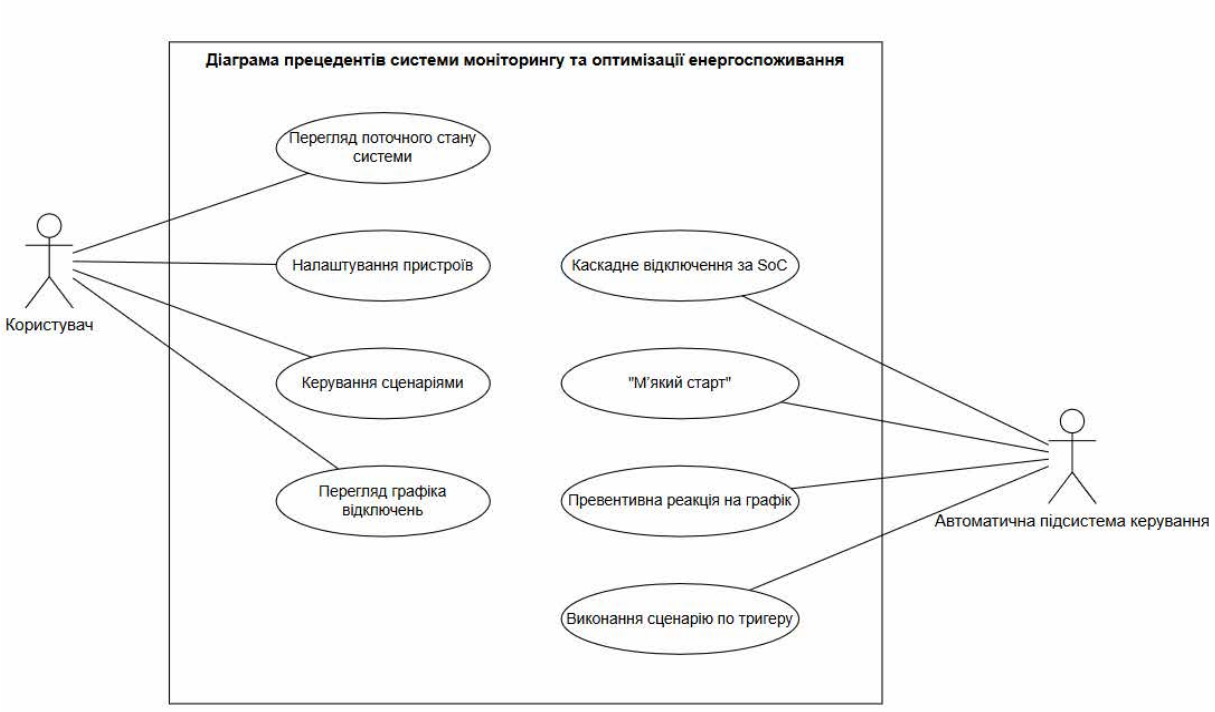


Рис. 1.1 – Діаграма прецедентів системи моніторингу та оптимізації енергоспоживання

1.3.2 Діаграма послідовності

Діаграма послідовності демонструє часовий порядок взаємодії учасників системи при настанні типової події – у цьому випадку при виявленні критичного рівня заряду батареї в режимі автономного живлення. Сценарій починається з дії користувача (встановлення порогу спрацювання каскадного відключення) та завершується відображенням нового стану системи в клієнтському додатку. У взаємодії беруть участь сім учасників: Користувач, Мобільний додаток, Оркестратор (логіка системи керування), Entity Object (сутність зберігання конфігурації), API джерела живлення, MQTT-брокер та API пристроїв.

Послідовність кроків наступна. Користувач задає в інтерфейсі мобільного додатка пороговий рівень заряду батареї для активації каскадного відключення. Мобільний додаток передає ці налаштування оркестратору, який зберігає їх у відповідній сутності та повертає підтвердження. Періодично оркестратор виконує опитування джерела живлення для отримання поточного значення SoC та активного джерела. У момент, коли отримане значення опускається нижче встановленого порогу, оркестратор формує запит на отримання списку пристроїв низького пріоритету. Для кожного такого пристрою публікується MQTT-команда вимикання, яка через брокер досягає API пристроїв та виконується на стороні розумної розетки. Після отримання підтвердження виконання оркестратор обмежує навантаження, яке відображається в інтерфейсі мобільного додатка. Користувач бачить оновлений статус системи з переліком автоматично відключених пристроїв та поточним рівнем заряду.

Графічне представлення діаграми послідовності подано на рис. 1.2.

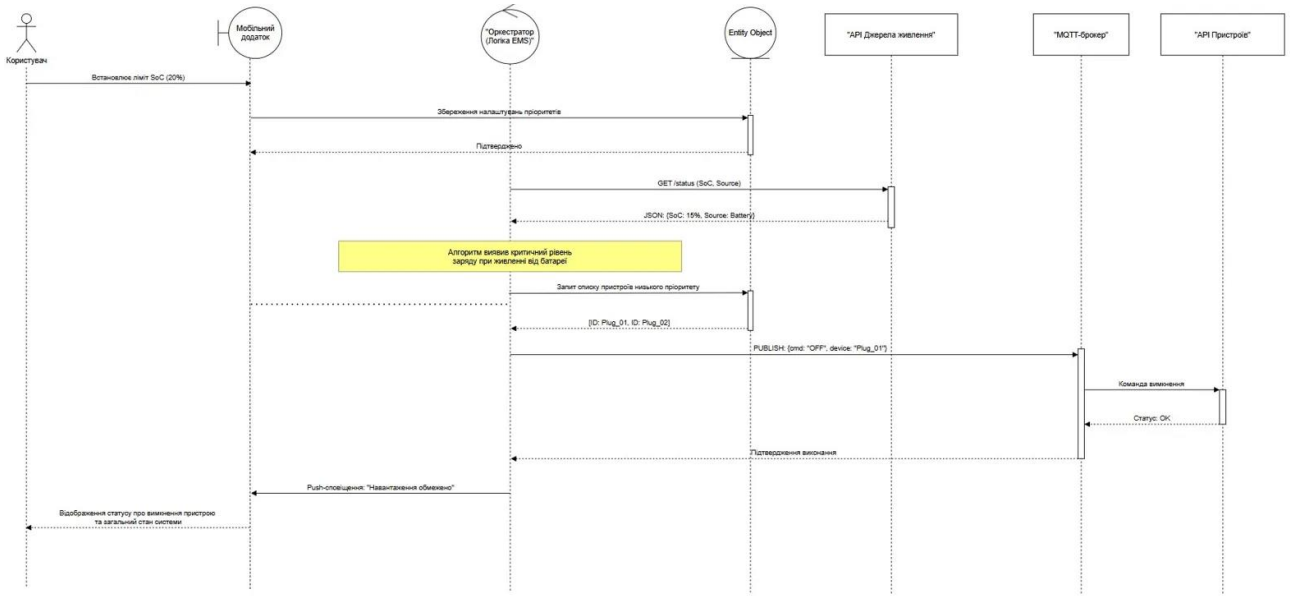


Рис. 1.2 – Діаграма послідовності реакції системи на критичний рівень заряду батареї

1.3.3 Діаграма активності

Діаграма активності формалізує алгоритм прийняття рішень підсистемою керування при настанні події в зовнішній електромережі. На відміну від діаграми послідовності, що оперує учасниками та повідомленнями, діаграма активності відображає послідовність дій та точки розгалуження логіки за умовами. Для моделювання обрано сценарій «Реакція системи на відключення зовнішньої мережі» – як один з найбільш характерних для предметної області.

Сценарій починається з фіксації події в потоці телеметрії – зникнення напруги на вводі з’єднання з зовнішньою мережею. Підсистема керування виконує запит до джерела живлення для уточнення поточного джерела (мережа чи батарея) та рівня заряду батареї. Якщо мережа працює, але напруга нестабільна, активується відповідна гілка обробки нестабільної напруги. Якщо мережа повністю відсутня, перевіряється рівень заряду батареї. За достатнього рівня заряду виконується автоматичне переведення системи в режим автономного живлення зі збереженням усіх активних навантажень. За низького рівня заряду активується каскадне відключення за класами критичності – відключаються спочатку другорядні, потім важливі споживачі, до залишення

лише критичних. У будь-якому з варіантів завершення оновлюється стан системи на дашборді мобільного додатка.

Графічне представлення діаграми активності подано на рис. 1.3.

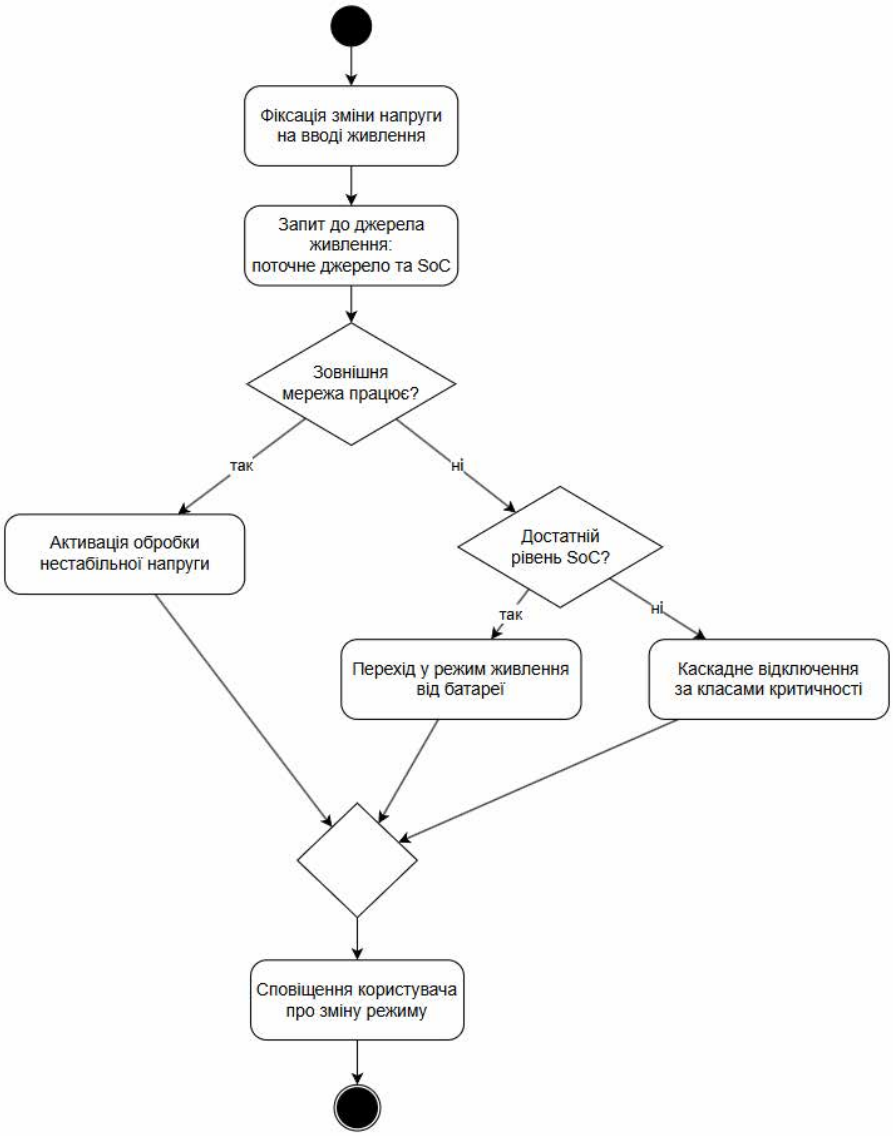


Рис. 1.3 – Діаграма активності реакції системи на відключення зовнішньої мережі

Сукупність трьох побудованих моделей формує мінімально необхідне уявлення про функціональну структуру предметної області. Діаграма прецедентів встановлює межі взаємодії з зовнішніми акторами, діаграма послідовності демонструє типовий часовий перебіг подій у системі, діаграма активності розкриває внутрішню логіку прийняття рішень. У підрозділі 1.4

виконується перевірка наявних на ринку рішень на предмет реалізації описаних взаємодій та логіки.

1.4 Огляд джерел та існуючих рішень

Огляд наявних на ринку програмних продуктів виконується для встановлення відповідності кожного представника класу систем переліку функціональних та нефункціональних вимог, сформульованих у підрозділі 1.2. До розгляду обрано чотири рішення, що покривають основні підкласи: універсальна екосистема смарт-будинку від великого виробника, відкрита платформа з активною спільнотою, фірмовий мобільний додаток виробника гібридних інверторів та хмарна платформа моніторингу для сонячних електростанцій.

1.4.1 Tuya Smart

Tuya Smart – універсальна екосистема смарт-будинку китайської компанії Tuya Inc., що об’єднує понад мільйон сертифікованих пристроїв різних виробників, маркованих логотипом «Powered by Tuya». До асортименту входять розумні розетки, реле, лампи, термостати, датчики, камери. Керування виконується через мобільний додаток або через локальний хаб. Платформа оперує концепцією сценаріїв «якщо – то», в яких умовами виступають час доби, стан датчиків та геолокація. Енергетичні аспекти – гібридні інвертори, рівень заряду батареї, реакція на блекаут – у функціональному наборі відсутні.

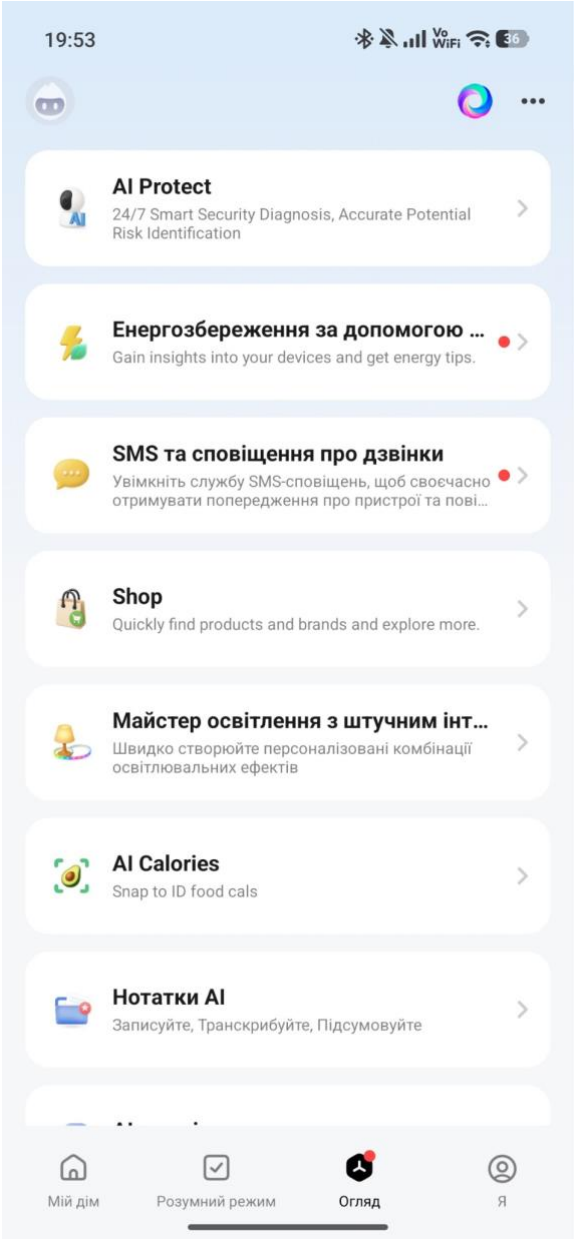


Рис. 1.4 – Інтерфейс додатка Tuya Smart

Переваги:

- широкий перелік підтримуваного обладнання, включно з бюджетними пристроями;
- зручний інтерфейс мобільного додатка для Android та iOS;
- розвинута система користувацьких сценаріїв на основі часових і геолокаційних тригерів.

Недоліки:

- відсутність функцій для роботи з гібридними енергоустановками (інвертор, батарея, графік відключень);
- закритість екосистеми – стороннє обладнання без сертифікації Tuya не інтегрується;
- передача телеметрії через хмарну інфраструктуру виробника, що залежить від наявності інтернет-з'єднання.

1.4.2 Home Assistant

Home Assistant [5] – відкрита платформа автоматизації розумного будинку, що розгортається на локальному сервері користувача (як правило, на мікрокомп'ютері Raspberry Pi або контейнері Docker). Підтримує понад 2 000 інтеграцій через систему модулів та надає мову опису сценаріїв на YAML. У спільноті існують модулі для роботи з гібридними інверторами окремих виробників (Solax, Victron, Deye), а також з MQTT-брокерами для обміну з кастомним обладнанням. Користувач формує власну логіку автоматизації самостійно, що відкриває значні можливості, але одночасно вимагає інженерної кваліфікації.

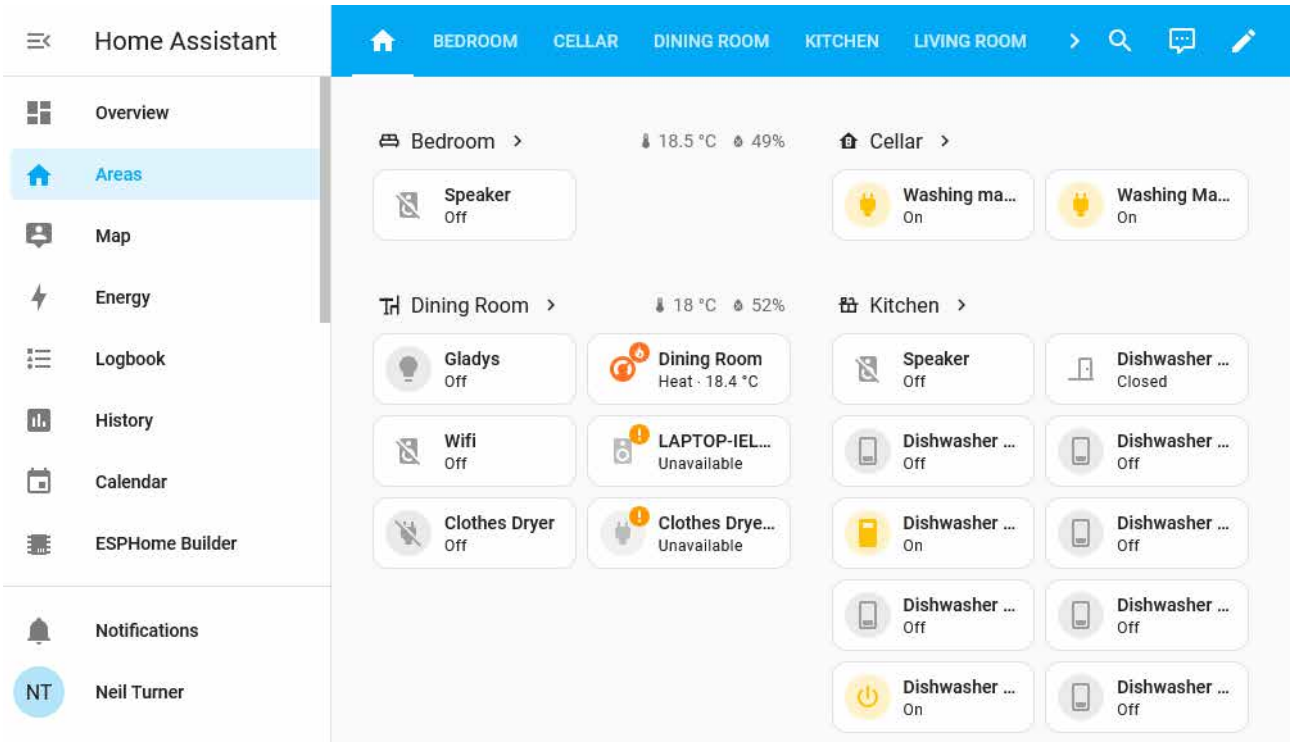


Рис. 1.5 – Інтерфейс Home Assistant з прикладом дашборда енергомоніторингу

Переваги:

- відкритий код, повна локальна робота без хмарних сервісів;
- широкий перелік інтеграцій з обладнанням різних виробників;
- гнучкість сценаріїв обмежена лише навичками користувача.

Недоліки:

- висока складність первинного налаштування та супроводу для нетехнічного користувача;
- відсутність готової логіки реакції на блекаути та каскадного відключення «з коробки» – користувач має реалізувати її самостійно у YAML-сценаріях;
- готові сценарії спільноти орієнтовані переважно на ринки з стабільним електропостачанням і не враховують специфіку графіків відключень.

1.4.3 EcoFlow App

EcoFlow App – фірмовий мобільний додаток виробника портативних та стаціонарних електростанцій (моделей River, Delta, Power Kits). Працює виключно з обладнанням EcoFlow та надає детальну телеметрію енергетичної установки – рівень заряду батареї, миттєве споживання і генерацію, температурні режими, передбачуваний час до повного розряду. Для обладнання серії Power Kits підтримується керування фірмовими розетками-розширювачами, що формують замкнений контур керування навантаженнями.

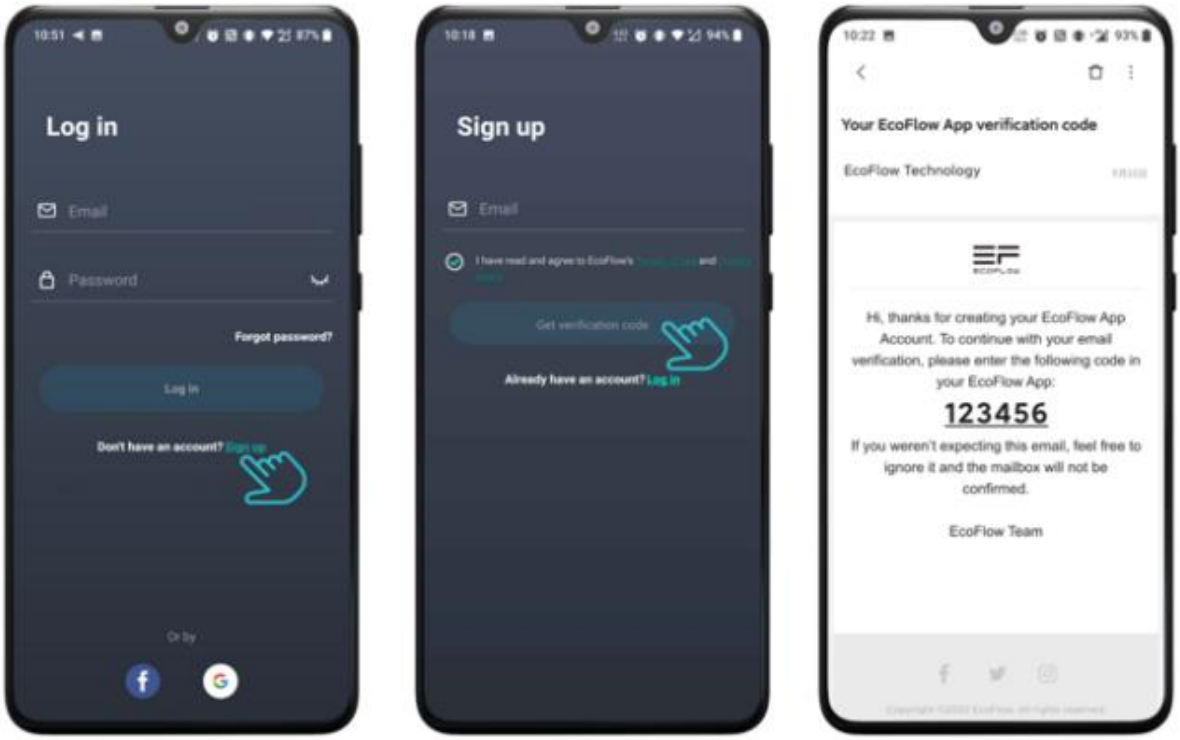


Рис. 1.6 – Інтерфейс додатка EcoFlow App

Переваги:

- розгорнута телеметрія енергоустановки з якісною візуалізацією;
- підтримка сценаріїв заряджання та інтеграція з фотоелектричними панелями виробника;
- стабільна робота мобільного додатка на Android та iOS.

Недоліки:

- працює тільки з обладнанням EcoFlow – інші інвертори та сторонні розумні розетки не підтримуються;
- відсутність роботи з графіком планових відключень за чергами споживачів;
- висока вартість входу в екосистему (стаціонарна станція з модулями розширення стартує від 4 000 USD).

1.4.4 Solax Cloud

Solax Cloud – хмарна платформа моніторингу від виробника гібридних інверторів Solax Power. Складається з мобільного додатка для кінцевого користувача та веб-інтерфейсу для дилерів та сервісних інженерів. Орієнтована переважно на побутові сонячні електростанції з накопиченням енергії. Надає історичні графіки генерації та споживання, статистику за добу/місяць/рік, базові сповіщення про несправності. Керування побутовими навантаженнями за межами обладнання Solax не передбачене.

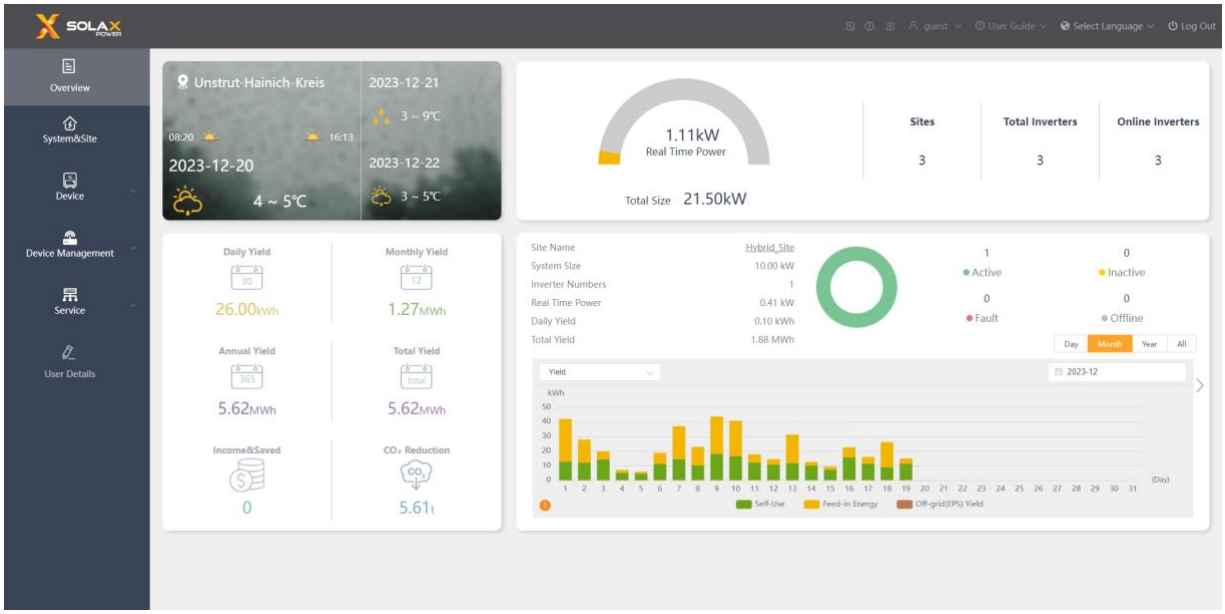


Рис. 1.7 – Інтерфейс Solax Cloud

Переваги:

- детальна історична аналітика генерації та споживання;
- автоматичне формування звітів для дилерів сонячних установок;
- глобальна інфраструктура моніторингу з підтримкою кількох регіональних дата-центрів.

Недоліки:

- працює виключно з інверторами Solax;
- відсутність керування побутовими навантаженнями (розетки, реле);
- залежність від хмари виробника – при її недоступності пропадає весь функціонал моніторингу.

1.4.5 Зведений аналіз відповідності вимогам

Результати зведеного аналізу подано у таблиці 1.1

Таблиця 1.1 – Відповідність наявних рішень функціональним вимогам

Вимога	**Tuya Smart**	**Home Assistant**	**EcoFlow App**	**Solax Cloud**
Керування побутовими пристроями	+	+	+	-
Телеметрія інвертора та батареї	-	±[1]	+	+
Класифікація стану напруги	-	-	-	-
Каскадне відключення за SoC	-	-	-	-
М'який старт пристроїв	-	-	-	-
Превентивна реакція на графік	-	-	-	-
Апаратна незалежність	-	+	-	-

[1] – реалізується через сторонні модулі спільноти та потребує ручного налаштування в YAML.

Жоден з чотирьох оглянутих представників не реалізує одночасно класифікації стану напруги, каскадного відключення за рівнем заряду батареї, м'якого старту пристроїв та превентивної реакції на графік планових відключень. Зазначена функціональна прогалина обумовлює доцільність розробки нового програмного забезпечення.

1.5 Постановка завдання

Метою даного проєкту є розробка програмного забезпечення для моніторингу та оптимізації енергоспоживання в розумному будинку, яке дозволить автоматично захищати побутове обладнання від нестабільної напруги, раціонально витрачати ресурс акумуляторної батареї в умовах планових та аварійних відключень зовнішньої електромережі та надавати користувачеві засоби спостереження за станом енергетичної установки. Програмне забезпечення має враховувати характерні для сучасного українського ринку умови експлуатації – графіки відключень за чергами споживачів, нестабільну якість напруги, обмеження пікової потужності побутових гібридних інверторів – та забезпечувати незалежність від конкретних виробників обладнання й зручність використання нетехнічним користувачем.

Вхідні дані системи

- Система повинна обробляти такі вхідні дані:
- параметри телеметрії гібридного інвертора: миттєва вихідна потужність, напруга змінного струму на стороні навантажень, частота, рівень заряду акумуляторної батареї, активне джерело живлення;
- стани підключених побутових пристроїв: ідентифікатор, увімкнено / вимкнено, миттєва споживана потужність;

- конфігураційні параметри пристроїв: клас критичності (критичний / важливий / некритичний), номінальна потужність, індивідуальна затримка м’якого старту;
- графік планових відключень електромережі для відповідної черги споживача від постачальника DTEK Yasno [17];
- користувацькі сценарії автоматизації: умова спрацювання (час доби або режим присутності), цільовий набір пристроїв, тип дії (увімкнення або вимкнення);
- ручні команди користувача на безпосереднє керування пристроями;
- облікові дані користувачів для автентифікації в системі.

Вихідні дані системи

Система повинна генерувати такі результати:

- зведений стан енергетичної установки в інтерфейсі мобільного додатка (поточне джерело живлення, рівень заряду батареї, миттєва сумарна потужність, перелік активних пристроїв);
- історичні графіки споживання, рівня заряду батареї та кількості відключень за горизонти день, тиждень, місяць;
- команди керування на сторону розумних розеток для вмикання та вимикання пристроїв за результатами роботи автоматичних правил або за прямою дією користувача;
- діагностичні повідомлення про блокування дій, що порушують технологічні обмеження (перевищення піка інвертора, конфлікт з захисною автоматикою);
- записи журналу подій про спрацювання автоматичних правил (каскадне відключення, «м'який старт», превентивна реакція на графік).

Умовно-постійна інформація

- Умовно-постійна інформація передбачає:
- порогові значення класифікації стану напруги (нижня та верхня межі норми, межа перенапруги);
- порогові значення SoC для трьох рівнів каскадного відключення;

- технологічний ліміт пікової потужності гібридного інвертора;
- довідник класів критичності побутових пристроїв;
- перелік черг споживачів постачальника електроенергії.

Функціональні вимоги

Розроблюване програмне забезпечення реалізує такі функціональні можливості:

- збір та зберігання телеметрії від гібридного інвертора з періодичністю не довшою за дві секунди;
- класифікація стану зовнішньої електромережі за чотирма дискретними станами (норма, просідання, перенапруга, відсутність живлення);
- каскадне відключення побутових навантажень за рівнем заряду батареї з трьома порогами активації;
- «м'який старт» пристроїв після відновлення якісної напруги з повторною перевіркою стану напруги перед кожним вмиканням;
- захист від перевищення піка вихідної потужності інвертора з блокуванням дій, що порушують ліміт;
- превентивне відключення некритичних споживачів за заданий час до прогнозованого моменту планового відключення;
- ручне керування підключеними пристроями з інтерфейсу мобільного додатка;
- створення та виконання користувацьких сценаріїв автоматизації за тригерами часу доби та режиму присутності;
- візуалізація поточного стану енергетичної установки та історичних показників у клієнтському додатку.

Нефункціональні вимоги

- Програмне забезпечення задовольняє такі якісні характеристики:
- апаратна незалежність архітектури – ізоляція джерела телеметрії та каналу керування пристроями за внутрішнім контрактом, що дозволяє підключення обладнання інших виробників через розробку відповідних адаптерів без модифікації ядра системи;

- швидкодія реагування на критичні події в електромережі з граничною затримкою не більше двох секунд між фіксацією зміни стану та виконанням захисної дії;
- кросплатформеність клієнтського додатка – підтримка мобільних платформ Android та iOS на основі єдиної кодової бази;
- захищеність каналу обміну даними між клієнтським додатком та серверною частиною з автентифікацією користувачів за обліковими записами.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ 2 присвячено етапу проєктування програмної системи, що складається з двох основних частин – інформаційного забезпечення та прикладного програмного забезпечення. На початку проєктування інформаційного забезпечення створюється логічна модель даних у вигляді ER-діаграми [20] з обґрунтуванням її відповідності третій нормальній формі [21]. Подальші підрозділи послідовно розкривають прикладну частину системи: діаграма класів формалізує доменну модель, діаграма пакетів демонструє декомпозицію системи на модулі, а діаграма компонентів завершує етап проєктування представленням розгорнутих компонентів та зв’язків між ними.

2.1 Логічна модель даних

Інформаційна база системи EnergyHub реалізована на основі реляційної СУБД PostgreSQL [9] версії 15. Вибір реляційної моделі обґрунтований характером оперативних даних системи – кожна сутність предметної області (користувач, пристрій, джерело живлення, сценарій, графік відключень, запис телеметрії) має жорстко визначений набір атрибутів, що не змінюється від запису до запису, а зв’язки між сутностями добре формалізуються мовою зовнішніх ключів. Альтернативні моделі – документна або графова – не дають у даному випадку відчутних переваг і ускладнюють підтримку цілісності даних на рівні СУБД.

2.1.1 Опис сутностей предметної області

Логічна модель включає сім сутностей, що відповідають ключовим інформаційним об’єктам системи.

Сутність **Користувач** (*users*) представляє зареєстрованого користувача системи. Первинним ключем є цілочисельний ідентифікатор *id_user*. До основних атрибутів належать: електронна адреса для автентифікації (унікальне

значення), ідентифікатор облікового запису для зовнішньої автентифікації (Google OAuth), рівень доступу (звичайний користувач або адміністратор), поточний режим присутності («вдома» / «не вдома») та номер черги відключень за класифікацією постачальника електроенергії DTEK Yasno.

Сутність **Джерело живлення** (*power_source*) описує гібридний інвертор, з яким працює користувач. Кожен користувач у системі асоційований не більш ніж з одним джерелом живлення (зв'язок один до одного). До атрибутів сутності входять: модель інвертора, поточний рівень заряду акумуляторної батареї, поточне сумарне навантаження, ознака наявності зовнішньої мережі, миттєве значення напруги в мережі. Усі чисельні показники обробляються в режимі реального часу і відображають останній відомий стан установки.

Сутність **Пристрій** (*device*) представляє побутовий електроприлад, підключений до системи через комутаційний пристрій (розумну розетку або реле). Атрибути сутності включають назву пристрою, групу пріоритету (критичний / важливий / некритичний), поточний стан реле (увімкнено або вимкнено), номінальну потужність у ватах, ознаку можливості переривання живлення під час роботи та індивідуальну затримку м'якого старту в секундах. Кожен пристрій належить одному користувачеві.

Сутність **Запис телеметрії** (*telemetry_record*) забезпечує зберігання історії спостережуваних метрик. Кожен запис фіксує одне вимірювання та містить часову мітку реєстрації, тип метрики (рівень заряду, напруга, потужність тощо), чисельне значення, опціональні посилання на джерело живлення та пристрій, з яких отримано вимірювання, а також обов'язкове посилання на користувача-власника даних. На сутність накладено чотири складені індекси для прискорення вибірок при побудові історичних графіків та статистичних зведень.

Сутність **Сценарій оптимізації** (*optimization_scenario*) описує користувацьке правило автоматизації. Атрибути включають назву правила, інтервал часу спрацювання (поля *time_from* та *time_to* у форматі HH:MM), тип дня (будній, вихідний або довільний), вимогу щодо режиму присутності та

ознаку активності правила. Кожен сценарій належить одному користувачеві та опціонально пов’язаний з конкретним джерелом живлення.

Сутність **Сценарій-Пристрій** (*scenario_device*) є асоціативною сутністю, що реалізує зв’язок «багато до багатьох» між сценаріями та пристроями. Первинним ключем є складений ключ з ідентифікаторів сценарію та пристрою. До атрибутів додано поле дії (*action*), що визначає, увімкнути або вимкнути конкретний пристрій при спрацюванні сценарію. Наявність окремого атрибута дії перетворює таблицю-зв’язку на повноцінну асоціативну сутність та виправдовує її окреме існування у моделі.

Сутність **Графік відключень** (*outage_schedule*) зберігає календарний розклад планових відключень електроенергії, отриманий від зовнішнього джерела. До атрибутів належать: дата, час початку та закінчення інтервалу відключення, тип відключення (плановане або можливе), черга споживача, ідентифікатор зовнішнього джерела даних, часова мітка отримання запису. Окреме булеве поле *preemptive_fired* фіксує, чи було вже виконане превентивне відключення некритичних споживачів для даного інтервалу – це запобігає повторному спрацюванню при перезапуску системи.

2.1.2 ER-діаграма

Графічне представлення логічної моделі даних подано на рис. 2.1.

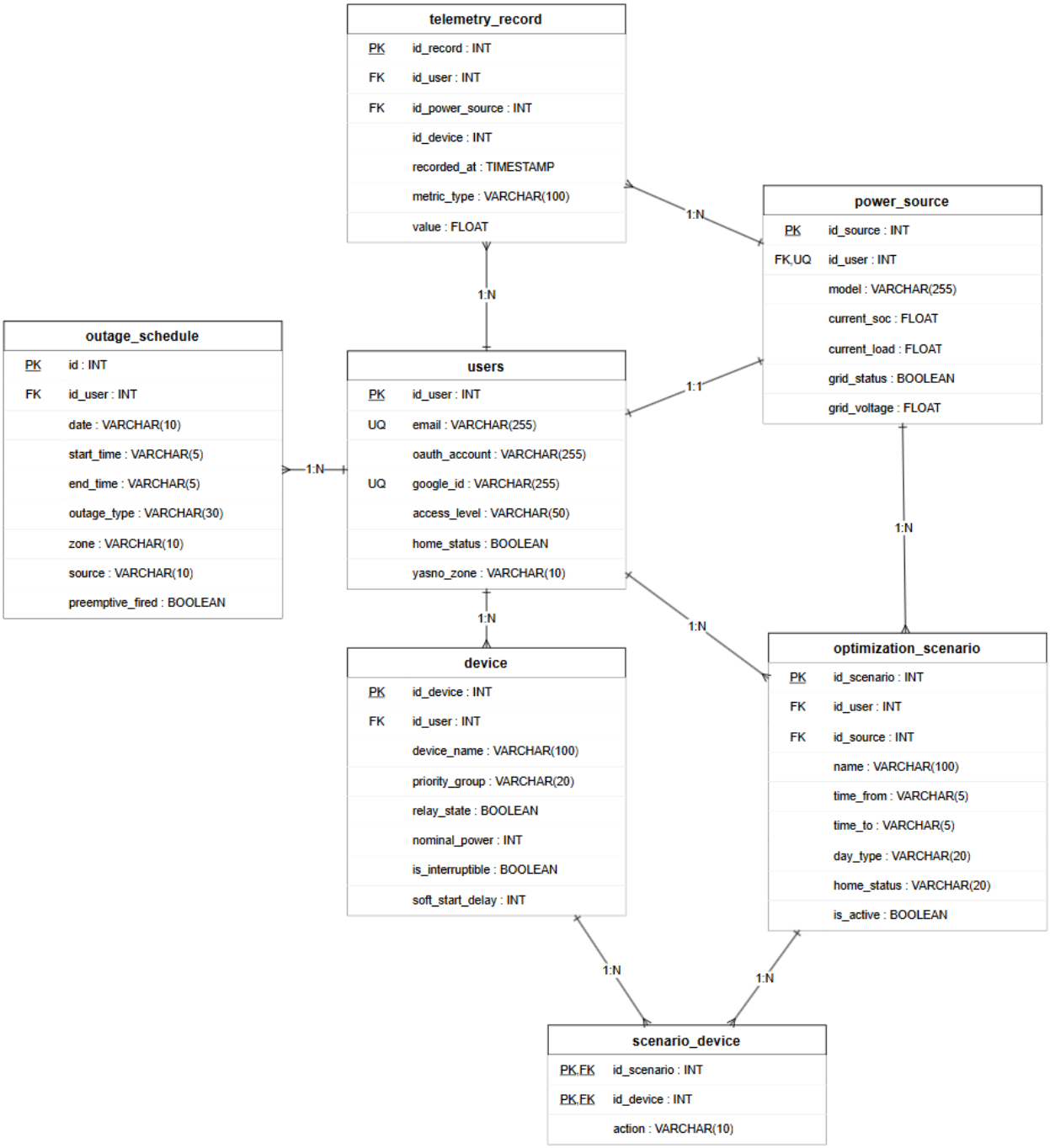


Рис. 2.1 – ER-діаграма інформаційної бази системи

На ER-діаграмі зв'язки позначені нотацією Crow's Foot: 1:1 між User та PowerSource, п'ять зв'язків 1:N та M:N між сценарієм і пристроєм через scenario_device. Каскадне видалення від User.

- зв’язок «один до одного» (1:1) між Користувачем та Джерелом живлення, реалізований через унікальний зовнішній ключ *id_user* у таблиці *power_source*;
- зв’язок «один до багатьох» (1:N) у п’яти випадках: Користувач – Пристрій, Користувач – Сценарій оптимізації, Користувач – Графік відключень, Користувач – Запис телеметрії, Джерело живлення – Запис телеметрії;
- зв’язок «багато до багатьох» (M:N) між Сценарієм оптимізації та Пристроєм, декомпозований у дві сутності 1:N через введення асоціативної сутності *scenario_device*.

Усі зв’язки від сутності Користувач реалізовані з правилом каскадного видалення (*ON DELETE CASCADE*): при видаленні облікового запису користувача автоматично видаляються всі його пристрої, сценарії, графіки відключень та записи телеметрії, що забезпечує референційну цілісність бази даних без потреби у ручному прибиранні.

2.1.3 Обґрунтування відповідності реляційним вимогам

Реляційна модель вимагає від інформаційної бази відповідності низці нормальних форм. Розглянемо побудовану модель з точки зору перших трьох нормальних форм, що є достатнім критерієм для оперативної бази даних.

Побудована модель відповідає трьом нормальним формам.

Перша нормальна форма (1НФ). Перша нормальна форма вимагає атомарності значень атрибутів – кожен атрибут зберігає неподільне значення, а не структуру (масив, об’єкт, список). У побудованій моделі всі атрибути є скалярними. Числові атрибути (рівень заряду, потужність, затримка) мають тип *Int* або *Float*, текстові (назва, тип метрики, час) – *VarChar* фіксованої довжини, часові – *Timestamp* або форматований рядок *VarChar(5)* для часу доби. Жоден атрибут не містить списків або вкладених структур. Значення, що в логічній моделі могли бути списками (наприклад, перелік пристроїв сценарію), у фізичній моделі винесені в окрему асоціативну сутність *scenario_device*. Таким чином, перша нормальна форма виконується для всіх сутностей моделі.

Друга нормальна форма (2НФ). Друга нормальна форма вимагає, щоб усі неключові атрибути функціонально залежали від повного первинного ключа, а не від його частини. Ця вимога актуальна лише для сутностей з композитним первинним ключем – у побудованій моделі такою є асоціативна сутність *scenario_device*. Її первинний ключ складається з полів *id_scenario* та *id_device*, а єдиний неключовий атрибут – поле *action* – функціонально залежить від обох частин ключа одночасно: дія, що буде виконана, визначається саме парою «сценарій + пристрій», оскільки один і той самий пристрій може фігурувати в різних сценаріях з різними діями (наприклад, в одному сценарії – увімкнення, в іншому – вимикання). Решта сутностей моделі мають прості (одно-полеві) первинні ключі, тож умова 2НФ для них виконується автоматично.

Третя нормальна форма (3НФ). Третя нормальна форма вимагає відсутності транзитивних залежностей – тобто залежностей одного неключового атрибута від іншого неключового атрибута. Розглянемо потенційні точки порушення:

- у сутності *device* атрибути *priority_group*, *nominal_power*, *is_interruptible*, *soft_start_delay* є незалежними характеристиками пристрою, що задаються користувачем при первинному налаштуванні; жоден з них не виводиться з іншого;
- у сутності *users* номер черги відключень *yasno_zone* визначається фізичною адресою об’єкта і не залежить від інших атрибутів профілю користувача;
- у сутності *optimization_scenario* атрибути часу спрацювання, типу дня та вимоги щодо режиму присутності є самостійними параметрами правила;
- у сутності *telemetry_record* чисельне значення вимірювання *value* змістовно інтерпретується разом з типом метрики *metric_type*, але це не транзитивна залежність – обидва атрибути визначаються виключно первинним ключем *id_record*.

Таким чином, у моделі відсутні транзитивні залежності між неключовими атрибутами, і третя нормальна форма виконується.

Виконання 1НФ, 2НФ та 3НФ підтверджує відповідність побудованої моделі реляційним вимогам та її придатність для реалізації в реляційній СУБД PostgreSQL.

2.2 Діаграма класів та кооперацій

Діаграма класів формалізує статичну структуру системи: атрибути, операції та зв'язки між класами. Розбиття на кооперації виокремлює групи класів для конкретних сценаріїв використання.

2.2.1 Загальна діаграма класів

Доменна модель системи включає сім класів-сутностей та чотири перелічувані типи (енумерації), що формалізують дискретні стани атрибутів.

Клас *User* представляє зареєстрованого користувача системи. Атрибутами класу є ідентифікатор, електронна адреса, рівень доступу типу *AccessLevel*, поточний режим присутності, ідентифікатор зони відключень. До операцій класу віднесено перемикання режиму присутності (*toggleHomeStatus*) та перевірку наявності адміністративних прав (*isAdmin*).

Клас *PowerSource* моделює гібридний інвертор. Атрибути класу містять модель пристрою, поточний рівень заряду батареї, миттєве сумарне навантаження, ознаку наявності зовнішньої мережі та чисельне значення напруги. Ключовими операціями класу є класифікація стану напруги (*classifyVoltage*), що повертає значення типу *VoltageState* та обчислення доступної потужності (*calculateAvailablePower*) – різниці між технологічним лімітом інвертора та поточним сумарним навантаженням.

Клас *Device* представляє побутовий електроприлад, підключений до системи. Атрибути включають назву, групу пріоритету типу *PriorityGroup*, поточний стан реле, номінальну потужність, ознаку допустимості переривання живлення та індивідуальну затримку м’якого старту. Операції класу – увімкнення (*turnOn*) та вимикання (*turnOff*) пристрою, причому увімкнення

повертає булеву ознаку успіху, оскільки може бути заблоковане захисною автоматикою.

Клас *OptimizationScenario* описує користувацьке правило автоматизації. Атрибутами є назва, межі часового інтервалу спрацювання, тип дня, вимога щодо режиму присутності та ознака активності. Основними операціями класу є перевірка відповідності поточного контексту умовам сценарію (*matches*) та виконання дій сценарію над пов’язаними пристроями (*execute*).

Клас *ScenarioDevice* є асоціативним класом, що пов’язує сценарій з конкретним пристроєм та зберігає атрибут дії, яка має бути виконана при спрацюванні сценарію. Операція класу – застосування дії (*apply*) – інкапсулює логіку увімкнення або вимикання залежно від значення атрибута.

Клас *TelemetryRecord* представляє один запис телеметричного спостереження. Атрибутами є часова мітка реєстрації, тип метрики та чисельне значення. До операцій класу віднесено форматування часової мітки для відображення в інтерфейсі користувача (*getFormattedTime*).

Клас *OutageSchedule* зберігає інформацію про один інтервал планового відключення. Атрибутами є дата, межі часового інтервалу, тип відключення типу *OutageType*, зона споживача та ознака виконання превентивної реакції. Операція *isUpcoming* повертає булеву ознаку наближення інтервалу відключення з урахуванням заданого порогу часу, що використовується модулем превентивної реакції.

Перелічувані типи VoltageState (OFF, NORMAL, DIRTY, OVERVOLTAGE), PriorityGroup (CRITICAL, IMPORTANT, SECONDARY), AccessLevel (USER, ADMIN) та OutageType формалізують дискретні множини значень відповідних атрибутів.

Зв'язки між класами відповідають ER-діаграмі з підрозділу 2.1: User пов'язаний композицією з Device, OptimizationScenario, OutageSchedule, TelemetryRecord та асоціацією 1:1 з PowerSource. Видалення користувача каскадно видаляє всі пов'язані об'єкти.

Графічне представлення загальної діаграми класів подано на рис. 2.2.

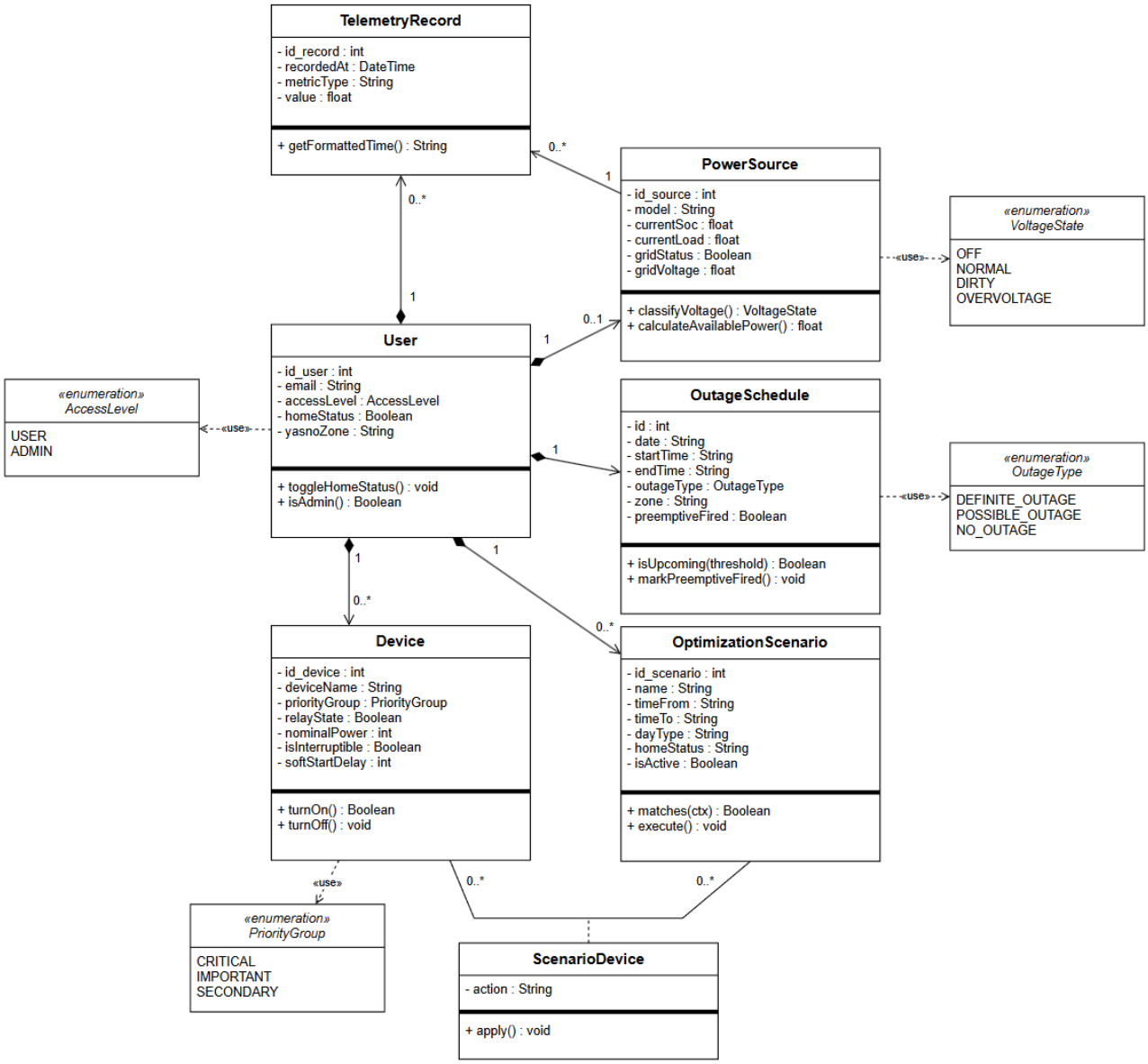


Рис. 2.2 – Загальна діаграма класів доменної моделі

2.2.2 Розбиття на кооперації

Для подальшого вибору шаблонів об’єктно-орієнтованого підходу загальна діаграма класів декомпозована на три кооперації за функціональною ознакою. Кожна кооперація виокремлює підмножину класів, що спільно реалізують один з ключових сценаріїв системи.

Кооперація «Моніторинг та зберігання телеметрії» включає класи *User*, *PowerSource* та *TelemetryRecord*. Сценарій взаємодії наступний:

компонент опитування інвертора періодично викликає операцію *updateTelemetry* об’єкта *PowerSource*, яка оновлює внутрішній стан джерела живлення на основі отриманого пакета. Кожне суттєве вимірювання трансформується в новий екземпляр *TelemetryRecord*, який пов’язується з відповідним користувачем та джерелом живлення. Накопичена історія використовується клієнтським додатком для побудови часових графіків. Графічне представлення кооперації подано на рис. 2.3.

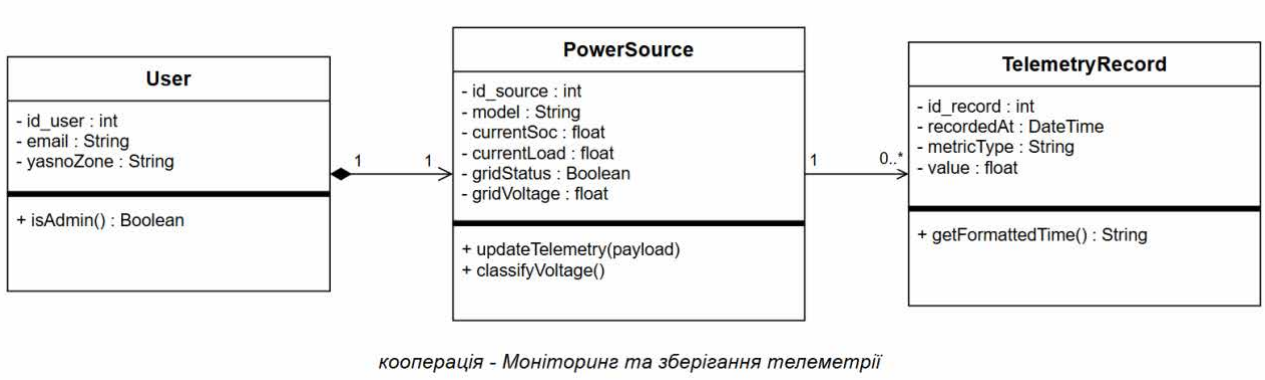


Рис. 2.3 – Кооперація «Моніторинг та зберігання телеметрії»

Кооперація «Захисна автоматика» включає класи *User*, *PowerSource* та *Device* з використанням енумерацій *VoltageState* та *PriorityGroup*. Сценарій реалізує дві основні поведінки. Перша – каскадне відключення навантажень: при опусканні рівня заряду батареї нижче порогових значень об’єкт *PowerSource* ініціює виклик операції *turnOff* для тих пристроїв, групи пріоритету яких відповідають поточному рівню каскаду. Друга поведінка – «м'який старт»: при поверненні якісної напруги об’єкт *PowerSource* послідовно викликає операцію *turnOn* для раніше відключених пристроїв з урахуванням індивідуальних затримок та повторної перевірки якості напруги через операцію *classifyVoltage*. Графічне представлення кооперації подано на рис. 2.4.

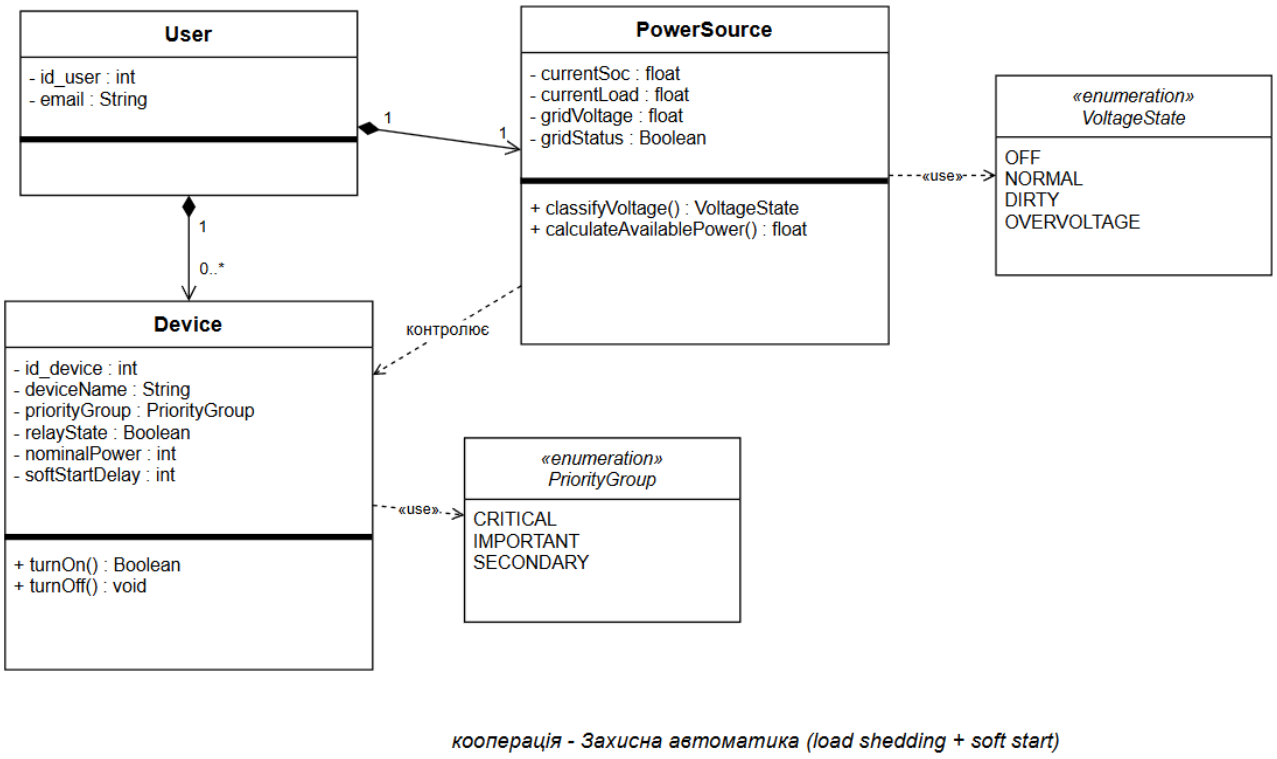


Рис. 2.4 – Кооперація «Захисна автоматика»

Кооперація «Сценарії та графіки відключень» включає класи *User*, *OptimizationScenario*, *ScenarioDevice*, *Device* та *OutageSchedule*. Сценарій взаємодії розгалужений на дві гілки. Перша – виконання користувацького сценарію: за тригером часу або зміни режиму присутності об’єкт *OptimizationScenario* викликає операцію *matches* для перевірки умов, і при позитивному результаті проходить колекцією пов’язаних *ScenarioDevice*, викликаючи для кожного операцію *apply*, яка делегує дію (увімкнення або вимикання) відповідному об’єкту *Device*. Друга гілка – превентивна реакція на графік відключень: модуль періодично перевіряє об’єкти *OutageSchedule* через операцію *isUpcoming*, і при наближенні інтервалу відключення ініціює відключення пристроїв груп *IMPORTANT* та *SECONDARY* з оновленням ознаки *preemptiveFired* для виключення повторного спрацювання. Графічне представлення кооперації подано на рис. 2.5.

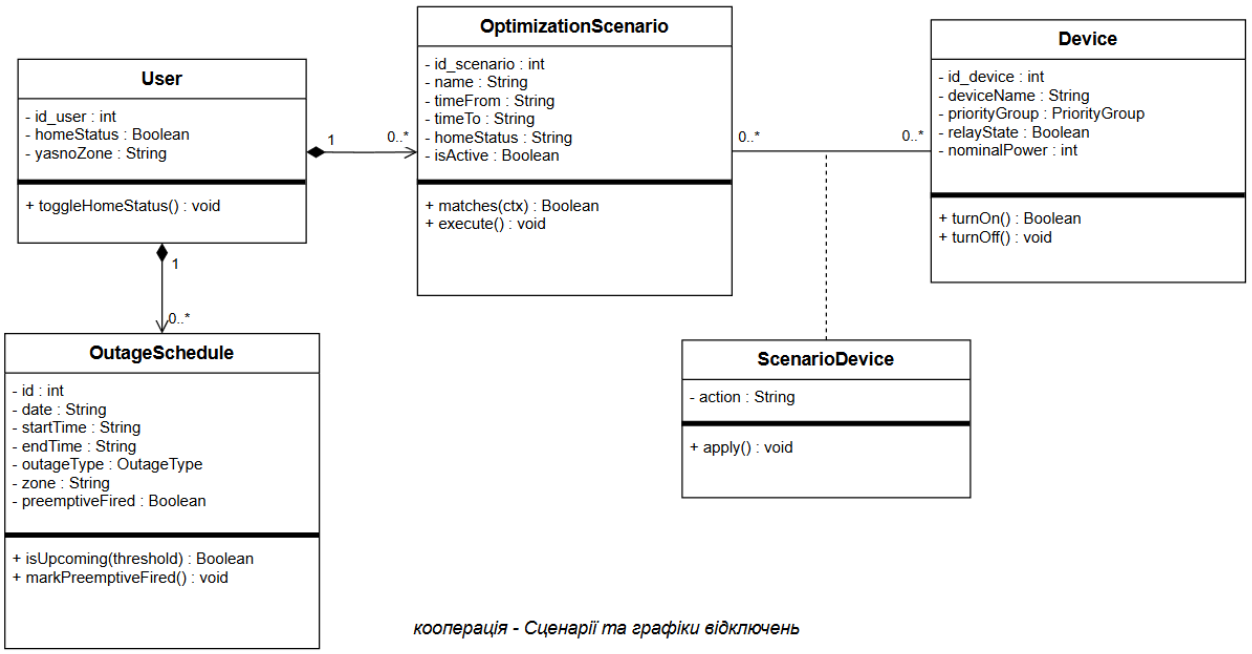


Рис. 2.5 – Кооперація «Сценарії та графіки відключень»

Виділення трьох кооперацій дає змогу обґрунтовано підійти до вибору шаблонів об’єктно-орієнтованого підходу. Для кооперації моніторингу природно застосовується шаблон Observer [22], оскільки клієнтські компоненти потребують повідомлення про оновлення стану джерела живлення. Для кооперації захисної автоматики застосовується шаблон Strategy, що інкапсулює різні алгоритми відбору пристроїв для відключення (за рівнем заряду, за класом критичності, за можливістю переривання). Для кооперації сценаріїв застосовується шаблон Command, який представляє кожну дію над пристроєм як окремий об’єкт, що дозволяє уніфікувати ручне керування та виконання сценаріїв. Перелічені шаблони визначатимуть структуру модулів програмної системи, що описуватиметься в розділі 3.

2.3 Діаграма пакетів

Діаграма пакетів представляє архітектуру програмної системи у вигляді ієрархії модулів-контейнерів. Кожен пакет групує функціонально близькі класи та підмодулі, що дозволяє розглядати систему на рівні абстракції вищому за

окремі класи. На відміну від діаграми класів, яка фокусується на статичній структурі доменної моделі, діаграма пакетів акцентує увагу на архітектурній декомпозиції – поділі системи на шари за відповідальностями та визначенні дозволених напрямків залежностей між шарами.

Архітектура системи EnergyHub побудована за чотиришаровою [23] моделлю, що відповідає принципам багатошарової архітектури (Layered Architecture). Така модель забезпечує сувору однонаправленість залежностей зверху вниз – кожен шар може використовувати лише шар, що розташований безпосередньо нижче, без зворотних посилань. Це виключає циклічні залежності між модулями та підвищує супроводжуваність системи: зміна реалізації нижчого шару не зачіпає вищі, якщо при цьому зберігається інтерфейс взаємодії.

Шар Presentation включає мобільний додаток (MobileApp), веб-інтерфейс (WebUI) та REST-шлюз (APIGateway) для маршрутизації запитів і валідації токенів OAuth 2.0 [24].

Шар BusinessLogic містить ядро системи: ScenarioEngine для користувацьких сценаріїв, RuleEngine для захисних правил (каскадне відключення, «м'який старт», захист від перевищення піка), SOCMonitor для моніторингу заряду та OutageService для обробки графіків відключень.

Шар Infrastructure надає MQTTBroker для обміну повідомленнями, AuthService для автентифікації та приватний пакет Database для доступу до PostgreSQL.

Шар ExternalServices містить адаптери до TuyaCloud (керування розетками), QoltecAPI (телеметрія інвертора), YasnoAPI (графіки відключень).

Залежності між шарами однонаправлені зверху вниз, що виключає циклічні зв'язки. Кооперації з підрозділу 2.2 розміщуються переважно в шарі BusinessLogic. Доменні класи – у пакеті Database шару Infrastructure.

Графічне представлення діаграми пакетів подано на рис. 2.6.

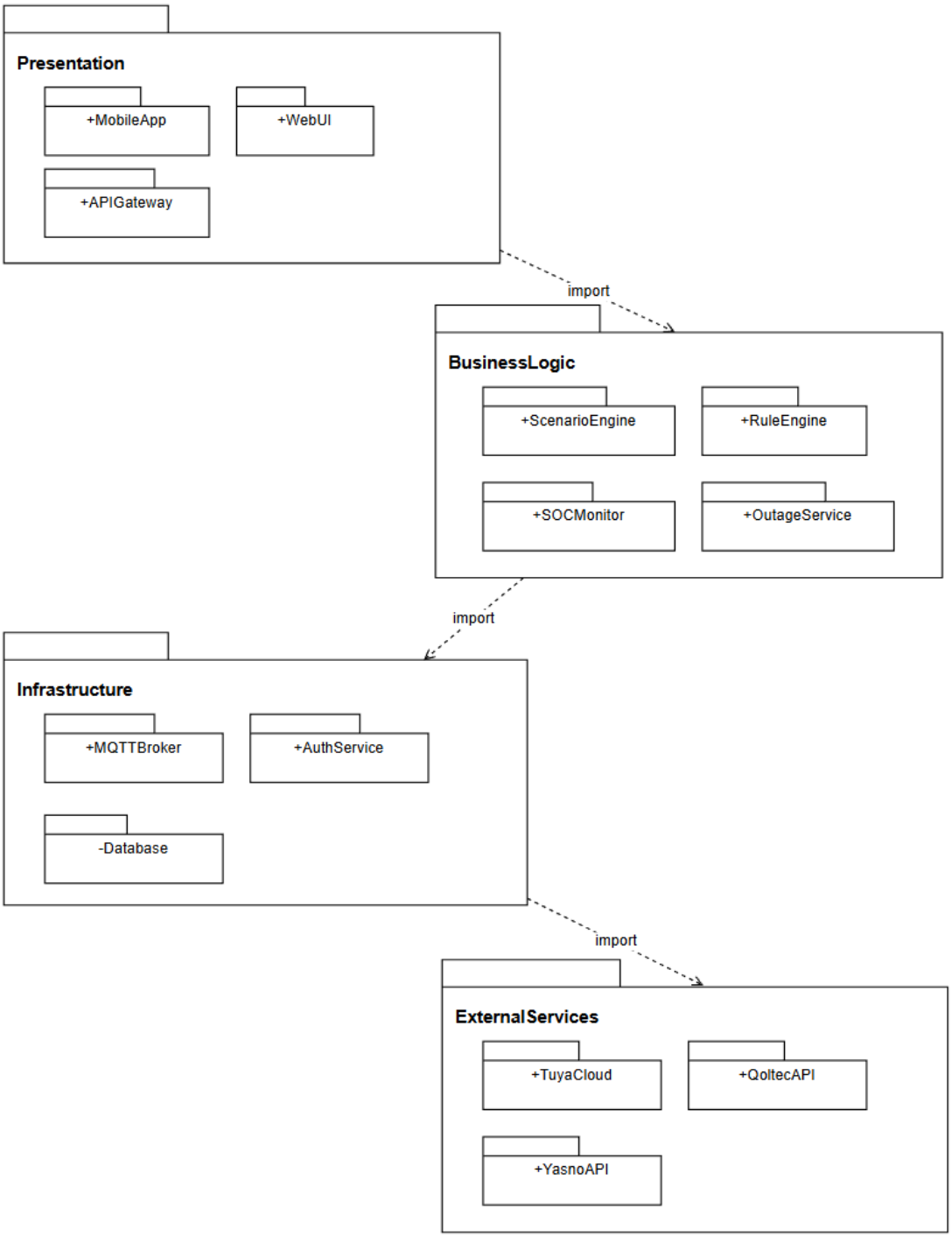


Рис. 2.6 – Діаграма пакетів системи EnergyHub

2.4 Діаграма компонентів

Діаграма компонентів представляє систему на рівні фізично розгортуваних одиниць. Архітектура поділена на чотири зони: клієнтська (React Native), серверна (Node.js + Express), база даних (PostgreSQL) та зовнішні сервіси

(Mosquitto, TuyaCloud, QoltecAPI, YasnoAPI). Протоколи: HTTP/REST, Prisma ORM/SQL, MQTT, HTTPS.

Архітектура EnergyHub поділена на чотири зони компонентів – клієнтську, серверну, рівень бази даних та зовнішні сервіси. Між зонами визначено протоколи взаємодії: HTTP/REST між клієнтом і сервером, Prisma ORM поверх SQL між сервером і базою даних, MQTT між сервером та обладнанням, HTTPS для звернення до зовнішніх API.

Мобільний застосунок (React Native, Expo) є єдиним інтерфейсом користувача. Включає п'ять екранів: дашборд, графіки, сценарії, налаштування пристрою, демо-режим. Взаємодіє з бекендом через HTTP/REST.

- Серверна частина (Node.js, Express, Prisma) є посередником між клієнтом, базою даних, MQTT-брокером і зовнішніми сервісами. Маршрути REST API розбиті на вісім функціональних груп. Бізнес-логіка реалізована в шести сервісних модулях.
- *routes/* – маршрути REST API за функціональними групами: *users.js* (автентифікація, профіль), *devices.js* (CRUD пристроїв, ручне керування реле), *scenarios.js* (управління користувацькими сценаріями), *telemetry.js* (запис і читання історії), *powerSource.js* (стан гібридного інвертора), *outages.js* (графіки відключень), *demo.js* (адміністративні команди), *status.js* (health-check).
- *services/* – бізнес-логіка системи: *ruleEngine.js* реалізує захисну автоматику – класифікацію стану напруги, каскадне відключення за рівнем SoC, послідовний «м'який старт» пристроїв з повторною перевіркою якості напруги; *scenarioEngine.js* виконує користувацькі сценарії автоматизації за тригерами часу та режиму присутності; *socService.js* забезпечує неперервний моніторинг рівня заряду акумулятора; *outageService.js* обробляє графіки планових відключень та ініціює превентивну реакцію; *yasnoService.js* отримує добові розклади з зовнішнього джерела DTEK Yasno; *seedService.js* виконує початкове заповнення бази даних демонстраційними даними.

- *models/* – Prisma-схема та моделі: *User*, *Device*, *OptimizationScenario*, *ScenarioDevice*, *TelemetryRecord*, *PowerSource*, *OutageSchedule*. Prisma генерує типізовані запити до PostgreSQL та автоматизує управління схемою бази через систему міграцій.
- *middleware/* – проміжне програмне забезпечення: *auth.js* перевіряє JWT-токен у кожному захищеному запиті та виконує авторизацію за рівнем доступу користувача.
- *index.js* – точка входу серверного процесу: ініціалізація Express-застосунку, підписка на MQTT-топіки для отримання телеметрії, запуск періодичних задач (опитування SoC з періодом дві секунди, перевірка графіка відключень кожні тридцять секунд, виконання сценаріїв кожну хвилину).
- *docker-compose.yml* – файл оркестрації контейнерів, що запускає PostgreSQL з персистентним томом для збереження даних між перезапусками та Eclipse Mosquitto як MQTT-брокер.

PostgreSQL 15 зберігає всі персистентні дані. Prisma ORM забезпечує типізований доступ. Складені індекси на telemetry_record прискорюють побудову історичних графіків.

- *MQTTBroker* (Eclipse Mosquitto) – посередник обміну повідомленнями між серверною частиною та обладнанням за моделлю «публікація-підписка». Бекенд підписаний на топіки телеметрії від гібридного інвертора та публікує команди керування реле на топіки смарт-розеток.
- *TuyaCloud* API – хмарний сервіс Tuya для керування смарт-розетками. Бізнес-логіка через відповідний адаптер надсилає команди вмикання та вимикання пристроїв і отримує підтвердження виконання через Tuya OpenAPI.
- *QoltecAPI* – програмний інтерфейс гібридного інвертора Qoltec MaxFlow. Через брокер MQTT публікує телеметричні пакети з

показниками рівня заряду батареї, миттєвої потужності, напруги та активного джерела живлення.

- *YasnoAPI* – публічний API постачальника DTEK Yasno. Сервіс *yasnoService* періодично запитує добові розклади планових відключень за відповідною чергою споживача та зберігає їх у таблиці *outage_schedule*.

Графічне представлення діаграми компонентів подано на рис. 2.7.

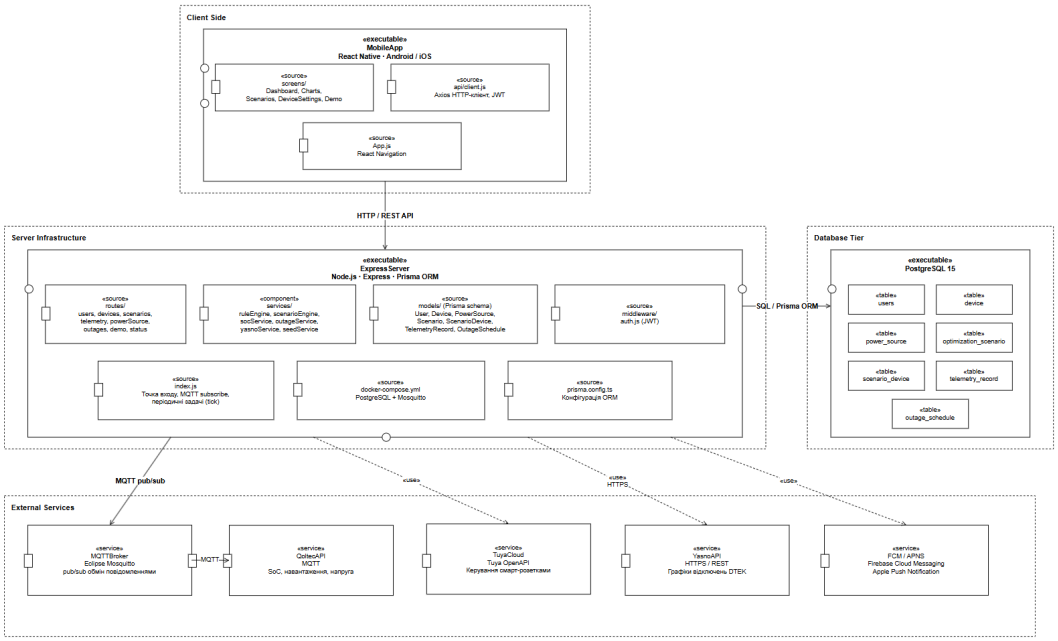


Рис. 2.7 – Діаграма компонентів системи EnergyHub

Завершення етапу проєктування діаграмою компонентів дає змогу перейти до реалізації – вибору конкретних інструментальних засобів розробки, написання коду модулів та налаштування інфраструктури розгортання, що становить зміст розділу 3.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ 3 присвячено реалізації програмної системи EnergyHub на основі проєктних рішень, сформульованих у розділі 2. У підрозділі 3.1 описано систему управління інформаційною безпекою, що включає механізми автентифікації, авторизації та ізоляції даних. У підрозділі 3.2 представлено фізичну реалізацію

інформаційної бази в реляційній СУБД PostgreSQL [9] з обґрунтуванням схеми індексування та механізмів забезпечення цілісності. Підрозділ 3.3 містить обґрунтування вибору технологічного стеку та інструментарію розробки. У підрозділі 3.4 наведено алгоритмізацію ключових модулів системи з блок-схемами та фрагментами програмного коду.

3.1 Розробка системи управління інформаційною безпекою

Програмне забезпечення EnergyHub оперує даними побутового рівня та надає засоби фізичного впливу на електроустановку, тому забезпечення інформаційної безпеки є невід'ємною складовою розробки. Підхід до захисту інформації побудовано з урахуванням загальних рекомендацій стандарту ISO/IEC 27001 [25] щодо управління доступом, автентифікації та логування, адаптованих до масштабу побутової системи. Реалізовано п'ять основних складових: рольову модель доступу, автентифікацію користувачів, авторизацію дій, ізоляцію персональних даних та логування значущих подій.

Рольова модель доступу. У системі передбачено два рівні доступу, що зберігаються в атрибуті *access_level* сутності *User*. Рівень *user* надає права на повсякденну роботу з системою – перегляд телеметрії, керування власними пристроями, створення сценаріїв автоматизації. Рівень *admin* додатково надає права на виконання адміністративних дій – управління графіками відключень для всіх користувачів інсталяції, активацію демонстраційних команд керування симульованим інвертором. Розподіл функцій між рівнями реалізовано на стороні серверної частини через middleware-функцію *adminOnly*, яка перевіряє рівень доступу перед допуском до відповідних маршрутів API.

Автентифікація реалізована через JWT-токени [26]. Токен формується при вході і передається в заголовку Authorization. Архітектурно передбачено інтеграцію з Google OAuth. У поточній версії для прискорення розробки використовується спрощений механізм через заголовок X-User-Id – перехід на JWT вимагає зміни лише одного middleware-модуля.

Авторизація перевіряє приналежність ресурсу користувачеві через фільтр where: { id_user: req.user.id_user }. Для адміністративних операцій – додаткова перевірка через middleware adminOnly (HTTP 403).

Ізоляція даних реалізована через обов'язковий зовнішній ключ на користувача-власника в усіх сутностях. Для захисної автоматики – окреме сховище стану Map<userId, RuleEngineState> у пам'яті сервера.

Логування фіксує ключові дії: зміну стану мережі, каскадне відключення, «м'який старт», превентивну реакцію. Кожен запис містить часову мітку, ідентифікатор користувача, тип і параметри події.

Обмін даними виконується за протоколом HTTPS з TLS 1.2+. MQTT-комунікація у виробничому середовищі – через MQTTS.

3.2 Розробка інформаційної бази

Інформаційну базу системи EnergyHub реалізовано в реляційній СУБД PostgreSQL версії 15 за схемою, що формально відповідає логічній моделі, представленій у підрозділі 2.1. Фізична реалізація доповнює логічну модель технічними деталями: конкретними типами даних PostgreSQL, обмеженнями цілісності, схемою індексування для оптимізації типових запитів та механізмами міграції схеми між версіями.

Числові ідентифікатори використовують Int з автоінкрементом. Текстові атрибути – VarChar з обмеженням довжини (255 для email, 100 для назв, 5 для часу HH:MM). Чисельні показники – Float. Часові мітки – Timestamp(3).

Унікальні обмеження накладено на email та google_id у User, а також на id_user в PowerSource (зв'язок 1:1). Каскадне видалення – на всіх ключах від User. Виняток – TelemetryRecord→PowerSource: ON DELETE SET NULL, щоб при заміні інвертора зберегти історичну телеметрію.

На TelemetryRecord (найбільша за обсягом сутність – ~30 записів/хв) створено чотири складених індекси для типових запитів: за користувачем і метрикою, за пристроєм, за типом метрики та за джерелом живлення. На OutageSchedule – два індекси: за користувачем і датою та за датою і зоною.

Міграції автоматизовано через Prisma. Поточна версія має три міграційних файли: ініціалізація, додавання затримки м'якого старту та розширення для сценаріїв і графіків відключень.

Сервіс seedService.js при першому запуску створює два демонстраційних облікових записи з типовим набором пристроїв та графік відключень.

3.3 Вибір інструментарію

Вибір інструментарію розробки виконано на основі переліку нефункціональних вимог, сформульованих у підрозділі 1.2: апаратна незалежність, швидкодія реагування, кросплатформеність клієнтського додатка, багатокористувацький режим. Кожне рішення обґрунтовано з точки зору відповідності переліченим вимогам та з порівнянням з альтернативними варіантами.

Мова програмування серверної частини – JavaScript на платформі Node.js [6] версії 20 LTS. Альтернативами розглядалися Python (фреймворки Django, FastAPI), Java (Spring Boot) та Go. Вибір JavaScript обґрунтований двома основними чинниками. По-перше, єдина мова для серверної та клієнтської частин системи зменшує когнітивне навантаження на розробника та виключає затрати на перемикання контексту між технологічними стеками. По-друге, асинхронна модель виконання Node.js на основі циклу подій (event loop) є природним вибором для систем, що обробляють велику кількість одночасних з’єднань з тривалими інтервалами очікування – підписки на MQTT-топіки, періодичні задачі моніторингу, REST-запити від мобільних клієнтів. Альтернативні платформи з блокуючою моделлю вимагали б створення

окремого потоку на кожен MQTT-підключений клієнт, що неефективно для побутового сценарію з десятками користувачів.

Веб-фреймворк – Express [7] версії 5**.** Альтернативами розглядалися Koa, Fastify, NestJS. Express обрано через мінімалізм архітектури, відсутність нав'язаних структурних обмежень та зрілість екосистеми – фреймворк використовується у виробничих системах понад десять років, має широку базу документації та готових інтеграцій. NestJS, що базується на парадигмі модулів і декораторів TypeScript, було відхилено через надмірну для побутової системи складність та обов'язкову прив'язку до TypeScript. Fastify надає вищу пропускну здатність за рахунок JIT-компіляції валідаторів, але ця перевага не суттєва для очікуваного навантаження системи (порядку 10–100 запитів на секунду).

Засіб об'єктно-реляційного відображення – Prisma версії 7.8.0 Альтернативами розглядалися TypeORM, Sequelize та прямі запити через драйвер *pg-native*. Prisma обрано через три ключові переваги: автоматична генерація типізованого клієнта на основі декларативної схеми, що виключає клас помилок невідповідності типів між кодом та базою даних; вбудована система міграцій з підтримкою прямого та зворотного перетворення; декларативна мова опису моделі (Prisma Schema Language), що візуально читається простіше за рівняні Sequelize-моделі чи TypeORM-декоратори. Прямі запити через *pg-native* забезпечують максимальну швидкодію, але вимагають значно більших витрат на підтримку коду при змінах схеми.

Система управління базами даних – PostgreSQL версії 15. Альтернативами розглядалися MySQL та документна СУБД MongoDB. PostgreSQL обрано через підтримку складних типів даних (JSONB, масиви, перелічувані типи), повноцінних транзакцій з рівнями ізоляції, аналітичних функцій (віконні функції, агрегати за часовими інтервалами) та зрілу систему індексування з підтримкою складених і часткових індексів. MongoDB було відхилено через жорстку реляційність даних предметної області – кожна сутність має чітко визначений набір атрибутів, що не виправдовує гнучкість документної

моделі. MySQL поступається PostgreSQL у підтримці аналітичних запитів, які можуть знадобитись для майбутніх розширень функціональності.

Протокол обміну з обладнанням – MQTT 3.1.1 [9], брокер Eclipse Mosquitto 2.0 [8]. Вибір здійснювався між HTTP polling, WebSocket та MQTT. MQTT – стандарт для IoT, підтримується більшістю смарт-обладнання. Модель «публікація-підписка» дозволяє декільком підписникам одночасно отримувати телеметрію. Під час розробки виникла неочевидна проблема: при швидкому перезапуску серверного процесу Mosquitto іноді не встигав звільнити сесію попереднього клієнта, і нова підписка на ті самі топіки не працювала. Повідомлення просто губились без жодної помилки в логах. На пошук причини було витрачено дві доби через припущення, що проблема полягає в коді парсингу пакетів. Проблему було вирішено додаванням унікального client ID з таймстемпом та параметром clean session = true.

Фреймворк клієнтського додатка – React Native у середовищі Expo SDK 54. Вибір здійснювався між Flutter та React Native. Flutter було відхилено через Dart – щоб уникнути вивчення окремої мови заради одного проєкту. React Native дозволяє використовувати JavaScript – ту саму мову, що й серверна частина, –та забезпечує єдину кодову базу для Android та iOS. Expo спростив збірку – готовий шаблон, швидке перезавантаження, розповсюдження тестових збірок без окремих сервісів. Проте на практиці Expo мав свої підводні камені. На пристроях з Android API 24-26 зафіксовано артефакти відмальовування графіків бібліотеки react-native-chart-kit (зникнення або дублювання осі Y). Дефект було усунуто примусовим перерендерингом компонента при зміні орієнтації екрану. На платформі iOS аналогічних дефектів не зафіксовано, натомість виявлено затримку навігації між екранами при швидкому перемиканні вкладок, що було вирішено переходом на відкладену ініціалізацію екранів.

Контейнеризація – Docker та Docker Compose. Серверний процес, PostgreSQL та Mosquitto розгортаються в ізольованих контейнерах. На апаратній платформі Apple Silicon (M1) контейнер PostgreSQL 15 не ініціалізувався через дефект образу для архітектури ARM64. Проблему було вирішено явним

зазначенням платформи linux/arm64/v8 у конфігурації Docker Compose. На робочій Linux-машині подібних проблем не було.

Засоби розробки. Робота над проєктом виконувалась у середовищі Visual Studio Code з розширеннями для підтримки Prisma Schema, ESLint, Prettier. Для проєктування діаграм використовувався онлайн-редактор draw.io (diagrams.net) у форматі XML, що дозволяє контролювати версії діаграм нарівні з програмним кодом.

3.4 Алгоритмізація та програмування модулів

Підрозділ розкриває реалізацію двох ключових алгоритмів захисної автоматики –каскадного скидання навантаження за рівнем заряду батареї та м'якого старту пристроїв. Обидва алгоритми відсутні в оглянутих у підрозділі 1.4 рішеннях та становлять основу функціональної відмінності розроблюваної системи.

3.4.1 Алгоритм каскадного скидання навантаження

Каскадне скидання активується в режимі автономного живлення. Три пороги: 50 % → другорядні, 20 % → важливі, 10 % → аварійне. Прапорці в RuleEngineState виключають повторне спрацювання.

Блок-схема алгоритму подана на рис. 3.1.

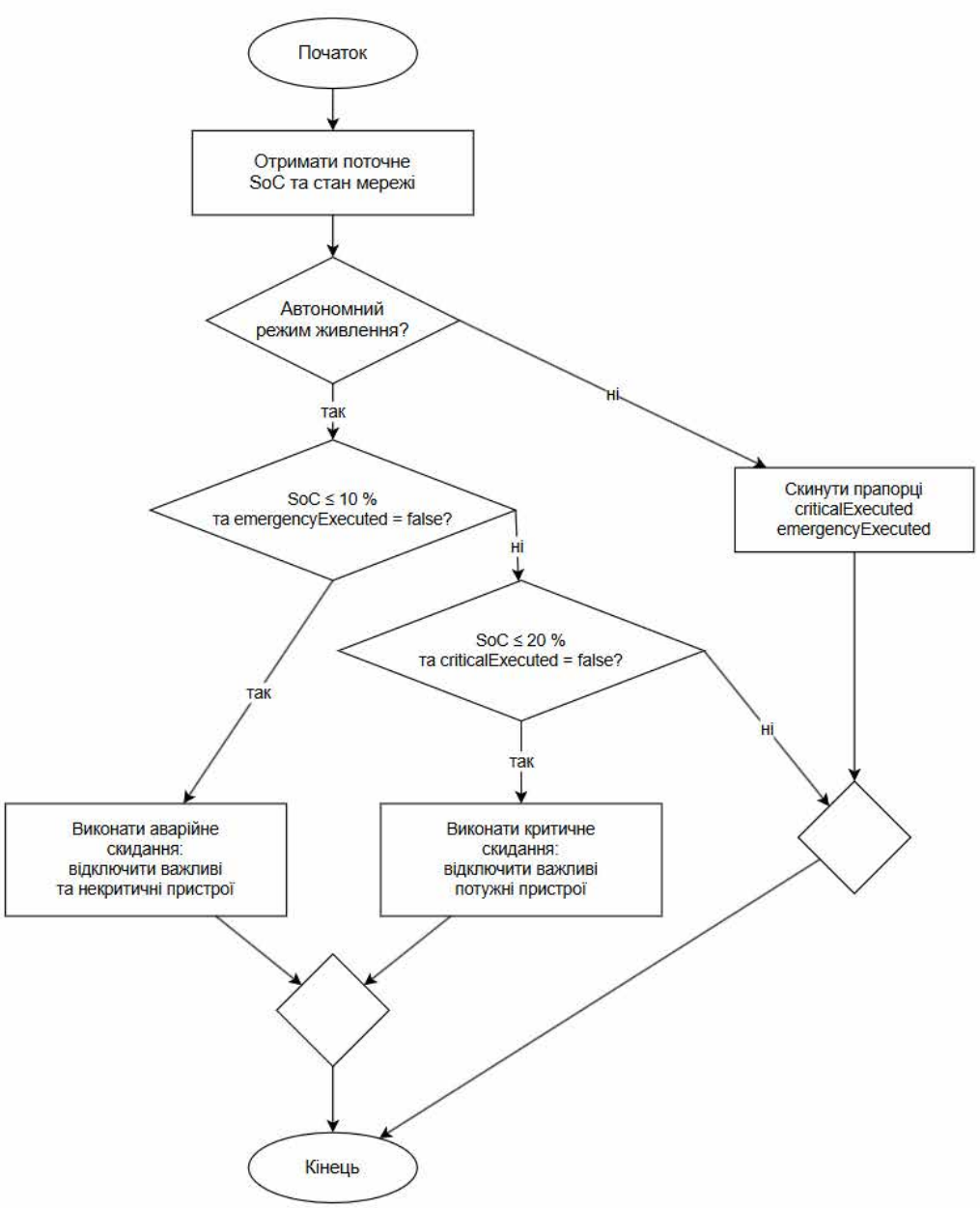


Рис. 3.1 – Блок-схема алгоритму каскадного скидання навантаження

Програмна реалізація алгоритму виконана в модулі *services/ruleEngine.js*. Лістинг 3.1 представляє фрагмент функції *handleSocProtection*, що приймає рішення про активацію відповідного рівня каскаду на основі поточного значення SoC та прапорців стану.

Лістинг 3.1 – *Фрагмент функції handleSocProtection (services/ruleEngine.js)*

```
async function handleSocProtection(userId, soc) {
const state = getState(userId);
if (soc <= C.SOC_EMERGENCY && !state.emergencyExecuted) {
```

```
console.log(`[user=${userId}] SoC ${soc}% – аварійний`);
await executeEmergencyShedding(userId);
state.emergencyExecuted = true;
state.criticalExecuted = true;
return
}
if (soc <= C.SOC_CRITICAL && !state.criticalExecuted) {
console.log(`[user=${userId}] SoC ${soc}% – критичний`);
await executeCriticalShedding(userId);
state.criticalExecuted = true;
}}
```

Перевірки йдуть від найнижчого порогу до найвищого. Після аварійного скидання (10 %) автоматично ставиться і criticalExecuted. Константи порогів – в окремому constants.js.

Сам відбір пристроїв для відключення на кожному рівні каскаду виконується відповідними функціями *executeSmartShedding*, *executeCriticalShedding* та *executeEmergencyShedding*. Логіка відбору ілюструється лістингом 3.2 на прикладі функції «розумного» скидання, що активується при подоланні першого порогу каскаду.

Лістинг 3.2 – *Функція executeSmartShedding (services/ruleEngine.js)*

```
async function executeSmartShedding(userId) {

const devices = await prisma.device.findMany({
where: {
id_user: userId,
priority_group: 'secondary',
relay_state: true,
is_interruptible: true,
nominal_power: { gt: C.POWER_THRESHOLD_W },
},});

if (devices.length === 0) return;
for (const device of devices) {
await changeDeviceState(device.id_device, device.device_name, false);
}}
```

Фільтри вибірки: поточний користувач, низький пріоритет, увімкнений стан, допустимість переривання, потужність > 50 Вт. Останній фільтр виключає пристрої з малим споживанням.

3.4.2 Алгоритм м’якого старту

Алгоритм м’якого старту відповідає за відновлення стану електропостачання побутових пристроїв після переходу зовнішньої мережі зі стану нестабільності в нормальний режим. Прямолінійне одночасне вмикання всіх раніше відключених пристроїв створює стрибок пускового струму, що перевищує пікову потужність гібридного інвертора в декілька разів. Алгоритм розв’язує цю проблему через послідовне вмикання з індивідуальними затримками та повторною перевіркою якості напруги перед кожним кроком, що захищає інвертор від повторного перевантаження у разі нестійкого відновлення мережі.

При початку аварії формується снапшот увімкнених пристроїв (devicesOnBeforeOutage). Пристрої, вимкнені вручну до аварії, у снапшот не потрапляють і не вмикаються автоматично.

Блок-схема алгоритму подана на рис. 3.2.

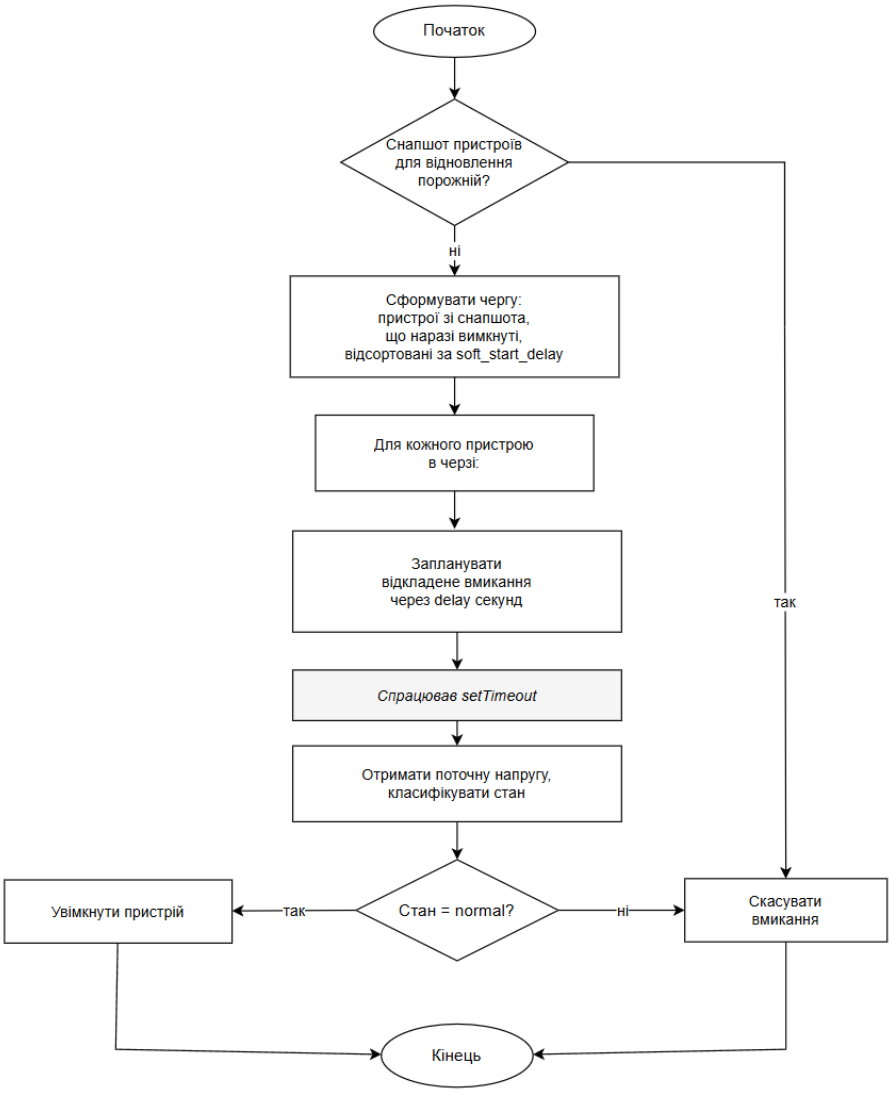


Рис. 3.2 – Блок-схема алгоритму м’якого старту

Реалізація алгоритму виконана в функції *executeSoftStart* модуля *services/ruleEngine.js*. Лістинг 3.3 представляє основну частину функції, що формує чергу пристроїв та виконує заплановане вмикання.

Лістинг 3.3 – *Фрагмент функції executeSoftStart (services/ruleEngine.js)*

```
async function executeSoftStart(userId) {
const state = getState(userId);

if (state.devicesOnBeforeOutage.size === 0) return;
const devices = await prisma.device.findMany({

where: {
id_user: userId,
id_device: { in: [...state.devicesOnBeforeOutage] },
relay_state: false,
},
```

```
orderBy: { soft_start_delay: 'asc' },
});
if (devices.length === 0) {
state.devicesOnBeforeOutage.clear();
return;
}
for (const device of devices) {
const delayMs = (device.soft_start_delay || 0) * 1000;
if (delayMs === 0) {
await changeDeviceState(device.id_device, device.device_name, true);
} else {
setTimeout(async () => {
const source = await prisma.powerSource.findUnique({
where: { id_user: userId },
});
if (classifyVoltage(source?.grid_voltage ?? 0) !== 'normal') {
console.log(`Soft Start "${device.device_name}" скасовано`);
return;}
await changeDeviceState(device.id_device, device.device_name, true);
}, delayMs);
}}
state.devicesOnBeforeOutage.clear();
}
```

Відомим обмеженням поточної реалізації є використання функції setTimeout для відкладеного вмикання: метод devicesOnBeforeOutage.clear() виконується синхронно після створення таймерів, тому при настанні нової аварії до завершення всіх відкладених callback'ів снапшот вже буде порожнім. У продуктивному розгортанні цю поведінку доцільно замінити на послідовне очікування з використанням async/await та перевіркою стану мережі між кожним кроком.

Пристрої зі снапшота сортуються за зростанням затримки: критичні (затримка 0) вмикаються першими, компресорні (тривалі паузи) – останніми. Перед кожним вмиканням з відкладеного callback'у – повторний запит напруги і classifyVoltage. При аномалії – команда скасовується (шаблон «re-check before action»).

Завершення розробки ключових модулів захисної автоматики, інформаційної бази та інструментарію забезпечення безпеки створює основу для переходу до етапу тестування та підготовки системи до експлуатації, що становить зміст розділу 4.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

Розділ 4 містить рекомендації щодо впровадження та експлуатації розробленої системи EnergyHub. У підрозділі 4.1 представлено результати функціонального тестування на прикладі двох ключових сценаріїв захисної автоматики. Підрозділ 4.2 формулює вимоги до апаратного та програмного середовища, в якому система може бути розгорнута, а також подає діаграму розгортання, що відображає цільову конфігурацію продуктивної експлуатації. Підрозділ 4.3 описує склад інсталяційного пакета порядок розгортання системи в новому середовищі.

4.1 Тестування

Тестування виконано методом «чорної скриньки» через інтерфейс мобільного клієнта демо-режим. Обрано два ключові сценарії: каскадне скидання навантаження та «м'який старт» пристроїв.

4.1.1 Тестування каскадного скидання навантаження

Метою тестового сценарію є верифікація автоматичної реакції системи на досягнення порогових значень рівня заряду батареї у режимі автономного живлення. Передумовою тестування є наявність зареєстрованого облікового запису користувача з налаштованими побутовими пристроями трьох класів критичності та активним джерелом живлення в стані відсутності зовнішньої мережі.

Початковий стан: холодильник (критичний, 150 Вт), кондиціонер (некритичний, 1500 Вт), чайник (некритичний, 2000 Вт), ПК (важливий, 450 Вт). Мережа відсутня, заряд 60 %.Скріншот початкового стану дашборду наведено на рис. 4.1.

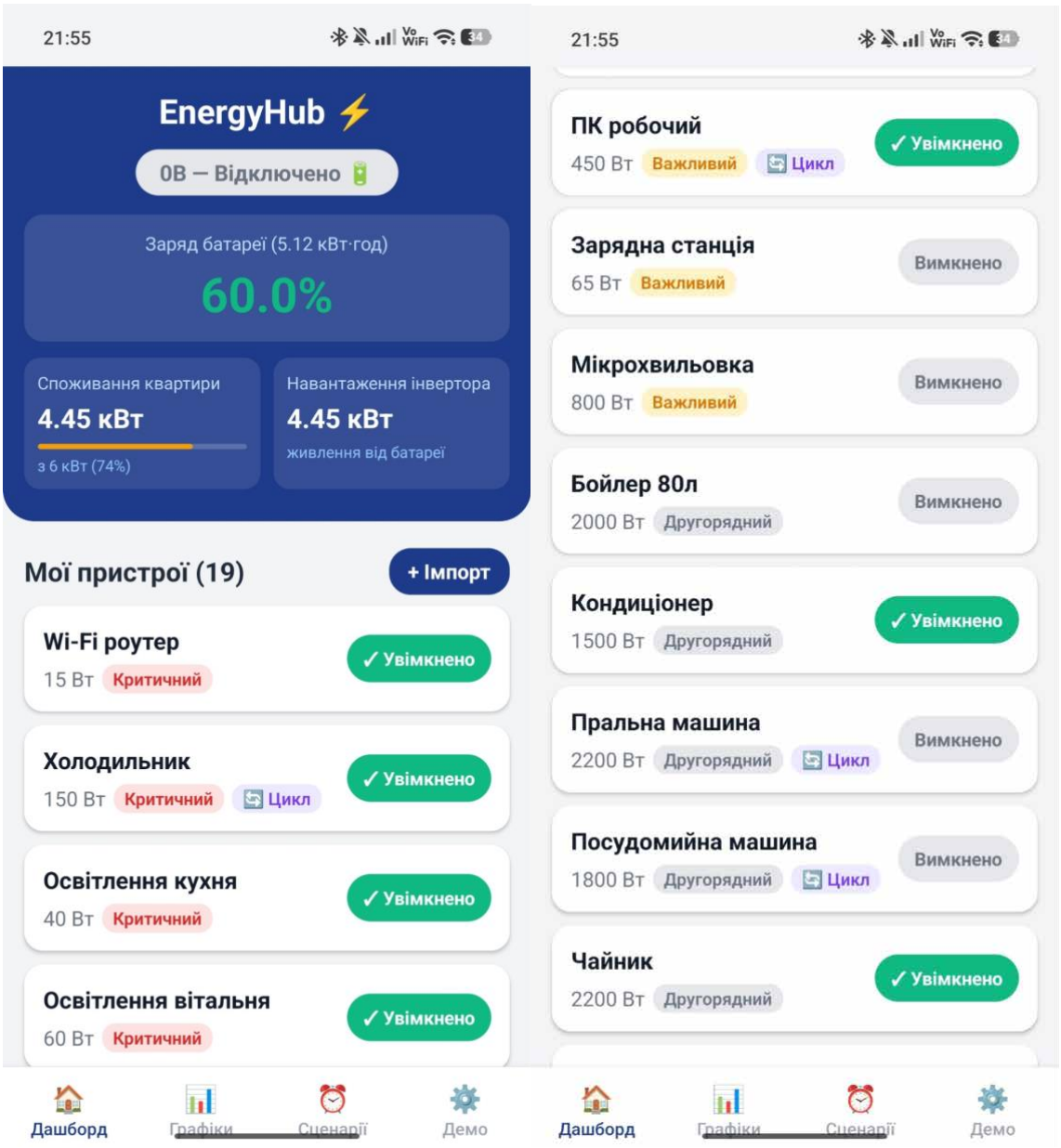


Рис. 4.1 – Початковий стан системи перед каскадним скиданням

При зниженні SoC до 49 % система відключила кондиціонер та чайник (другорядні з потужністю > 50 Вт). ПК та холодильник залишились активними.

Скріншот стану системи після спрацювання першого каскаду наведено на рис. 4.2.

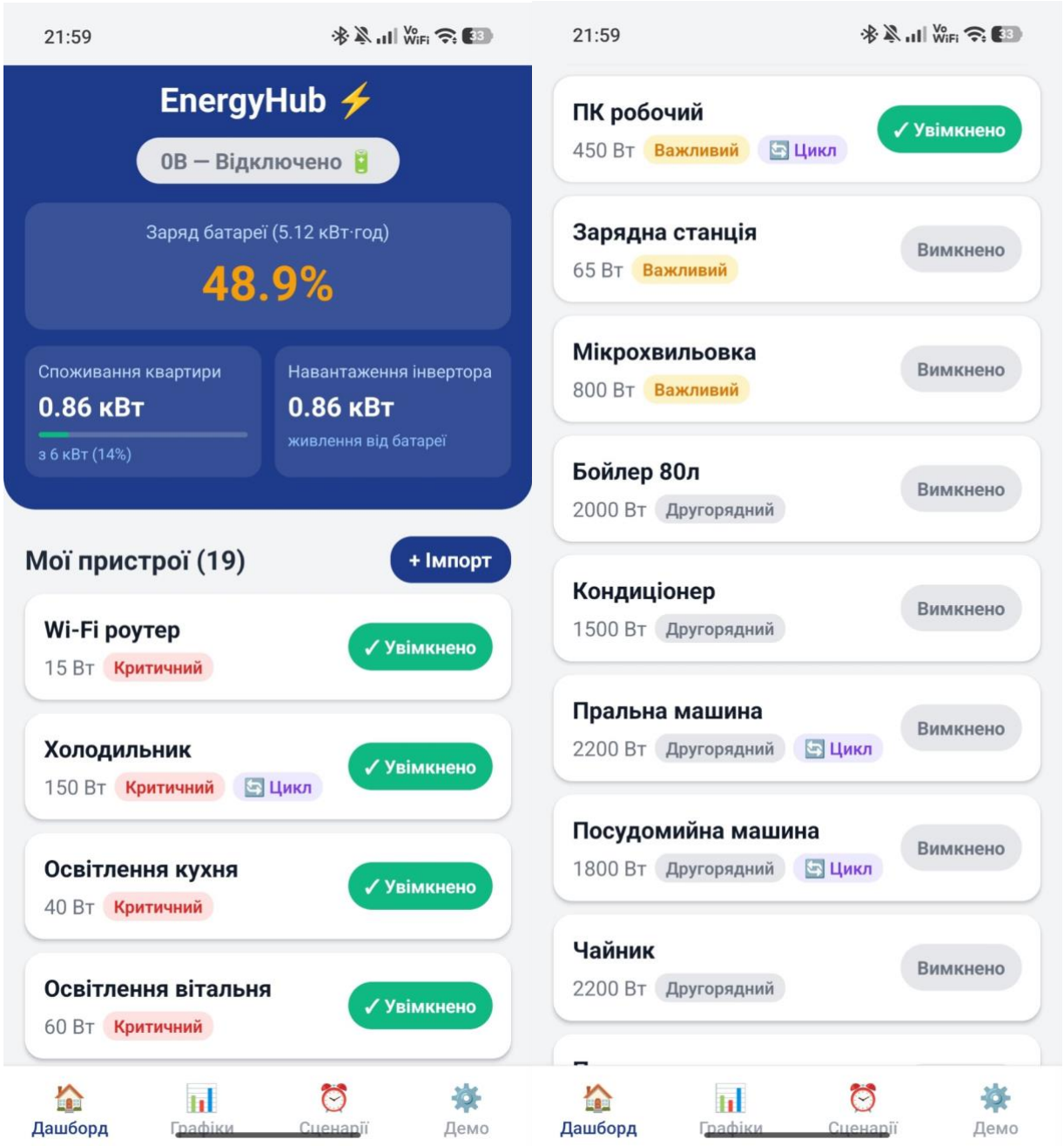


Рис. 4.2 – Стан системи після м’якого скидання навантаження

При SoC 19 % відключено ПК (важливий клас). Прапорець criticalExecuted виключив повторне спрацювання.

Скріншот стану системи наведено на рис. 4.3.

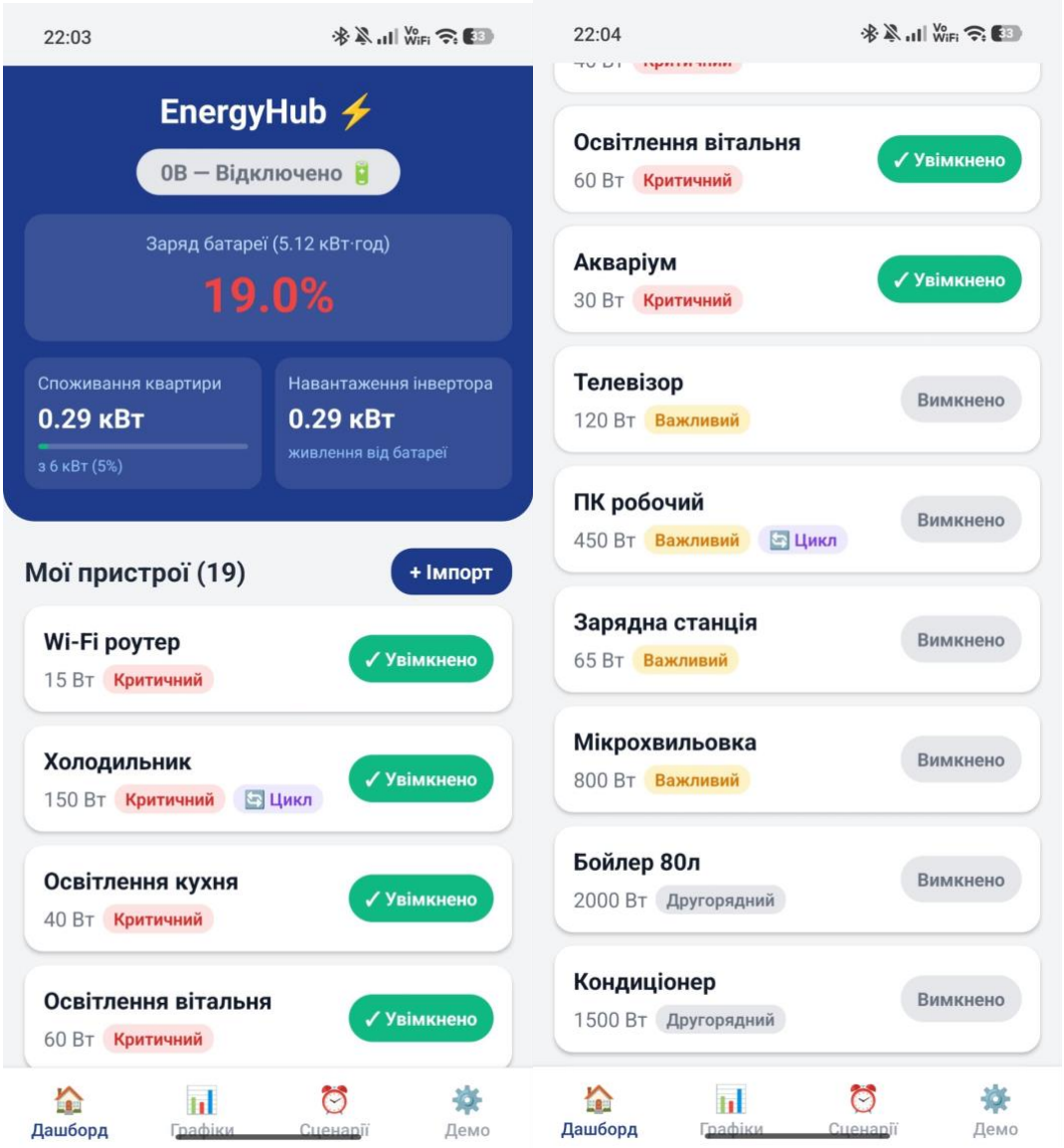


Рис. 4.3 – Стан системи після критичного скидання навантаження

При SoC 8 % відключено всі не критичні пристрої. Холодильник залишився – умова «цикл» забороняє вимикання.

Каскад відпрацював коректно для трьох рівнів. Затримка між фіксацією SoC та виконанням команд – до трьох секунд.

4.1.2 Тестування м’якого старту

Метою тестового сценарію є верифікація послідовного відновлення живлення пристроїв з повторною перевіркою якості напруги після переходу зовнішньої мережі зі стану відсутності в нормальний стан.

Початковий стан: роутер, холодильник, освітлення, акваріум, телевізор (важливий, затримка 0 с), ПК (важливий, 30 с), кондиціонер (некритичний, 90 с), чайник (некритичний, 0 с). Всі увімкнені, мережа нормальна.

Через Demo-режим вимкнено мережу. Система зафіксувала снапшот активних пристроїв та виконала перше скидання некритичних навантажень.

Через хвилину повернено мережу. Система запустила «м'який старт» – сформувала чергу з відключених пристроїв зі снапшота, запланувала вмикання з індивідуальними затримками.

Перед кожним вмиканням виконано повторну перевірку напруги. Стан normal – пристрій увімкнувся. Скріншот стану системи після завершення м’якого старту наведено на рис. 4.4.

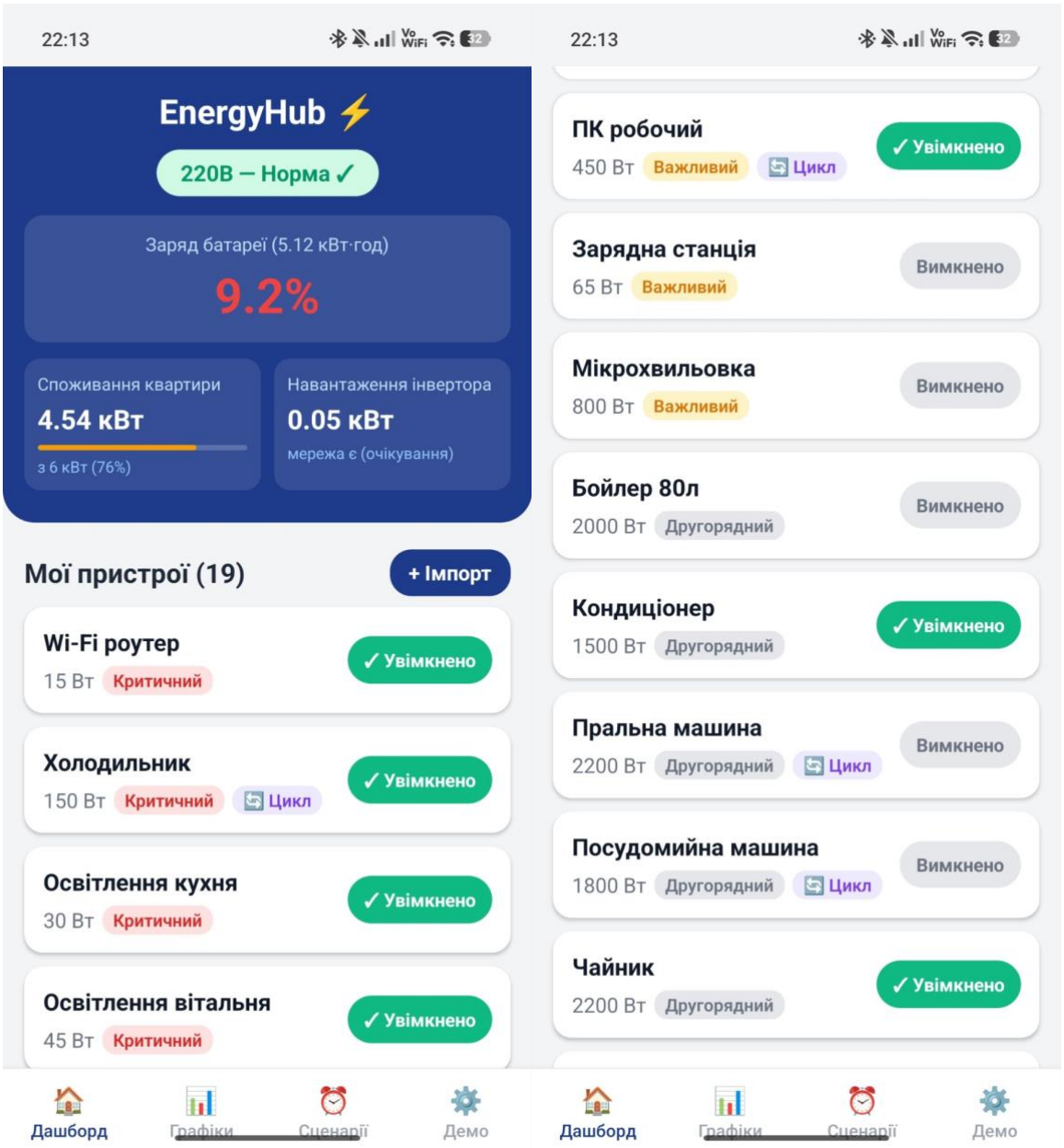


Рис. 4.4 – Стан системи після завершення м’якого старту

Висновок щодо тесту. Сценарій м’якого старту відпрацював коректно як у штатному, так і в граничному випадку нестійкого відновлення мережі. Послідовність вмикань відповідає сортуванню за зростанням індивідуальної затримки. Захисний шаблон повторної перевірки якості напруги виключає виконання потенційно небезпечних дій на основі застарілих даних. Результат відповідає очікуванню функціональної вимоги про «м'який старт», сформульованої в підрозділі 1.5.

4.1.3 Загальні результати тестування

За результатами виконання двох сценаріїв функціонального тестування підтверджено відповідність розробленої системи ключовим функціональним вимогам, що визначають специфіку предметної області. Каскадне скидання навантаження за рівнем заряду батареї та «м'який старт» пристроїв з повторною перевіркою якості напруги є відмінними характеристиками EnergyHub порівняно з універсальними платформами розумного будинку (Tuya Smart, Home Assistant) та фірмовими додатками виробників енергетичного обладнання (EcoFlow App, Solax Cloud). Тестування підтверджує, що сформульована у підрозділі 1.5 мета розробки досягнута на рівні базової функціональності, реалізованої в межах бакалаврської роботи.

4.2 Вимоги до апаратного та програмного забезпечення

Розробка EnergyHub виконана з орієнтацією на сучасні відкриті технології, що знижує вимоги до апаратного та програмного середовища експлуатації. Серверна частина системи розгортається у вигляді трьох контейнеризованих процесів – Express-сервер, СУБД PostgreSQL [9], MQTT-брокер Mosquitto [11] – що можуть запускатися як на одній фізичній чи віртуальній машині, так і на окремих вузлах інфраструктури в залежності від навантаження та вимог до надійності.

Вимоги до серверної частини. Мінімальна конфігурація серверного вузла, придатна для побутового сценарію експлуатації (один користувач з кількома десятками побутових пристроїв), включає:

- операційна система – Linux Ubuntu 22.04 LTS або новіша, macOS 14 або новіша, Windows 11;
- процесор – двоядерний x86-64 або ARM64 з тактовою частотою від 1,8 ГГц;
- оперативна пам’ять – 2 ГБ, з яких приблизно 800 МБ використовується серверним процесом та СУБД у штатному режимі;

- дисковий простір – 10 ГБ для системи, контейнерів та накопиченої історії телеметрії за рік експлуатації;
- мережа – Ethernet або Wi-Fi із стабільним з’єднанням до локального MQTT-брокера та доступом в Інтернет для отримання графіків відключень від DTEK Yasno.

Рекомендована конфігурація для багатокористувацького сценарію (декілька десятків облікових записів у межах однієї інсталяції) – чотириядерний процесор, 4 ГБ оперативної пам’яті, SSD-накопичувач об’ємом 50 ГБ. Програмне забезпечення серверного вузла включає середовище виконання Node.js версії 20 LTS або новішої, реляційну СУБД PostgreSQL версії 15 або новішої, MQTT-брокер Eclipse Mosquitto версії 2.0 або новіший, систему контейнеризації Docker [12] версії 24 або новішу разом з утилітою оркестрації Docker Compose.

Вимоги до клієнтського додатка. Мобільний застосунок EnergyHub підтримує дві платформи з єдиної кодової бази React Native. Мінімальні вимоги до мобільного пристрою:

- для платформи Android – операційна система версії 7.0 (API 24) або новіша, мінімум 2 ГБ оперативної пам’яті, 100 МБ вільного простору в пам’яті пристрою;
- для платформи iOS – операційна система iOS 13.0 або новіша, апаратна платформа iPhone 8 або новіший, 100 МБ вільного простору.

Мобільний пристрій взаємодіє з серверною частиною виключно через REST API за протоколом HTTPS, тому фізичне розташування пристрою не має значення за умови наявності мережевого доступу до сервера. У сценаріях з мобільним зв’язком (передача даних через стільникову мережу) очікувані обсяги трафіку не перевищують 1 МБ на годину штатної роботи додатка, що робить систему придатною для експлуатації навіть у місцевостях з обмеженою пропускною здатністю.

Вимоги до обладнання енергоустановки. Цільова конфігурація системи в реальному середовищі експлуатації передбачає підключення до фізичного

гібридного інвертора, що публікує телеметрію за протоколом MQTT, та смарт-розеток з підтримкою керування через відповідну хмару або локальний шлюз. До підтримуваних виробників інверторів належать Qoltec (модель MaxFlow), Solax Power, Deye, Must, а також інші виробники, що реалізують стандартний MQTT-інтерфейс. До підтримуваних розеток належать пристрої з логотипом Tuya Smart, а також деякі моделі Sonoff та Shelly, що працюють через MQTT. На етапі розробки замість фізичного обладнання використовувався програмний симулятор інвертора Qoltec, що публікує телеметричні пакети в локальний MQTT-брокер з тією самою структурою повідомлень, що очікується від реального обладнання.

Діаграма розгортання. Цільова конфігурація системи у виробничому середовищі експлуатації передбачає три фізичні вузли: домашній сервер користувача, що виконує контейнеризовані процеси серверної частини та СУБД; мобільний пристрій користувача з встановленим клієнтським додатком; зовнішнє обладнання енергоустановки, що включає гібридний інвертор та смарт-розетки. Графічне представлення діаграми розгортання подано на рис. 4.5.

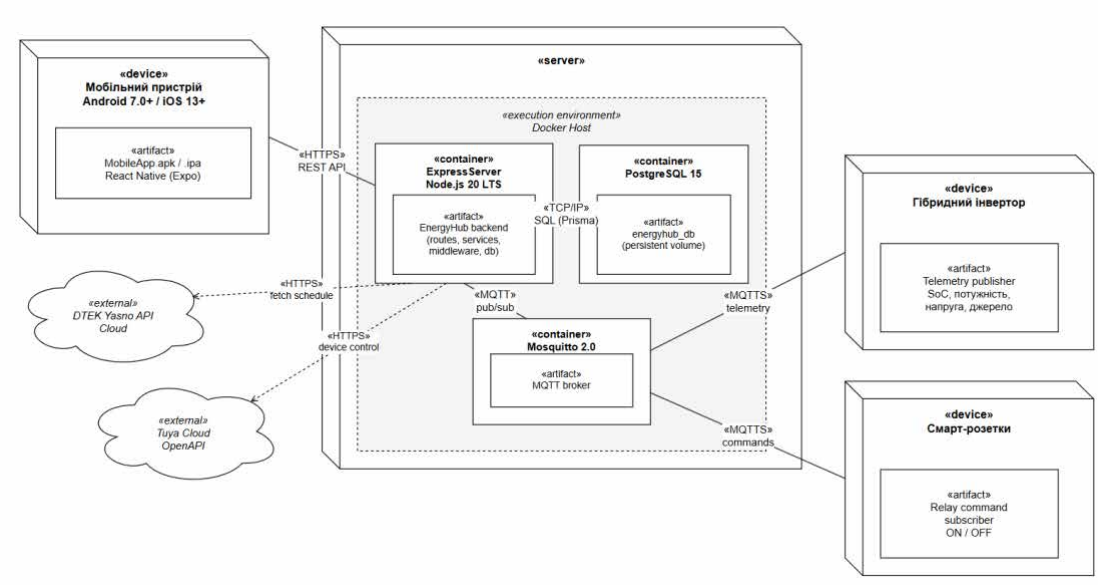


Рис. 4.5 – Діаграма розгортання системи EnergyHub у виробничому середовищі

На діаграмі вузли представлено стереотипом *«device»* для фізичних пристроїв та *«execution environment»* для контейнеризованих процесів. Зв’язки

між вузлами підписані протоколами обміну: HTTPS для каналу мобільний застосунок – серверна частина, TCP/IP для внутрішнього з’єднання серверного процесу з СУБД, MQTT для взаємодії з MQTT-брокером, MQTT-S (MQTT over TLS) для обміну з обладнанням, HTTPS для звернень до зовнішніх API постачальника DTEK Yasno та хмари Tuya.

4.3 Склад інсталяційного пакета

Інсталяційний пакет системи EnergyHub логічно розділений на користувацький дистрибутив (мобільний застосунок) та серверне оточення. Завдяки хмарній архітектурі системи, розповсюдження клієнтської частини здійснюється у вигляді повністю скомпільованого та готового до використання продукту, що не вимагає від кінцевого користувача навичок програмування чи конфігурації підключень.

Склад клієнтської частини. Для кінцевого користувача інсталяційний пакет представляє собою єдиний файл мобільного застосунку формату APK (Android Package Kit) – EnergyHub.apk. У цей інсталяційний файл запаковано:

- оптимізований та скомпільований код користувацького інтерфейсу (дашборд, графіки, сценарії, налаштування пристроїв);
- графічні ресурси застосунку (іконки, splash-екран);
- інтегрований HTTP-клієнт, у конфігурації якого на етапі збірки (build) жорстко задано URL-адресу віддаленого загальнодоступного сервера. Завдяки цьому застосунок автоматично встановлює з'єднання з бекендом одразу після першого запуску.

Склад серверної частини. Серверний компонент системи вже розгорнутий на віддаленому загальнодоступному хостингу і функціонує в автономному режимі. Його архітектура включає: – контейнеризовану інфраструктуру (база даних PostgreSQL та MQTT-брокер Eclipse Mosquitto), розгорнуту за допомогою конфігурації Docker Compose; – виконуваний процес бекенду на базі середовища Node.js, який обробляє REST API запити від

мобільного додатка та містить модулі бізнес-логіки (ruleEngine.js, scenarioEngine.js, yasnoService.js тощо); – програмний імітатор телеметрії (simulator.js), що генерує показники роботи гібридного інвертора та публікує їх у MQTT-топіки для тестування та демонстрації роботи алгоритмів.

Документація. У комплекті постачається посібник користувача (README.md), який описує функціонал системи, та історія версій (CHANGELOG.md).

Розгортання системи. Оскільки серверна інфраструктура розгорнута заздалегідь, процес встановлення системи для кінцевого користувача зведено до мінімуму і складається з трьох кроків:

1. завантаження файлу EnergyHub.apk на цільовий мобільний пристрій під управлінням ОС Android;
2. надання операційній системі дозволу на встановлення застосунків з невідомих джерел (якщо це вимагається політикою безпеки пристрою);
3. встановлення та запуск програми. Синхронізація з базою даних та отримання актуальної телеметрії відбуваються автоматично при відкритті головного екрана.

ВИСНОВКИ

У першому розділі бакалаврської кваліфікаційної роботи виконано системний аналіз предметної області – програмних систем моніторингу та оптимізації енергоспоживання в умовах гібридного електропостачання для «розумних» будинків. Проаналізовано специфіку експлуатації побутових енергоустановок в Україні з урахуванням планових та аварійних відключень електромережі, нестабільної якості напруги та обмежень пікової потужності гібридних інверторів. Сформульовано перелік функціональних та нефункціональних вимог, що визначають клас програмних рішень для предметної області. Виконано порівняльний аналіз чотирьох наявних на ринку рішень – Tuya Smart, Home Assistant, EcoFlow App та Solax Cloud – що дозволив встановити прогалину у функціональному покритті: жоден з оглянутих представників не реалізує одночасно класифікації стану напруги, каскадного відключення за рівнем заряду батареї, м’якого старту пристроїв та превентивної реакції на графік відключень. Сформульовано постановку задачі розробки нового програмного забезпечення, що поєднує функції моніторингу енергетичного стану та активного керування побутовими навантаженнями в межах єдиної апаратно-незалежної архітектури.

У другому розділі спроєктовано архітектуру програмної системи EnergyHub та структуру баз даних. Побудовано логічну модель даних у вигляді ER-діаграми з обґрунтуванням відповідності третій нормальній формі та обрано реляційну СУБД PostgreSQL як платформу зберігання. Розроблено доменну модель системи у вигляді UML-діаграми класів з виділенням семи класів-сутностей та чотирьох перелічуваних типів, з подальшою декомпозицією на три кооперації за функціональною ознакою – моніторинг та зберігання телеметрії, захисну автоматику, виконання сценаріїв та обробку графіків відключень. Побудовано діаграму пакетів, що формалізує чотиришарову архітектуру системи з однонаправленими залежностями зверху вниз від рівня представлення до

зовнішніх сервісів. Завершено етап проєктування діаграмою компонентів, яка відображає фізичну організацію виконуваних артефактів та зовнішні інтерфейси системи.

У третьому розділі реалізовано програмну частину системи. Серверна частина створена з використанням платформи Node.js, фреймворку Express та об’єктно-реляційного відображення Prisma, що в сукупності забезпечує асинхронну модель виконання та типобезпечний доступ до даних. Розроблено інформаційну базу в СУБД PostgreSQL зі складеними індексами для оптимізації типових запитів історичних графіків. Спроєктовано механізм JWT-автентифікації, ізоляції персональних даних та логування значущих подій. Реалізовано клієнтський мобільний застосунок засобами React Native у середовищі Expo з єдиною кодовою базою для платформ Android та iOS. Окремо реалізовано модуль захисної автоматики, що містить два ключові алгоритми предметної області – каскадне скидання навантаження за рівнем заряду батареї з трирівневою моделлю пріоритетів та «м'який старт» пристроїв з повторною перевіркою якості напруги перед кожним кроком вмикання.

У четвертому розділі виконано функціональне тестування розробленої системи методом «чорної скриньки» через інтерфейс мобільного клієнта та засоби демонстраційного режиму. Перевірено коректність роботи каскадного скидання навантаження для всіх трьох порогів спрацювання (50 %, 20 %, 10 %) та м’якого старту пристроїв у штатному й граничному випадку нестійкого відновлення мережі. Сформульовано вимоги до апаратного та програмного середовища експлуатації, що включають мінімальну та рекомендовану конфігурації серверного вузла, вимоги до мобільних платформ Android та iOS, а також перелік підтримуваного обладнання енергоустановки. Побудовано діаграму розгортання системи у виробничому середовищі. Описано склад інсталяційного пакета.

У результаті виконання бакалаврської кваліфікаційної роботи розроблено повноцінне програмне забезпечення EnergyHub для моніторингу та оптимізації енергоспоживання в умовах гібридного електропостачання, що відповідає

сформульованій у вступі меті – автоматизованому захисту побутового обладнання, раціональному витрачанню ресурсу акумуляторної батареї та забезпеченню користувача засобами спостереження за станом енергетичної установки. Сукупність дев’яти функціональних можливостей системи (збір та зберігання телеметрії, класифікація стану електромережі, каскадне відключення за рівнем заряду, «м'який старт» пристроїв, захист від перевищення піка потужності інвертора, превентивна реакція на графік відключень, ручне керування пристроями, користувацькі сценарії автоматизації, візуалізація стану та історії) разом з чотирма нефункціональними характеристиками (апаратна незалежність, швидкодія реагування, кросплатформеність клієнта, захищеність каналу обміну даними) формує конкурентну позицію системи відносно наявних на ринку рішень. Перспективи подальшого розвитку проєкту включають реалізацію передбачених архітектурою, але не охоплених у межах бакалаврської роботи функцій – повноцінних push-сповіщень через FCM/APNS, інтеграції з фізичним обладнанням Tuya та Qoltec через відповідні програмні інтерфейси та формування рекомендацій користувачу щодо оптимізації споживання на основі статистики тривалої експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. НЕК «Укренерго». Про оператора системи передачі. URL: https://ua.energy/ (дата звернення: 12.05.2026).
2. EN 50160:2010/A3:2019. Voltage characteristics of electricity supplied by public electricity networks. Brussels : CENELEC, 2019. 42 p.
3. Wang J., Liu P., Hicks-Garner J. et al. Cycle-life model for graphite-LiFePO4 cells. Journal of Power Sources. 2011. Vol. 196, Iss. 8. P. 3942–3948.
4. Tuya Smart – IoT Development Platform. Tuya Inc. URL: https://developer.tuya.com/ (дата звернення: 10.05.2026).
5. Home Assistant – Open source home automation. URL: https://www.home-assistant.io/ (дата звернення: 10.05.2026).
6. Node.js Documentation. Node.js Foundation. URL: https://nodejs.org/en/docs (дата звернення: 10.05.2026).
7. Express.js – Fast, unopinionated, minimalist web framework for Node.js. OpenJS Foundation. URL: https://expressjs.com/ (дата звернення: 10.05.2026).
8. Prisma Documentation. Prisma Data, Inc. URL: https://www.prisma.io/docs (дата звернення: 10.05.2026).
9. PostgreSQL 15 Documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/15/ (дата звернення: 10.05.2026).
10. Banks A., Gupta R. MQTT Version 3.1.1. OASIS Standard, 29 October 2014. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (дата звернення: 10.05.2026).
11. Eclipse Mosquitto – An open source MQTT broker. Eclipse Foundation. URL: https://mosquitto.org/ (дата звернення: 10.05.2026).
12. Docker Documentation. Docker, Inc. URL: https://docs.docker.com/ (дата звернення: 10.05.2026).

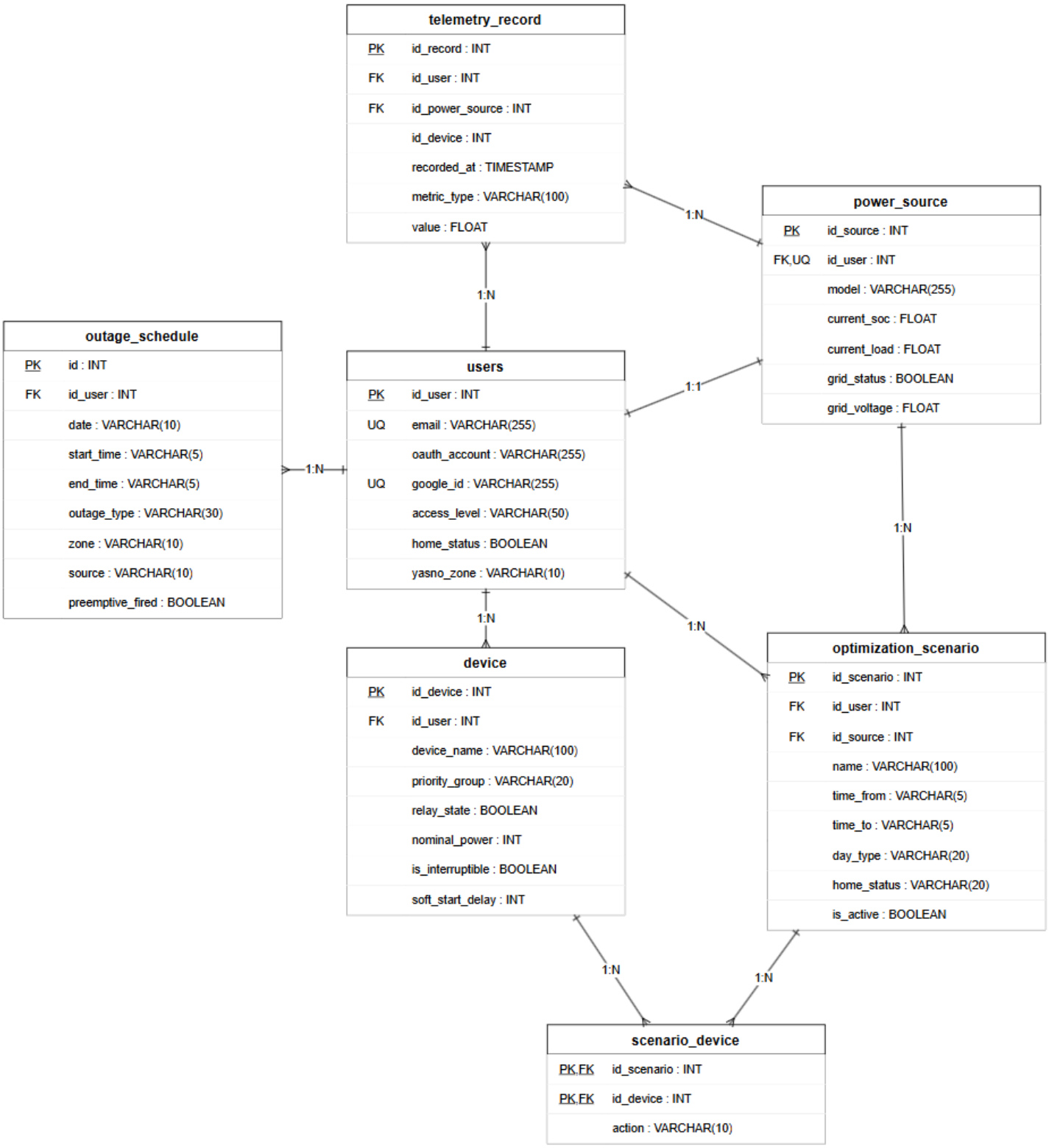
13. React Native Documentation. Meta Platforms, Inc. URL: https://reactnative.dev/docs/getting-started (дата звернення: 10.05.2026).
14. Expo Documentation. Expo, Inc. URL: https://docs.expo.dev/ (дата звернення: 10.05.2026).
15. Стогній Б. С., Кириленко О. В., Праховник А. В., Денисюк С. П. Інтелектуальні електричні мережі: світовий досвід та перспективи України. Праці Інституту електродинаміки НАН України. 2012. Вип. 32. С. 5–20.
16. Денисюк С. П., Котсар О. В. Smart Grid як концепція модернізації систем електропостачання. Енергетика та електрифікація. 2014. № 7. С. 21–29.
17. DTEK Yasno – Графік відключень електроенергії. ДТЕК. URL: https://kyiv.yasno.com.ua/schedule-turn-off-electricity (дата звернення: 12.05.2026).
18. Кириленко О. В., Басок Б. І., Базюк Т. М. Розвиток систем акумулювання енергії в Україні: стан та перспективи. Вісник НАН України. 2021. № 9. С. 50–63.
19. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2004. 175 p.
20. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River : Prentice Hall, 2004. 703 p.
21. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education, 2016. 810 p.
22. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
23. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley, 2021. 456 p.
24. OAuth 2.0 Authorization Framework. RFC 6749. Internet Engineering Task Force, October 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 10.05.2026).

25. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection – Information security management systems – Requirements. Geneva : ISO, 2022. 26 p.
26. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force, May 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 10.05.2026).

ДОДАТКИ

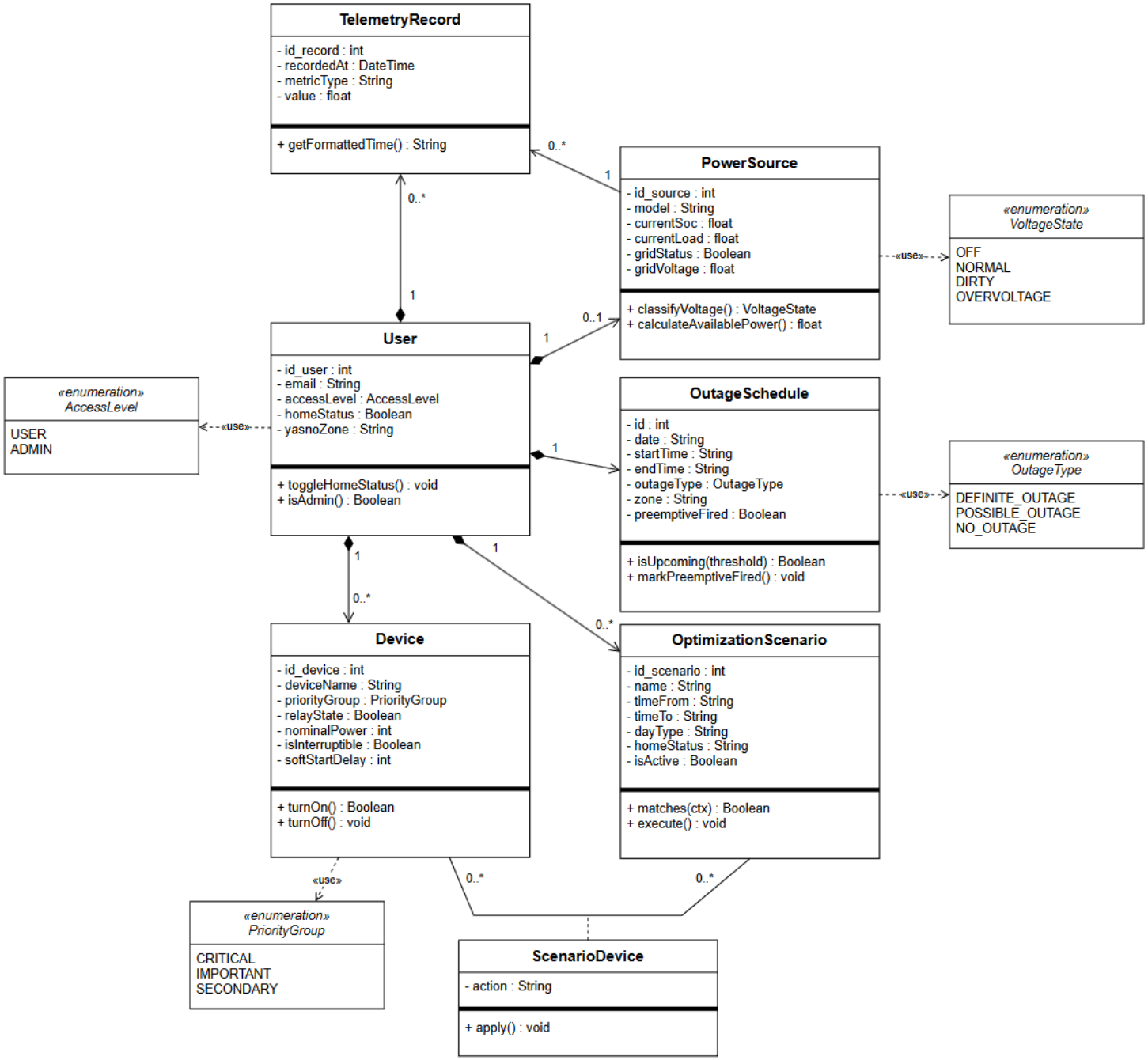
Додаток А

ER-діаграма інформаційної бази системи



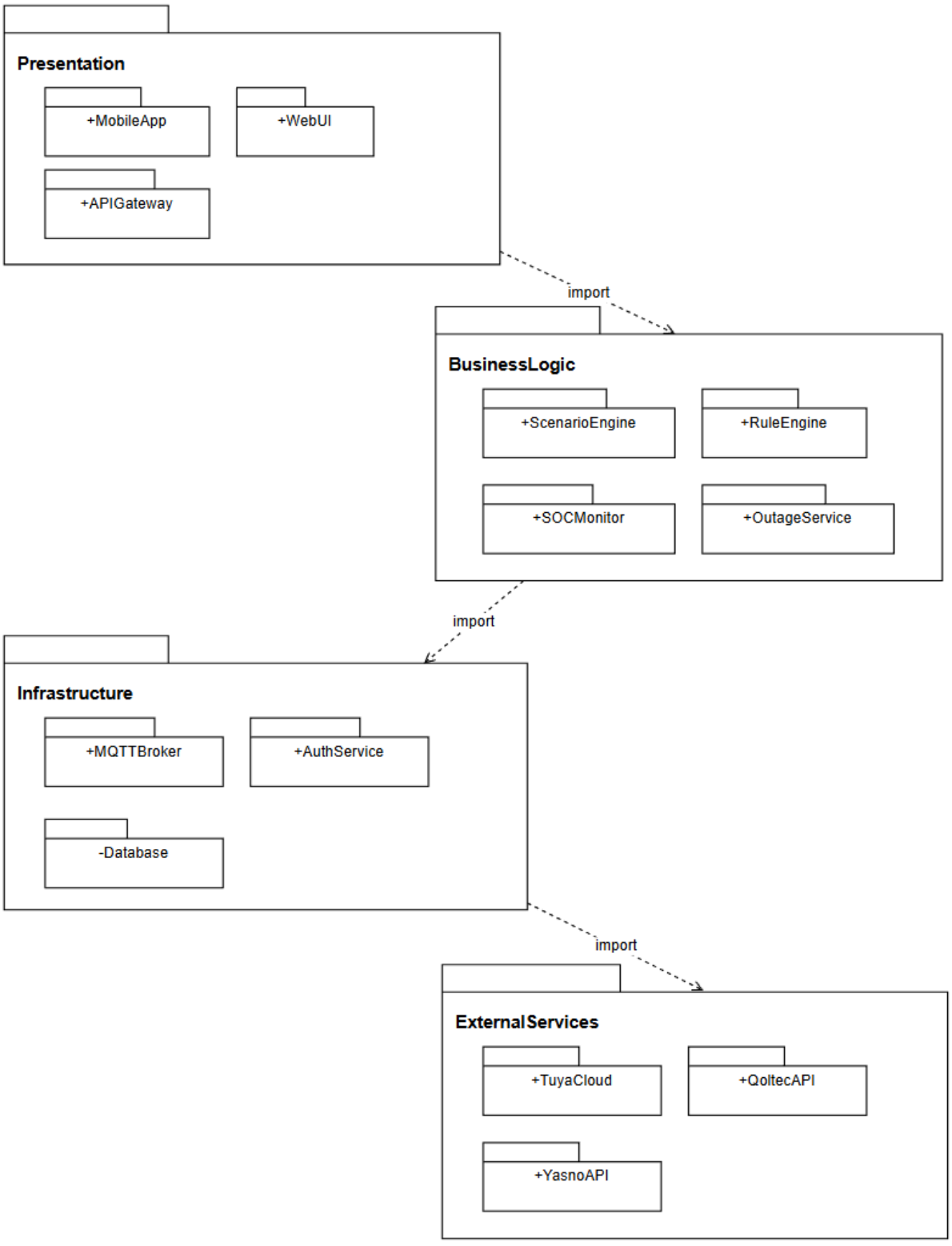
Додаток Б

Загальна діаграма класів



Додаток В

Діаграма пакетів системи



Додаток Д

Лістинг ключових програмних модулів

У цьому додатку наведено фрагменти коду ключових програмних модулів системи EnergyHub, що демонструють реалізацію основних алгоритмів захисної автоматики: каскадного скидання навантаження за рівнем заряду акумуляторної батареї, м'якого старту пристроїв з повторною перевіркою якості напруги, класифікації стану зовнішньої електромережі, двигуна користувацьких сценаріїв автоматизації та превентивної реакції на графік планових відключень DTEK Yasno [17].

Д.1 Алгоритм каскадного скидання навантаження

Модуль ruleEngine.js реалізує трирівневе каскадне скидання навантаження. Функція handleSocProtection перевіряє поточний рівень заряду батареї та активує відповідний рівень каскаду: при SoC ≤ 50 % вимикаються другорядні прилади (executeSmartShedding), при SoC ≤ 20 % – важливі (executeCriticalShedding), при SoC ≤ 10 % – аварійне скидання всіх некритичних навантажень (executeEmergencyShedding). Прапорці criticalExecuted та emergencyExecuted у стані користувача запобігають повторному спрацюванню.

```
async function handleSocProtection(userId, soc) {
    const state = getState(userId);

    if (soc <= C.SOC_EMERGENCY && !state.emergencyExecuted) {
        console.log(`[user=${userId}] SoC ${soc}% – аварійний`);
        await executeEmergencyShedding(userId);
        state.emergencyExecuted = true;
        state.criticalExecuted  = true;
        return;
    }

    if (soc <= C.SOC_CRITICAL && !state.criticalExecuted) {
        console.log(`[user=${userId}] SoC ${soc}% – критичний`);
        await executeCriticalShedding(userId);
        state.criticalExecuted = true;
    }
}
```

```
async function executeSmartShedding(userId) {
    const devices = await prisma.device.findMany({
        where: {
            id_user:          userId,
            priority_group:   'secondary',
            relay_state:      true,
            is_interruptible: true,
            nominal_power:    { gt: C.POWER_THRESHOLD_W },
        },
    });
    if (devices.length === 0) return;
    for (const device of devices) {
        await changeDeviceState(device.id_device, device.device_name, false);
    }
}

async function executeCriticalShedding(userId) {
    const devices = await prisma.device.findMany({
        where: {
            id_user:        userId,
            priority_group: 'important',
            relay_state:    true,
            nominal_power:  { gt: C.POWER_THRESHOLD_W },
        },
    });
    for (const d of devices) {
        await changeDeviceState(d.id_device, d.device_name, false);
    }
}

async function executeEmergencyShedding(userId) {
    const devices = await prisma.device.findMany({
        where: {
            id_user:        userId,
            priority_group: { in: ['important', 'secondary'] },
            relay_state:    true,
        },
    });
    for (const d of devices) {
        await changeDeviceState(d.id_device, d.device_name, false);
    }
}
```

Д.2 Алгоритм м'якого старту пристроїв

При втраті зовнішнього живлення функція handleOutageStart формує снапшот поточних активних пристроїв у множині devicesOnBeforeOutage. При відновленні якісної напруги функція executeSoftStart формує чергу зі снапшота, сортовану за зростанням індивідуальної затримки. Пристрої з нульовою затримкою вмикаються негайно; решта – через setTimeout з повторною перевіркою classifyVoltage перед кожним увімкненням. Якщо напруга на момент виконання callback не є нормальною, команда скасовується.

```
async function handleOutageStart(userId) {
    const state = getState(userId);
    state.criticalExecuted  = false;
    state.emergencyExecuted = false;
```

```
    const activeDevices = await prisma.device.findMany({
        where:  { id_user: userId, relay_state: true },
        select: { id_device: true, device_name: true },
    });
    state.devicesOnBeforeOutage = new Set(
        activeDevices.map(d => d.id_device)
    );
    await executeSmartShedding(userId);
}

async function executeSoftStart(userId) {
    const state = getState(userId);
    if (state.devicesOnBeforeOutage.size === 0) return;

    const devices = await prisma.device.findMany({
        where: {
            id_user:     userId,
            id_device:   { in: [...state.devicesOnBeforeOutage] },
            relay_state: false,
        },
        orderBy: { soft_start_delay: 'asc' },
    });
    if (devices.length === 0) {
        state.devicesOnBeforeOutage.clear();
        return;
    }

    for (const device of devices) {
        const delayMs = (device.soft_start_delay || 0) * 1000;
        const id   = device.id_device;
        const name = device.device_name;

        if (delayMs === 0) {
            await changeDeviceState(id, name, true);
        } else {
            setTimeout(async () => {
                const source = await prisma.powerSource.findUnique({
                    where: { id_user: userId },
                });
                const v = source?.grid_voltage ?? 0;
                if (classifyVoltage(v) !== 'normal') {
                    console.log(`Soft Start "${name}" скасовано – ${v}В`);
                    return;
                }
                await changeDeviceState(id, name, true);
            }, delayMs);
        }
    }
    state.devicesOnBeforeOutage.clear();
}
```

Д.3 Класифікація стану напруги та диспетчеризація подій

Функція classifyVoltage повертає один з чотирьох дискретних станів напруги (off, normal, dirty, overvoltage) на основі порогових значень, визначених у модулі constants.js. Функція processTelemetry є точкою входу для обробки кожного пакета телеметрії: вона визначає поточний стан, порівнює з попереднім та делегує виконання відповідному обробнику – handleOutageStart при переході

в аварійний режим або handleGridRestore при нормалізації мережі. Функція changeDeviceState інкапсулює оновлення стану реле пристрою в базі даних.

```
// Файл: constants.js – порогові значення класифікації напруги
module.exports = {
    VOLTAGE_MIN_NORMAL:   200,   // нижня межа норми, В
    VOLTAGE_MAX_NORMAL:   245,   // верхня межа норми, В
    VOLTAGE_OVERVOLTAGE:  250,   // поріг перенапруги, В
    SOC_CRITICAL:          20,   // поріг критичного SoC, %
    SOC_EMERGENCY:         10,   // поріг аварійного SoC, %
    POWER_THRESHOLD_W:     50,   // мінімальна потужність для скидання, Вт
    MAX_INVERTER_KW:      6.0,   // пікова потужність інвертора, кВт
    BATTERY_CAPACITY_KWH: 5.12,  // ємність батареї, кВт·год
    // ...
};

// Файл: services/ruleEngine.js – класифікація стану напруги
function classifyVoltage(v) {
    if (v === 0)                                          return 'off';
    if (v >= C.VOLTAGE_MIN_NORMAL && v <= C.VOLTAGE_MAX_NORMAL) return 'normal';
    if (v > C.VOLTAGE_OVERVOLTAGE)                        return 'overvoltage';
    return 'dirty';
}

async function processTelemetry(userId, soc, gridStatus, voltage) {
    const state        = getState(userId);
    const currentState = classifyVoltage(voltage);

    const wasOnGrid = state.previousState === 'normal';
    const isOnGrid  = currentState        === 'normal';

    if (wasOnGrid && !isOnGrid) {
        await handleOutageStart(userId);
    }

    if (!wasOnGrid && isOnGrid) {
        await handleGridRestore(userId, voltage);
    }

    if (!isOnGrid) {
        await handleSocProtection(userId, soc);
    } else {
        state.criticalExecuted  = false;
        state.emergencyExecuted = false;
    }

    state.previousState = currentState;
}

async function changeDeviceState(deviceId, deviceName, targetState) {
    try {
        await prisma.device.update({
            where: { id_device: deviceId },
            data:  { relay_state: targetState },
        });
    } catch (error) {
        console.error(`"${deviceName}":`, error.message);
    }
}
```

Д.4 Двигун користувацьких сценаріїв автоматизації

Модуль scenarioEngine.js реалізує періодичне виконання користувацьких сценаріїв. Функція runMinuteTick викликається кожну хвилину та ітерує по всіх зареєстрованих користувачах. Для кожного користувача функція checkTimeScenarios перевіряє активні сценарії на відповідність поточному часу, типу дня та режиму присутності. При збігу умов функція executeScenario застосовує дії сценарію до пов'язаних пристроїв з попередньою перевіркою якості напруги для операцій увімкнення.

```
async function runMinuteTick() {
    const effectiveHour  = getEffectiveHour();
    const currentTimeStr = getCurrentTimeString();
    const now            = new Date();
    const dayType = (now.getDay() === 0 || now.getDay() === 6)
        ? 'weekend' : 'weekday';


    if (effectiveHour !== lastClearHour) {
        executedThisHour.clear();
        lastClearHour = effectiveHour;
    }


    const users = await prisma.user.findMany({});
    for (const user of users) {
        await checkTimeScenarios(user, currentTimeStr, dayType);
    }
}


async function checkTimeScenarios(user, currentTime, dayType) {
    const scenarios = await prisma.optimizationScenario.findMany({
        where:   { is_active: true, id_user: user.id_user },
        include: { devices: { include: { device: true } } },
    });
    if (scenarios.length === 0) return;


    for (const scenario of scenarios) {
        if (scenario.day_type !== 'any' && scenario.day_type !== dayType)
            continue;
        if (scenario.home_status !== 'any') {
            if (scenario.home_status === 'home' && !user.home_status)
                continue;
            if (scenario.home_status === 'away' && user.home_status)
                continue;
        }


        const fromTime = normalizeTime(scenario.time_from);
        const key      = `${scenario.id_scenario}_${currentTime}`;


        if (currentTime === fromTime && !executedThisHour.has(key)) {
            await executeScenario(user.id_user, scenario);
            executedThisHour.add(key);
        }
    }
}


async function executeScenario(userId, scenario) {
    for (const sd of scenario.devices) {
        const device = sd.device;
        if (!device || device.id_user !== userId) continue;
```

```
        const targetState = sd.action === 'turn_on';

        if (targetState) {
            const source = await prisma.powerSource.findUnique({
                where: { id_user: userId },
            });
            const v = source?.grid_voltage ?? 0;
            if (ruleEngine.classifyVoltage(v) !== 'normal') continue;
        }

        if (!device.is_interruptible && device.relay_state && !targetState)
            continue;

        await prisma.device.update({
            where: { id_device: device.id_device },
            data:  { relay_state: targetState },
        });
    }
}
```

Д.5 Превентивна реакція на графік планових відключень

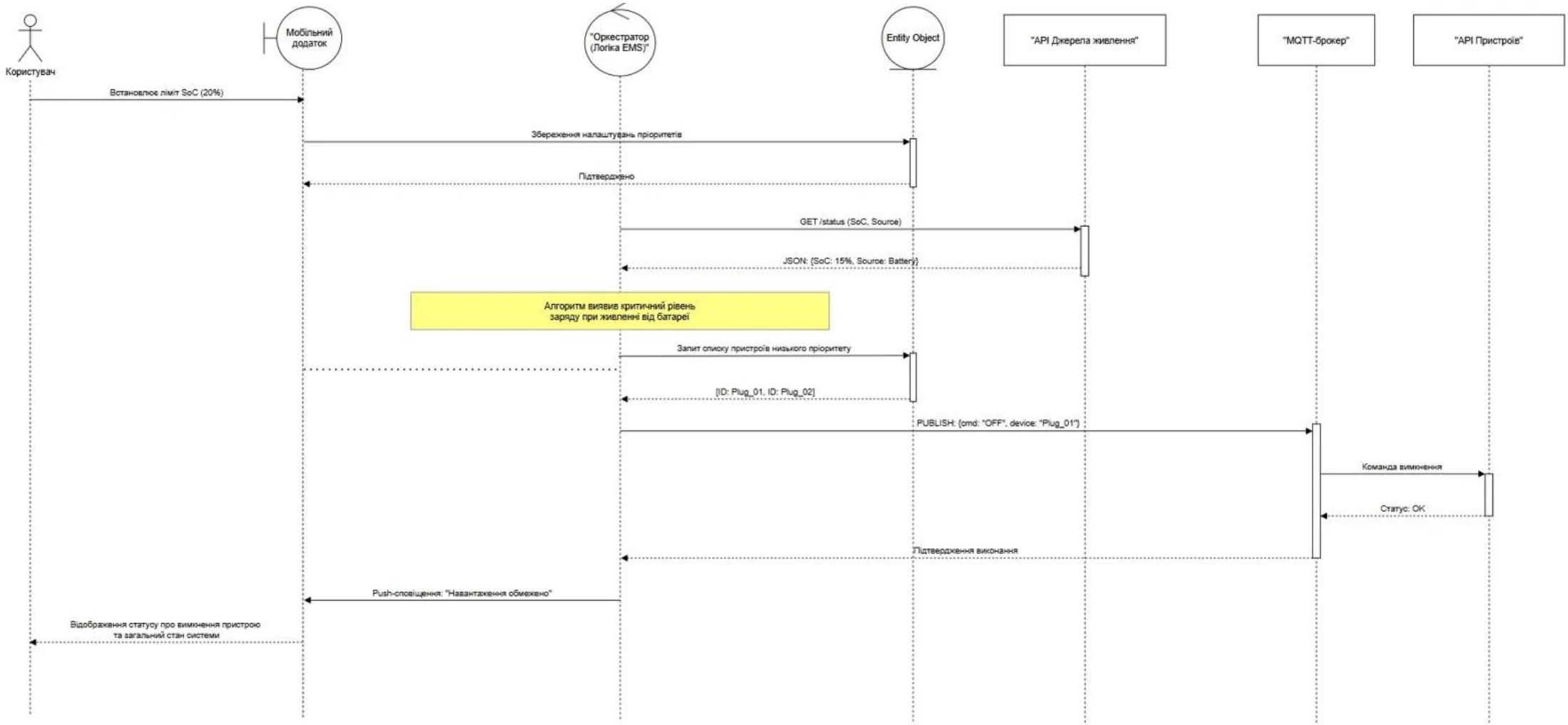
Інтеграція з API постачальника DTEK Yasno реалізована в модулі yasnoService.js. Функція fetchAndSaveScheduleForUser звертається до публічного API, парсить добовий розклад відключень для зазначеної черги споживача та зберігає інтервали в таблиці outage_schedule. Функція checkYasnoPreemptive з модуля scenarioEngine.js перевіряє наближення інтервалу відключення (за 15 хвилин) та ініціює превентивне скидання некритичних навантажень через ruleEngine.handleOutageStart. Булеве поле preemptive_fired запобігає повторному спрацюванню для одного інтервалу.

```
// Файл: services/yasnoService.js
const YASNO_URLS = [
    'https://api.yasno.com.ua/api/v1/pages/home/' +
        'schedule-turn-off-electricity',
    'https://yasno.com.ua/api/v1/electricity-outages-schedule/' +
        'get-current-schedule',
];

async function fetchAndSaveScheduleForUser(userId, zone) {
    for (const url of YASNO_URLS) {
        try {
            const response = await fetch(url);
            if (!response.ok) continue;
            const data = await response.json();

            const kyivSchedule = data?.components?.find(
                c => c.template_name ===
                    'electricity-outages-daily-schedule'
                    && c.city === 'kiev'
            );
            if (!kyivSchedule?.dailySchedule) return [];

            const today    = new Date().toISOString().split('T')[0];
```

```
            const tomorrow = new Date(Date.now() + 86400000)
                .toISOString().split('T')[0];
            const savedSlots = [];

            for (const [dateKey, dayData]
                    of Object.entries(kyivSchedule.dailySchedule)) {
                if (dateKey !== today && dateKey !== tomorrow) continue;
                const slots = dayData?.groups?.[zone] ?? [];

                await prisma.outageSchedule.deleteMany({
                    where: {
                        id_user: userId, date: dateKey,
                        zone, source: 'yasno',
                    },
                });

                for (const slot of slots) {
                    const record = await prisma.outageSchedule.create({
                        data: {
                            date:        dateKey,
                            start_time:  decimalToTime(slot.start),
                            end_time:    decimalToTime(slot.end),
                            outage_type: slot.type ?? 'DEFINITE_OUTAGE',
                            zone,
                            source:      'yasno',
                            preemptive_fired: false,
                            id_user:     userId,
                        },
                    });
                    savedSlots.push(record);
                }
            }
            return savedSlots;
        } catch { continue; }
    }
    return [];
}

// Файл: services/scenarioEngine.js – превентивна реакція
async function checkYasnoPreemptive(user) {
    const zone = user.yasno_zone ?? '1.1';
    const next = await yasnoService.getNextOutageNotFired(
        user.id_user, zone
    );
    if (!next) return;

    const PREEMPTIVE_MINUTES = 15;
    const shouldShed =
        next.minutes_until <= PREEMPTIVE_MINUTES &&
        next.minutes_until > 0 &&
        next.outage_type === 'DEFINITE_OUTAGE';

    if (shouldShed) {
        await ruleEngine.handleOutageStart(user.id_user);
        await prisma.outageSchedule.update({
            where: { id: next.id },
            data:  { preemptive_fired: true },
        });
    }
}
```

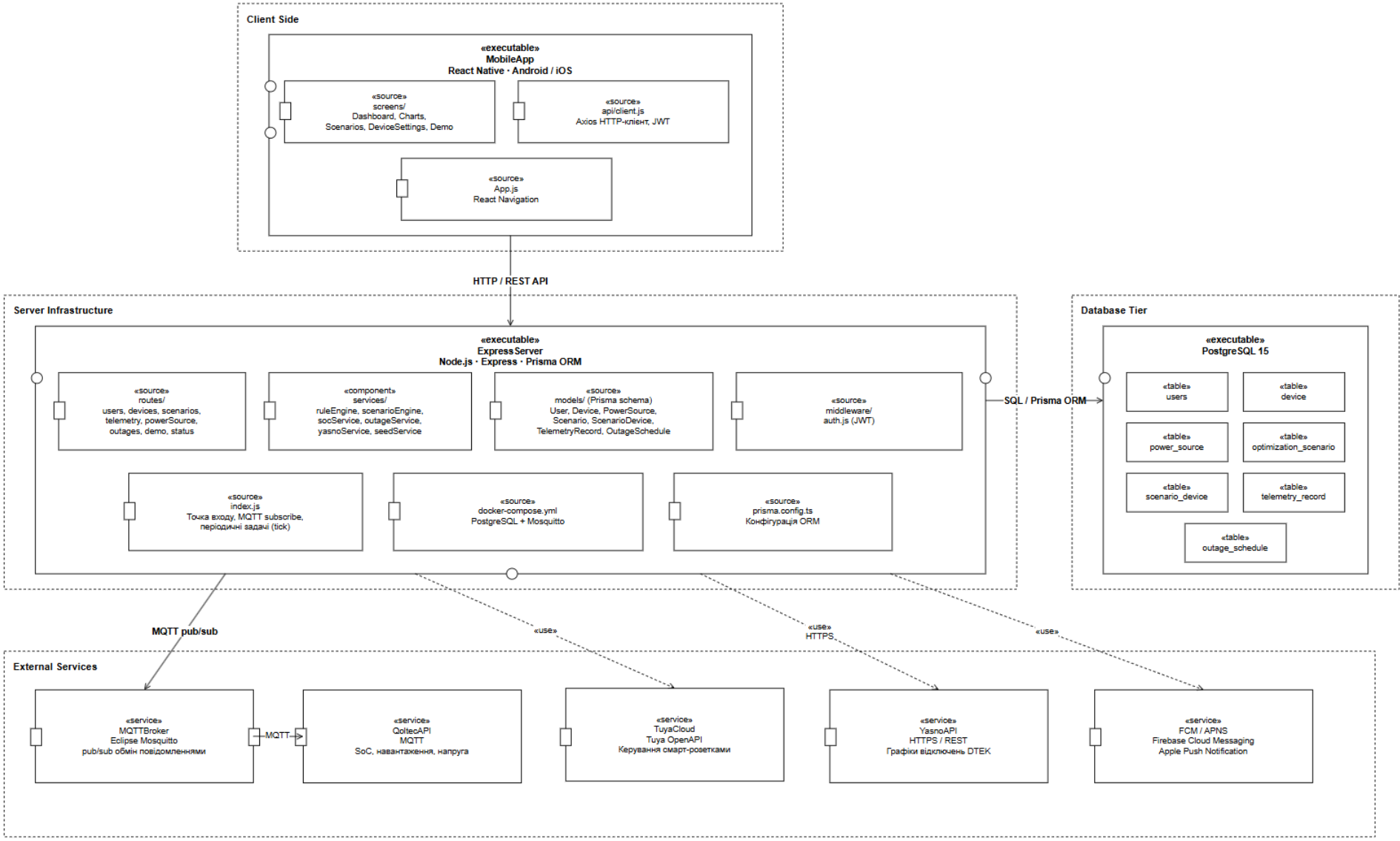
Додаток Е

Діаграма послідовності реакції системи на критичний рівень заряду батареї



Додаток Ж

Діаграма компонентів



Додаток И

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Ніколаєнко Михайло Станіславович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення системи моніторингу та оптимізації енергоспоживання в «розумному» будинку» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів штучного інтелекту (ШІ) зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [+] інструменти ШІ (вказати назву: Gemini, Claude) були використані для:
        - [+] перекладу іншомовних джерел;
        - [+] стилістичного редагування та перевірки граматики;
        - [+] допомоги у написанні/відлагодженні програмного коду;
        - [] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« »_____ 2026 р. _____ **Михайло НІКОЛАЄНКО**
(підпис)